UNIVERSIDADE D
COIMBRA

Pedro Miguel Sousa Leal

Navigation Solutions for Autonomous Mobile Robots using LIDAR

Dissertação no âmbito do Mestrado Integrado em Engenharia Física no ramo da Instrumentação
Orientada pelo Doutor Fernando António dos Santos Simões
Co-Orientada pelo Professor Doutor Jorge Afonso Cardoso Landeck
Apresentada ao Departamento de Física da Faculdade de Ciências e Tecnologia da Universidade de Coimbra

Setembro de 2021

Acknowledgements

Agradeço à Universidade de Coimbra e à Active Space Technologies pela oportunidade de fazer esta dissertação.

Obrigado aos meus orientadores pela ajuda durante este projecto. Um especial agradecimento ao Fernando Santos, que esteve presente durante todo o progresso, mostrando-se sempre disponível e dedicando muito do seu tempo a auxiliar-me na realização de esta dissertação.

Um enorme obrigado à minha familia, principalmente aos meus pais, pelos esforços que fazem para que eu possa ter uma vida cheia de oportunidades, pelo sustento, e pelo apoio. Em especial, também aos meus avós, que apesar das vidas duras, ainda hoje eu beneficío dos seus sacrifícios.

Agradeço aqui, em vão, aos meus cães, que podem não ler mas eu certifico-me que recebem festas.

Um agradecimento aos meus amigos de longa data, aos que me conhecem desde pequeno, e ainda hoje estão comigo. Vá para onde vá, sei que posso contar sempre convosco.

E por fim, aos meus amigos que conheci em Coimbra, tanto pelas aventuras nesta cidade como pelas noitadas no departamento.

Resumo

A área da robótica tem feito avanços monumentais e continua a fazê-lo. Durante décadas, robôs têm sido usados para realizar trabalho altamente especializado dentro do mundo industrial. Hoje em dia, enquanto os desenvolvimentos relativos a soluções autónomas continuam a materializar-se, esta tecnologia está a quebrar as fronteiras do mundo industrial e começam a ver-se robôs no nosso dia-a-dia. Desde carros autónomos a robôs de limpeza, a nossa interação com esta tecnologia está a tornar-se comum. Localização, mapeamento e planeamento de rota são conceitos importantes em robótica, e em sistemas de navegação. Executar localização e mapeamento em simultâneo, denominado de SLAM, é uma tarefa incrivelmente complexa, e, em aplicações em que há pessoas no local, estes sistemas têm de ser altamente robustos. A empresa Active Space Technologies está a explorar soluções para robôs móveis autônomos (AMRs) e esta dissertação pretende executar uma implementação baseada em open-source de um sistema de navegação, incluindo SLAM e planeamento de rota. Adicionalmente, um algoritmo de detecção de objectos (baseado em DeepLab) é integrado no sistema juntamente com outras funcionalidades.

Mais tarde, uma avaliação de seis soluções (SLAM Toolbox, Cartographer, RTAB-Map, HectorSLAM, Gmapping e LOAM) para a componente SLAM foi feita num ambiente real com dois sensores diferentes: o Intel Realsense L515 e o Velodyne VLP-16. Os resultados demonstram que ambos conseguem efectuar SLAM, sendo que o VLP-16 demonstrou excelentes resultados enquanto que o L515 necessitou de uma fonte robusta de odometria externa.

Finalmente, as soluções relativas à remoção de ruido detectado pela L515 em algumas circunstâncias (como uma fonte de luzes directa), e a segmentação de pessoas usando o algoritmo de detecção de objectos foram demonstradas.

Palavras-Chave – SLAM, Lidar, Robótica, Navegação, Detecção de Objectos, AMR

Abstract

The field of robotics has made monumental advancements and continues to do so. For decades, robots have been used for highly specialized work within the industrial world. Nowadays, as developments regarding autonomous solutions continue to materialize, this technology is breaking the industrial boundaries and robots are starting to be seen in our daily lives. From self-driving cars to cleaning robots, our interaction with this technology is slowly becoming common. Localization, mapping and path planning are major concepts in robotics, and any navigation system. Performing the first two simultaneously, known as SLAM, is an incredibly complex task, and when considering applications where there are people in the vicinity, these systems must be highly reliable. The company Active Space Technologies is exploring solutions regarding autonomous mobile robots (AMRs) and this dissertation aims to come up with an open-source implementation of a navigation system, including SLAM and path planning. Additionally, an object detection algorithm (using DeepLab) is integrated into the system as well as several other features.

Afterwards, an evaluation of six solutions (SLAM Toolbox, Cartographer, RTAB-Map, HectorSLAM, Gmapping and LOAM) for the SLAM component is carried out in a real world setting with two separate sensors: the Intel RealSense L515 lidar camera and the Velodyne VLP-16. The results reveal that both can effectively perform SLAM, with the VLP-16 showing excellent results while the L515 requires robust external odometry.

Lastly, solutions to remove systemic noise detected in the scans of the L515 under certain circumstances (such as direct light), as well as segmentation of people using the object detection algorithm are demonstrated.

Keywords – SLAM, Lidar, Robotics, Navigation, Object Detection, AMR

Acronyms

AGV - Automated Guided Vehicle

AMR - Autonomous Mobile Robot

DBN - Dynamic Bayesian Network

DOF - Degrees of Freedom

EKF - Extended Kalman Filter

GPS - Global Positioning System

ICP - Iterative Closest Point

IMU - Inertial Measurement Unit

LIDAR - Light Detection and Ranging

PF - Particle Filter

RANSAC - Random Sample Consensus

ROS - Robot Operative System

SLAM - Simultaneous Localization and Mapping

SURF - Speeded Up Robust Features

List of Figures

List of Tables

Table of Contents

Chapter 1

Introduction

Navigation systems focus on aiding an agent to travel within a physical setting. Nowadays, there is a constant development of this technology thanks to the rise of autonomous vehicles and robots, which require robust navigation systems to carry their tasks.

Celestial navigation allowed ancient civilizations to travel through the seas using instruments such as the sextant. Long range travels, with the aid of world maps, became frequent and it began the opening of the physical boundaries which limited what one person could experience in this world. Currently, the same thing is happening in the robotic domain, as we develop methods that allow robots to move and keep pushing the boundary of what a robot is capable of.

In this century, we are seeing automation break through the confines of the industrial world into our day-to-day thanks to the development of navigation systems.

Real world environments are uncertain and dynamic. Objects may change place between visits and there may be moving obstacles such as humans. Robots also rely on sensors and measurements that are affected by noise and in some cases, drift. Fully autonomous robots are required to have extremely robust localization and object detection for the safety of anyone in the vicinity, as well as for the robot's own integrity. These have three major basic components: mapping, localization and pathfinding.

Mapping with a known pose (position and orientation) has an uncertainty given by the range finder sensor. The same happens during localization with known predefined references. Navigation is relatively simple, assuming that the localization or the map

are known in advance. However, the same can not be said when travelling through an unknown environment while the pose of the robot is not certain.

This problem led to simultaneous localization and mapping (known as SLAM), which corresponds to the difficult task of having a robot to move through an unknown environment, while relying on sensors to both build a map and localize itself on it. SLAM solutions approach the uncertainty in the measurements explicitly and implement strategies that aim to reduce the pose error. These approaches entail having multiple and diverse sensors, calculating estimates and applying error minimization techniques. Most algorithms are based in three archetypes, two of which are implementations of the Recursive Bayes filter, Extended Kalman Filter (EKF) SLAM and FastSLAM. These two recursively update a model by comparing it with current measurements. The third archetype is called a graph-based approach, which relies in a graphical representation of poses and in error minimization processes when revisiting places.

After considering mapping and localization, the last step that gives autonomy to a robot is a guidance system, which allows the robot to execute path finding when given a goal, including exploring yet unrevealed areas. In most cases, path finding algorithms aim to obtain the shortest path to the objective after considering the constraints present in the environment.

1.1 Motivation

Recently, the prospect of self-driving vehicles and service robots, which have been represented in our culture for decades, is coming close to be a reality. The development of autonomous robots that are ready to face the unpredictability of the real world has been immense and, nowadays, we can see the fruits of it in driver-assistance systems, house cleaning robots and industry 4.0. The ability of a robot to perform navigation by itself in a setting is fundamental for it to perform its higher goal, thus contributing to improve localization and mapping implementations is also important.

This work was done in a partnership with Active Space Technologies. The company's branch Active Space Automation focuses in automated guided vehicles and

autonomous mobile robots manufacturing for industrial logistics. In parallel, since the beginning of the current Covid-19 pandemic, autonomous robots are being explored as an option to perform disinfection of spaces by travelling through buildings while carrying an UV-C light. Both in industrial situations as well as during disinfection of public spaces, highly dynamic environments are to be expected with goods, furniture and humans changing place. Particularly in the case of disinfection using UV-C light, the system is required to quickly turn the light off in case humans are present in the vicinity, because its radiation is harmful to living organisms. These factors led the company to invest in the development of robust navigation systems capable of mapping buildings and localizing themselves to perform these tasks.

As for my personal motivation, robotics has always been an interest of mine and this work represents an opportunity to learn and explore new skills. Furthermore, the proposed project was presented as very open-ended, with the core objective being a SLAM system, but the further possibility of adding functionalities constituted a chance to explore other branches such as artificial intelligence for object detection, which only increased my interest.

1.2 Goals

The intention behind this work is to implement a navigation system to be applied to autonomous mobile robots (AMRs) equipped with lidar. After the core system is able to perform simultaneous mapping and localization, and navigate through its map, functionalities such as the detection of objects and humans may be added as well. The system must be prepared for highly dynamic surroundings as the AMRs are meant to roam indoor environments, which are being utilized by humans and may suffer changes in their layout.

Open source methods are envisioned and the following sensors are to be used: Intel RealSense L515 (comprising a lidar, IMU and camera) as well as the Velodyne's VLP-16.

1.3 Document Structure

Chapter 2 is dedicated to presenting basic concepts in robotics, in particular simultaneous mapping and navigation, followed by a comparison of the three main paradigms and of the state of the art open-source SLAM implementations.

Chapter 3 describes the implementation developed in this work, from the core SLAM systems to the object detection, all integrated in ROS (Robot Operative System). Lastly, the conditions for the tests done to verify the SLAM approaches are explained.

Chapter 4 reports the results obtained after a test in a real world setting.

Chapter 5 contains the conclusion of the thesis and also describes the future work.

Chapter 2

Indoors autonomous navigation background

In this chapter the underlying theory associated with robotics is explored, as well as more specific concepts related to mapping and localization. The chapter closes with an overview of several papers, which compare various open-source methods that aim to tackle the SLAM component of navigation systems.

2.1 Probabilities in Sensors

Robotic systems rely on sensors, which are intrinsically noisy and can be affected by drift, limiting the confidence in the data. In order to have robust systems, noise needs to be addressed and taken into account. Wheel encoders are used in many localization systems, and because of slippage, these accumulate errors, causing uncertainties in the position over time. The same happens when using accelerometers and gyroscopes. These also accumulate drift and, after a certain period of time, blindly trusting their measurements eventually leads to failure.

A widely used solution is employing the recursive Bayesian filter as a tool to contain uncertainty by fusing various sensors with the goal of reducing the inevitable increase of uncertainty of the evolving state.

In this chapter, we define x_t as the state vector to be estimated, which might be representing the current pose (position and orientation). Similarly, x_{t-1} defines the previous state; we are defining each time step t as an iteration of the filter. Subsequently, u_t is defined as the "control input" and it can represent different things depending on the implementation. In some cases, it may be referring to a control

given to the robot. The system might accept a command such as to move forward one meter, which can then be used to predict its position. In other implementations, u_t might be referring to the data from odometry such as wheel encoders, or even a combination of several other sources of odometry including visual odometry or scan matching techniques. Finally, z_t is known as the observation and also depends on the implementation. In case the goal is to do localization, this measurement may be a GPS estimate. In this scenario, the drift experienced by odometry is contained by fusing its data with the GPS data. And yet, the robot's estimated pose is continuous and resistant to short losses of signal. In case of SLAM, it refers to the measurements of landmarks or obstacles obtained using range finder sensors or cameras.

2.1.1 Recursive Bayes Filter

Furthermore, the notation $bel(x_t)$ is used, its called "belief" and represents the system's estimates. It can be expressed by: $bel(x_t) = p(x_t|z_{1:t}, u_{1:t})$. The recursive Bayes filter can now be stated as[4]:

$$\overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1}) bel(x_{t-1}) dx_{t-1} \tag{2.1}$$

and

$$bel(x_t) = \eta\, p(z_t|x_t) \overline{bel}(x_t), \tag{2.2}$$

where η is a normalization factor. The state x_t is estimated recursively, taking into account both the control input u_t and the measurement z_t. The x_t is an unobserved Markov process, as it can be seen in (2.1)-(2.2), where only the immediate previous step is used for the current estimation. As for z_t, it is a hidden Markov process, it depends on x_t and it has a part in estimating the state.

The first implementation to be discussed of this filter is the Kalman Filter.

2.1.2 Kalman Filter

The Kalman Filter is an implementation of the Bayes filter. It can be used solely to improve odometry by fusing various sensors, or set as the basis for a whole SLAM implementation as explained in Section 2.3.2.

Three requirements must be met to implement the Kalman Filter. These assure that the relevant distributions stay Gaussian. First, the initial belief must follow a normal distribution. Second, this filter requires $p(x_t|u_t, x_{t-1})$ to be linear with a Gaussian noise meaning that

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t, \tag{2.3}$$

where the term $A_t x_{t-1}$ conveys how the system naturally evolves (without noise or control input). The A_t is an n-by-n matrix, where n is the dimensionality of the state vector. The $B_t u_t$ indicates how the control input affects the system. The B_t is a n-by-l matrix where l is the dimensionality of the control space; this means that the matrix also inserts the control into the state space. And ϵ_t is the process noise, i.e., a Gaussian vector with mean zero and covariance R_t. The process noise represents how the system uncertainty naturally evolves with time. And thirdly, the filter also requires the same linear assumption for the measurement probability $p(z_t|x_t)$, which must be linear in its arguments, resulting in

$$z_t = C_t x_t + \delta_t, \tag{2.4}$$

where $C_t x_t$ represents the expected observation given the predicted state of x_t. The C_t is the measurement model, a matrix with size k-by-n, where k is the dimesionality of the measurement space. The δ_t is the measurement noise with mean zero and covariance Q_t.

These requirements make the Kalman Filter unusable in many applications as sensor noise may not follow a Gaussian distribution.

Algorithm 1: The Kalman Filter

Function `Kalman Filter(`μ_{t-1}, Σ_{t-1},u_t, z_t`):`

1 $\overline{\mu} = A_t\mu_{t-1} + B_tu_t$

2 $\overline{\Sigma}_t = A_t\Sigma_{t-1}A_t^T + R_t$

3 $K_t = \overline{\Sigma}_tC_t^T(C_t\overline{\Sigma}_tC_t^T + Q_t)^{-1}$

4 $\mu_t = \overline{\mu} + K_t(z_t - C_t\overline{\mu}_t)$

5 $\Sigma_t = (I - K_tC_t)\overline{\Sigma}_t$

return μ_t, Σ_t;

The prediction part is referring to the lines 1 and 2, where the algorithm attempts to predict the state by taking into account the control u_t. Line 2 takes into account how motion increases the uncertainty, where $\overline{\sum}_t$ is the error covariance of the current prediction, which depends on our confidence in the control input.

The correction part refers to lines 3, 4 and 5. In line 3, the Kalman gain (K_t) is computed; this variable reflects how confident one is in the observations and in the predictions, in order to obtain the new belief. The mean μ_t is then updated as well as the covariance matrix.

The Kalman filter can be visualized in the figure 2.1: In a) a prediction is stated; b) introduces in bold the measurement with its uncertainty; c) the distribution after fusing both prediction and correction with the weights calculated in step 3; d) the new prediction is shown; e) the same happens as in b); f) the estimate is again calculated.

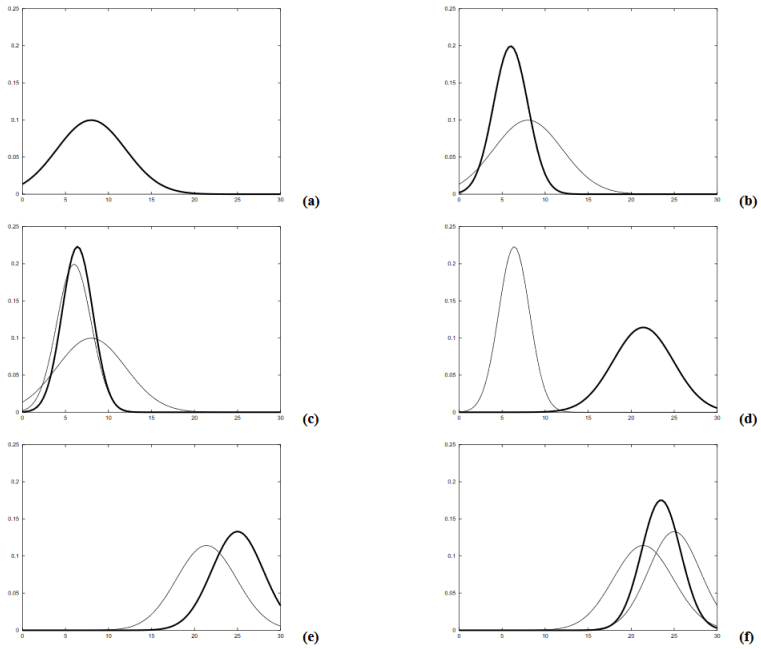


Figure 2.1: Kalman filter illustrated[4].

2.1.3 Extended Kalman Filter

In real world situations, most times we can not assume that the functions are linear or that the noise is Gaussian. We now state that

$$x_t = g(x_{t-1}, u_t) + \epsilon_t \tag{2.5}$$

and

$$z_t = h(x_t, z_t) + \delta_t, \tag{2.6}$$

where both g and h are non-linear functions that must be linearized, and to do so a first order Taylor expansion is done around a linearization point

$$g(\mu_t, x_{t-1}) \approx g(\mu_t, u_{t-1}) + G_t(x_{t-1} - \mu_{t-1}),\ with\ G_t = g'(u_t, \mu_{t-1}), \tag{2.7}$$

and

$$h(x_t) \approx h(\overline{\mu}_t) + H_t(x_t - \overline{\mu}_t),\ with\ H_t = h'(\overline{\mu}_t). \tag{2.8}$$

Now the algorithm takes the form

Algorithm 2: The Extended Kalman Filter

Function `Extended Kalman Filter(`μ_{t-1}, Σ_{t-1},u_t, z_t`):`

$\overline{\mu} = g(u_t, \mu_{t-1})$

$\overline{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$

$K_t = \overline{\Sigma}_t H_t^T (H_t \overline{\Sigma}_t H_t^T + Q_t)^{-1}$

$\mu_t = \overline{\mu} + K_t(z_t - h(\overline{\mu}_t))$

$\Sigma_t = (I - K_t H_t)\overline{\Sigma}$

return μ_t, Σ_t;

2.2 Localization and Mapping

Localization can be divided in two types: global localization and local localization[26]. Global localization methods localize a robot's pose relative to the map without previous knowledge of its position, while local localization requires the initial position of the robot and is focused in containing odometry error.

As examples of global localization techniques, there is particle filter localization[8] (also known as Monte Carlo Localization). In its simplest implementation, it starts with particles uniformly distributed throughout the map. With each observation the measurements are compared with the predicted state (obtained using an odometry model) for each particle, followed by importance weighting and resampling. Over time the particle cloud should converge to the robot's localization. As for local localization techniques, these exist so we always know where the robot is within a interval of uncertainty and provide initial estimates for a SLAM algorithm. One example is having scan matching algorithms using lidar point clouds (explored in Section 3.3) being fused with an IMU (inertial measurement unit) by an Extended Kalman Filter.

Mapping is mostly done in two ways. Vast environments may require a lot of space for storage, so there are methods to process the measurements for a lighter map.

Gridmaps split the map into subregions or cells c_i, and each cell has an associated probability of being occupied $p(c_i|x_{1:t}, z_{1:t})$ based on every measurement up until

t. This is not found in every case, only in probabilistic grid maps, in some other approaches cells only have the values 1 or 0 (occupied or free). Grid maps are also able to differentiate a free area from an unknown area. These can be 2D or 3D, and in vast environments it may present a computational resource problem, especially in 3D. To help solving this issue, some 3D map techniques use octrees. An octree divides data into nodes that represent a cubic volume. Each node may be divided into eight smaller cubic volumes and so on, until it reaches a minimum resolution. Octrees reduce computational requirements by allowing bigger spaces with the same values to be represented by only one node instead of several smaller nodes. To represent more complex structures, one can use the node subdivisions to represent data. This allows various spatial resolutions that are increased as needed.

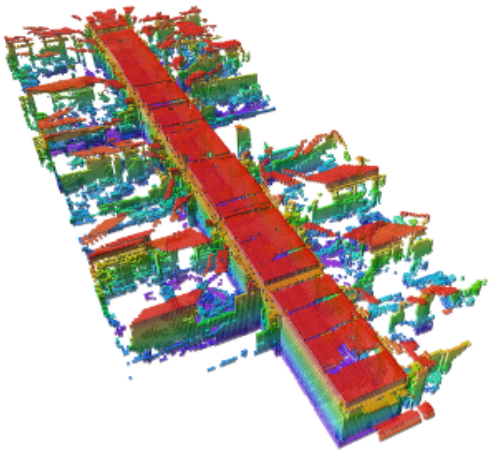


Figure 2.2: An indoors 3D map[12].

Feature-based maps only represent landmarks or features that are extracted from the camera or lidar data. The map is represented by the state vector $M = \{m_1, m_2, ..., m_n\}$ that contains the pose of each landmark. Using features results in only using a subset of the data coming from range finders or cameras being used to perform mapping, reducing the memory resources needed. There are several processes to select and extract features, e.g., these can be corners in laser scans or point clouds.

2.3 Simultaneous Localization and Mapping

SLAM designates the problem of being able to have an agent that is capable to both localize itself in an unknown environment while at the same time creating a representative map. A common setting is a robot that measures its own motion using sensors (e.g., wheel encoders and inertial measurement units), which is known as odometry paired with other sensors such as cameras and range finder sensors to obtain data regarding the environment. The problem lies in the fact that robot motion leads to an increase in the pose uncertainty since sensors are prone to noise and experience drift (e.g., accelerometers, gyroscopes and encoders).

Accomplishing accurate localization is not trivial since we do not have a static predefined map for reference. Creating a map while using it to localize is a difficult task, since the measurements from sensors, that are being used to build the map, comprise non-negligible pose uncertainty accumulated up until that point. This leads to the fact that inaccurate localization results in imperfect mapping and vice-versa. SLAM algorithms mainly focus on applying some form of the Bayes Filter or applying error minimization techniques in pose graphs while combining several sources of information for reduced uncertainty.

There are three base methods to approach this problem. The first is EKF SLAM, that uses the Extended Kalman Filter to implement SLAM. The second is using a particle filter as an alternative to implement the Bayes filter. The third is a graph-based approach.

Loop Closure

Loop closure is a major concept in SLAM, referring to the system's ability to recognize previously visited places. When a robot moves through the environment, uncertainty in both pose and map increase over time. Revisiting a location gives the necessary information to reduce the accumulated drift, which increases the map accuracy by reducing the uncertainty built up between visits. In case of graph-based approaches, a dedicated process called pose graph optimization is employed. However, inaccurate detection of a loop closure taints both the map and pose accuracy by misguiding the algorithm, in particular, the graph optimization process (in the case of graph-based SLAM).

Data Association

To implement SLAM there is a need to identify which landmark is which. The application of loop closure is only as good as the data association. It relates to the capacity of identifying and linking measurements of the same object or place, giving the awareness of revisiting locations and keeping track of landmarks and scenes.

2.3.1 Motion and observation models in SLAM

The motion model is written as $x = g(u_t, x_{t-1}) + \epsilon_t$ and represents the prediction and its uncertainty on the robot's movement when using only odometry and controls. In other words, it expresses the probability cloud where the robot hopefully is, before taking into account range finder information.

The observation model is expressed by $z_t = h(x_t, m) + \delta_t$, and it conveys how the measurements should look like, when considering the previously predicted pose and the map. This model is to be compared with the actual measurements. The opposite is called the inverse observation model ($m = f(x_t, z_t) + \delta_t$), which is used to insert a landmark into the map.

2.3.2 EKF SLAM

The Kalman Filter is explained in Subsection 2.1.2. Here we briefly cover how to perform SLAM using it. Correct data association is being assumed.

The state space is expressed as

$$\mu = (p_{1:t}, m_{1,x}, m_{1,y}, ..., m_{M,x}, m_{M,y})^T, \tag{2.9}$$

where it contains the current pose $p = (x, y, \theta)^T$,

with the covariance matrix

$$\begin{pmatrix} \Sigma_{p_t,p_t} & \Sigma_{p_t,m_1} & \dots & \Sigma_{p_t,m_N} \\ \Sigma_{m_1,p_t} & \Sigma_{m_1,m_1} & \dots & \Sigma_{m_1,m_N} \\ \vdots & \vdots & \ddots & \vdots \\ \Sigma_{m_N,p_t} & \Sigma_{m_N,m_1} & \dots & \Sigma_{m_N,m_N} \end{pmatrix}. \tag{2.10}$$

In a SLAM implementation using the EKF, the prediction is the motion model, and the correction is the observation model.

The prediction takes the odometry and the current state to update the estimate according to $g(\mu_t, x_{t-1})$ as covered in subsection 2.1.2; its covariance is also updated by taking into account the process noise.

The correction step is more nuanced. Landmarks have to be initialized the first time they are observed. To do so, observations must be converted into the state space. The current pose estimate and the current measurement are used to do so.

There is a need to identify and link landmarks from different measurements. Wrongful data associations lead to errors, especially since the EKF is not capable of keeping multiple possible associations (no multimodality).

Here, function $h(\overline{\mu}_t)$ takes the updated pose and predicts how, according to the current understanding of the landmarks' layout, the landmarks should be observed. This is done using the process of ray casting.

2.3.3 FastSLAM and FastSLAM 2.0

In the case of FastSLAM, each particle has its associated position but also the associated perceived environment. This results in a high-dimensional state space

$$x = (x_{1:t}, m_{1,x}, m_{1,y}, ..., m_{M,x}, m_{M,y})^T. \tag{2.11}$$

The posterior, which is the estimate after taking into account odometry and observations, is expressed as:

$$p(x_{0:t}, m_{1:M}|z_{1:t}, u_{1:t}). \tag{2.12}$$

Given that complexity increases exponentially with the dimensions of the state space, particle filter performance may not allow an online SLAM application with many landmarks. FastSLAM is a Rao-Blackwellized particle filter, so the posterior is divided in two, the path posterior and the map posterior using the definition of conditional probability,

$$p(x_{0:t}, m_{1:M}|z_{1:t}, u_{1:t}) = p(x_{0:t}|z_{1:t}, u_{1:t})p(m_{1:M}|z_{1:t}, x_{0:t}, u_t). \tag{2.13}$$

This way, $p(m_{1:M}|z_{1:t}, x_{0:t}, u_t)$ (landmark estimation) is computed using EKFs for each particle. Meanwhile, the robot's path is estimated by the particle filter.

Futhermore, an assumption is made where each possible pose given by the particle filter is treated as a known variable. Since the pose of each particle is assumed to be known, each individual landmark is conditionally independent from each other as one can easily see in a DBN (Dynamic Bayesian Network), where we can visually see how variables are related (Figure 2.3).

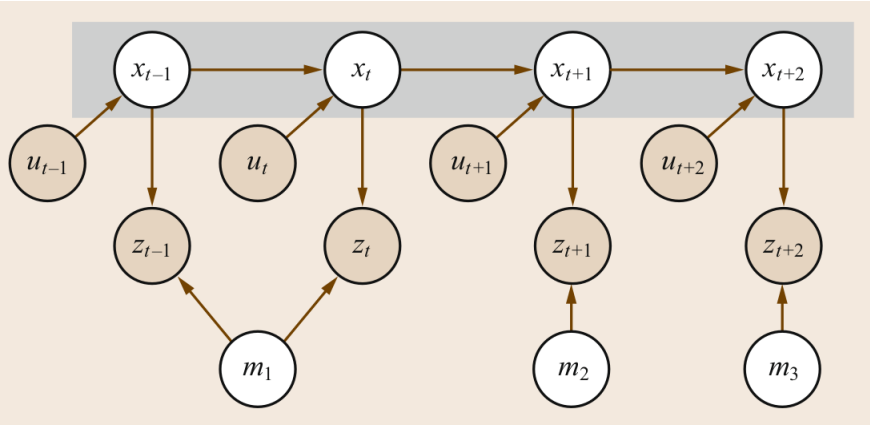


Figure 2.3: Dynamic Bayes Network[3].

If the pose is known, landmarks are independent from each other (landmark position is a function of both the robot's pose and measurements), allowing the use of an EKF for each observed landmark instead of an higher dimensional EKF

$$p(x_{0:t}|z_{1:t}, u_{1:t})\, p(m_{1:M}|z_{1:t}, x_{0:t}) = p(x_{0:t}|z_{1:t}, u_{1:t}) \prod_{i=1}^{M} p(m_i|z_{1:t}, x_{0:t}). \tag{2.14}$$

Mapping is then done utilizing a 2-Dimensional (if only x and y coordinates are used for landmarks) Extended Kalman filter for each landmark within range of the sensors, resulting in M $\times$ K (where M is the number of particles and K the number of landmarks) parallel Kalman filters. Having several low dimensional EKFs is more efficient than having one high dimensional EKF.

The algorithm repeats the following steps:

1. Use each particle to sample from the motion model $x_t \sim p(x_t|u_t, x_{t-1})$;
2. Use the measurement data to update landmarks using EKF;
3. Consider how measurements fit with the current understanding of the map to do importance weighting;
4. Taking into account the weight of each particle, perform resampling with replacement on the current particle set.

As mentioned in step 3 and for us to know which particles are more accurate, after applying the movement on the particles by sampling from the motion model, we must check how the current observations fit with the current predicted observations. Particles whose maps are coherent with the current observations will have a higher weight value

$$w_t^{[i]} = \frac{p(x_t|z_t, u_t)}{p(x|z_{t-1}, u_t)}. \tag{2.15}$$

Here the target distribution is divided by the proposal distribution. This leads to[28]

$$w_t^{[i]} \approx \eta\, |2\pi Q_t^{[i]}|^{-\frac{1}{2}}\, exp\Big\{ -\frac{1}{2}(z_t - \hat{z}_t)^T Q_t^{[i]-1}(z_t - \hat{z}_t)\Big\}. \tag{2.16}$$

In this form, it is clear to see that particles whose current observations fit with their model of the map ($\hat{z}_t$ is the predicted observation) have higher weight values, leading to a higher chance of surviving the resampling step.

Unlike EKF SLAM, this implementation makes it possible to have multiple hypotheses for data association.

In the real world, odometry sources can be very noisy and have an extremely flat distribution. This results in the proposal distribution of particles being scattered with a high percentage of bad predictions and most of them being lost in the resampling step.

FastSLAM 2.0 improves the previous iteration of FastSLAM by incorporating the observation in the proposal distribution. The optimized proposal distribution $p(x_t|x_{t-1}, u_t, z_t)$ is the product of the previous proposal distribution and the probability of the measurement z_t. As one can see in Figure 2.4, while taking into account only the control input results in a flat distribution, the observation model is extremely precise.

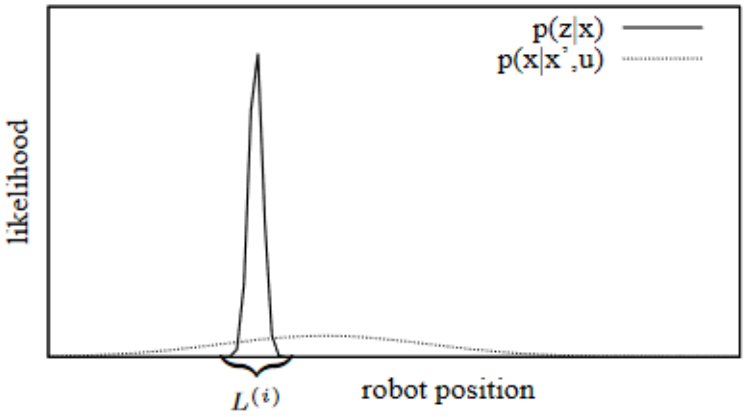


Figure 2.4: Comparison between odometry and laser range distributions[28].

Introducing the observations into the prediction step leads to a more accurate proposal distribution, and therefore more particles with higher weights, resulting in a higher variety of particles being kept over time (reducing particle deprivation). Consequently, there are a higher number of differently valid hypotheses for the trajectory and map.

2.3.4 Graph-based

The graph based approach[9] builds a pose graph, where each pose at a time t is represented by a node. These nodes are interconnected by edges that represent a constraint between the two poses. A new pose can be connected to the immediately previous pose by any type of odometry, for example, wheel encoders. These nodes can also be connected by what is mentioned as "virtual observations". These edges result of nodes that are not directly connected (such as x_t and x_{t+1}) but because they observe the same place in the environment.

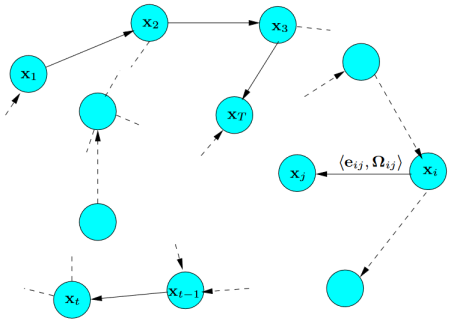


Figure 2.5: A demonstation of a pose graph

$$X^T = (x_1, \quad x_2, \quad x_3, \quad ..., \quad x_i), \tag{2.17}$$

where x_i is the pose of node i.

Direct edges are given by the system's odometry. These may come from wheel encoders, IMU, visual odometry or matching two sucessive lidar scans. The covariance matrix from this transformation is expected to have higher values in cases where the system's odometry relies on only one sensor. In addition, wheel encoders and IMUs tend to give less precise results versus ICP (Iterative Closest Point) scan matching with laser scans.

As for **virtual measurements**, these reflect loop closures. These measurements are the result of algorithms like ICP, or possibly algorithms that match RGB color images. The ICP algorithm is mentioned and explained in Section 3.3. Scan matching of measurements taken on nodes under a certain threshold of distance between

both poses is done; the two point clouds are compared to identify if they represent the same place, then the rigid transformation that overlaps both scans correctly is obtained. Ideally, this will be the transformation between the poses and will be used to create a virtual edge.

This way, when the robot passes through a previously visited location, it observes the former position when the older scan of the same place was taken. The difference between the position of both nodes as seen in the graph ($\hat{z}_{ij}$) and the transformation given by the scan matching algorithm (z_{ij}) gives the error that needs to be minimized, according to

$$e_{ij}(x_i, x_j) = z_{ij} - \hat{z}_{ij}(x_i, x_j). \tag{2.18}$$

Figure 2.6 presents a small graph where a previous pose (x_j) is being observed from the current pose x_i is shown.

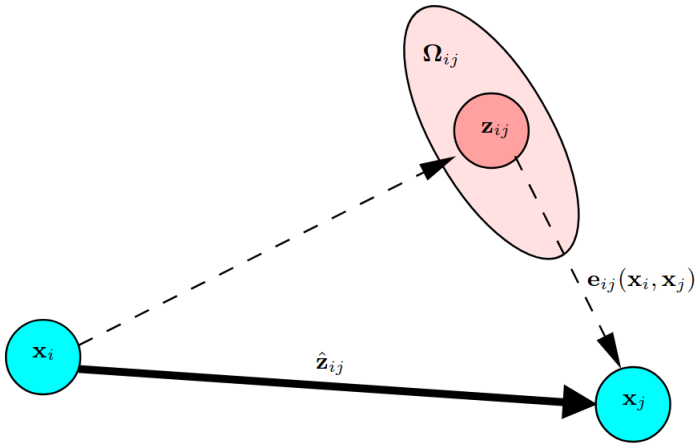


Figure 2.6: A pose graph when a previously seen location is revisited[9].

The following step is to minimize this error, e_{ij}, using the Least Squares method,

$$x^* = \underset{x}{\operatorname{argmin}} \sum_{\langle i,j \rangle \epsilon C} e_{ij}^T \Omega_{ij} e_{ij}. \tag{2.19}$$

In this case, most algorithms use the Gauss-Newton or Levenberg–Marquardt method because we are dealing with a non-linear problem. Here, it can be seen that the meas-

urements of which we are more confident about have a higher influence thanks to using the information matrix (Covariance matrix is Ω^{-1}).

2.4 Evaluation of the SLAM paradigms

The **EKF SLAM** has an order of computational complexity $O(k^2)$, where k is the number of landmarks, since it keeps a covariance matrix containing information on every single landmark seen. This fact creates a technical limit on its scalability, specially considering how an online SLAM application needs to respect time constrains to function properly. The approximation to a Gaussian distribution comes with costs, including the inability to have a multimodal distribution, which results in mistakes when relying on inaccurate data association.

The **FastSLAM** computational complexity is $O(Mlog(k))$, where M is the number of particles. It increases linearly with the particle number and logarithmically with the number of landmarks. FastSLAM has a higher scalability than the paradigm mentioned above. The particle filter is not limited to a certain distribution (only the landmark pose is approximated to a Gaussian distribution, not the poses), and can take the form of any model and keep its multimodality. Therefore, it is able to keep information about multiple possible data associations. Problems arise when repeated visits to already mapped places happen, which introduces particle depletion and requires adaptive sampling measures.

Lastly, **Graph-based** approaches' complexity is roughly linear with the edges, E, of the graph $O(E)$, which makes it the best option for large-scale environments. This happens since not all nodes are interconnected, and each node is only linked to few others. However, if the path of the robot is extensive, the optimization process will also be more expensive to compute since there are more nodes. The method is sensitive to inaccurate data associations that lead to the pose graph being wrongly optimized.

2.5 Comparison among ROS-based SLAM implementations

There are dozens of different open-source implementations of SLAM that can be found. We are interested in looking into implementations that mainly use lidar but may use IMU and RGB data, either in 2D or 3D. The ROS (Robot Operating System) has been used for a lot of open-source development of SLAM implementations.

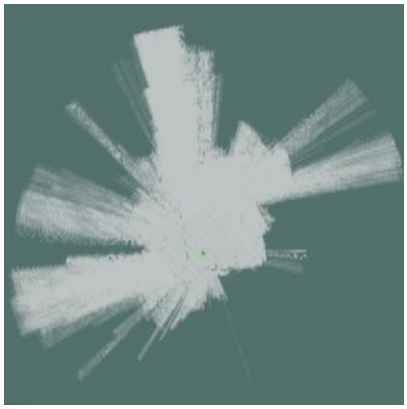
Santos et al. (2013)[25] discusses five methods that use lidar data: Hector-SLAM, which heavily relies on scan matching for 2D mapping, Gmapping, a Rao-Blackwellized PF-based SLAM, KartoSLAM, a graph-based SLAM, CoreSLAM, that is a very short PF-based algorithm, and LagoSLAM, another graph-based method. Both simulations and real world experiments are made using a Hokuyo URG-04LX-UG01, which is a 2D lidar (270° field of view and up to 5.6 m of range), where the resulting maps are compared to the ground truth map.

Gmapping, Hector-SLAM and KartoSLAM are consistently shown to result in the most accurate maps in both simulations and real world experiments, with KartoSLAM being the most accurate in the real world setting.

Afanasyev et al. (2017)[1] presents tests made in indoor homogeneous environments. The methods used are paired with lidar, cameras, stereo and RGB-Depth cameras. Several SLAM methods' trajectory were evaluated, with the relevant methods to this work being Hector-SLAM (paired with a Hokuyo UTM-30LX, a 2D lidar with 30 meters of maximum range) and RTAB-Map (which in this case uses pointclouds given by RGB-D sensors).

Hector-SLAM offers a more accurate trajectory with an average deviation of 0.11 m, while RTAB-Map showed a 0.42 m average deviation. However, the route performed in this test was short; considering how Hector-SLAM does not perform loop closure and RTAB-MAP is a graph-based approach, results would likely differ in courses with a higher scale. It is also worth noting that different sensors were selected, with the lidar used by Hector-SLAM being more accurate and with a wider view than the RGB-D camera.

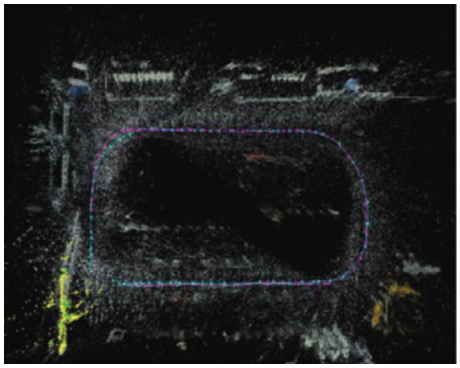
Lastly, Filipenko et al. (2018)[7] performs real world tests with Gmapping, Hector-SLAM and Cartographer with a Hokuyo UTM-30LX. Regarding the resulting maps, both HectorSLAM[13] and Cartographer[11] ensured good results. Surprisingly, when considering previous benchmarkings, Gmapping built a completely inaccurate map (Figure 2.7).



(a) Map by Gmapping

(b) Map by HectorSLAM

(c) Map by Cartographer



(d) Map by RTAB-Map

Figure 2.7: Maps obtained by the different algorithms[7].

Concerning trajectories, HectorSLAM and Cartographer provided the best results followed by RTAB-MAP (Figure 2.8).

System	RMSE (m)	Mean (m)	Median (m)	Std. (m)	Min (m)	Max (m)
Cartographer	0.024	0.017	0.013	0.021	0.001	0.07
LSD SLAM	0.301	0.277	0.262	0.117	0.08	0.553
ORB SLAM (mono)	0.166	0.159	0.164	0.047	0.047	0.257
DSO	0.459	0.403	0.419	0.219	0.007	0.764
ZEDfu	0.726	0.631	0.692	0.358	0.002	1.323
RTAB map	0.163	0.138	0.110	0.085	0.004	0.349
ORB SLAM (stereo)	0.190	0.151	0.102	0.115	0.004	0.414
S-PTAM (no loop cl.)	0.338	0.268	0.244	0.206	0.001	0.768
S-PTAM (loop cl.)	0.295	0.257	0.242	0.145	0.006	1.119

Figure 2.8: Differences in trajectory[7].

As seen above, HectorSLAM gives excellent results on both mapping and trajectory with the downside of only providing 2D maps, but since it does not perform loop closure, a solution might be to construct a map in parallel utilizing trajectory information with Octomap[12]. A known problem of any approach that utilizes solely scan matching (as in Hector-SLAM) arises in situations as travelling through corridors (with few to none features other than parallel walls), where the longitudinal uncertainty increases. Yagfarov et al. (2018)[29] tests increasing the speed of the platform robot and demonstrates it to have a harsher impact on HectorSLAM when compared to Gmapping and Cartographer.

The equivalent of featureless corridors may happen in homogeneous environments regarding color when using RGB dependent algorithms such as the loop closure detection method used by RTAB-Map.

Chapter 3

Implementation

The navigational system takes RGB, IMU and lidar data as inputs and outputs the robot's 2D pose, a 2D or 3D occupancy map and both a rotation and translation vector to be used by a movement control system after selecting a destination. The inverse kinematic equations applied are intended for mecanum wheels. Several other functionalities are explored regarding people detection and noise removal in order to increase robustness.

3.1 Sensors

One of the sensors used in the system is an Intel RealSense L515, which contains a lidar system, RGB camera and a 6DOF IMU comprising a gyroscope and an accelerometer. The other is a Velodyne VLP-16, a powerful lidar.

3.1.1 Intel RealSense L515

The system creates a 3D pointcloud containing 307200 (640×480) points, with a FOV of 70°×55° and a range of 0.3 to 9 m stated in the product's datasheet. The update frequency is 30 Hz. Empirical data regarding range is shown in Table 3.1.

Figure 3.1: The Intel RealSense L515.

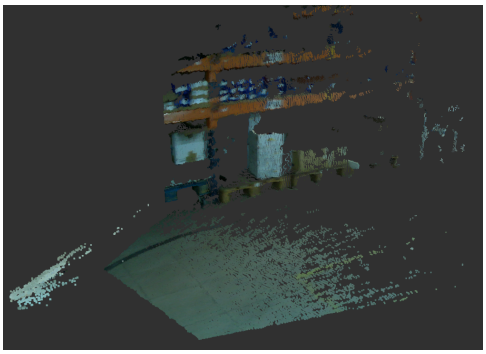
Real distance (m)	Mean measured distance (m)	Standard deviation (m)
0.5	0.4999	0.0009
2.5	2.5003	0.0007
5	5.0274	0.0096
7	7.0408	0.0108
9	9.0068	0.0175

Table 3.1: Intel RealSense L515's test measurements

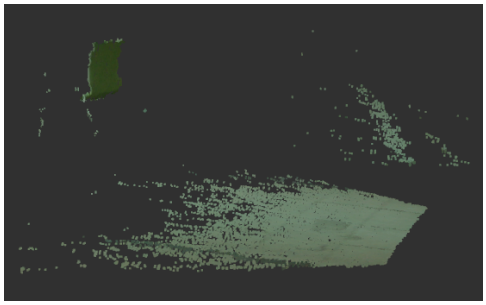
The results shown above (Table 3.1) are obtained by defining a circle around the center point of the depth image and obtaining the mean of the range values over a period of time.

The lidar's pointcloud is affected by noise, and in some situations it introduces severe problems into the system. In a very illuminated room with several windows, during the day, several points can be seen scattered. These points may be removed with outlier filters. However, during tests, it became evident that light sources pointed at the lidar, as well as metallic reflective surfaces led to a dense cluster of points near the lidar. This dense cluster can not be removed using outlier filters, as it can manifest itself up to 3 m from the lidar. Defining a minimum distance for points also does not work without risking the removal of important data. When considering a

2D scan, the previously mentioned scattered noise is minimal. However, this dense noise reassembles an obstacle and impacts both ICP algorithms and introduces false obstacles in the map.

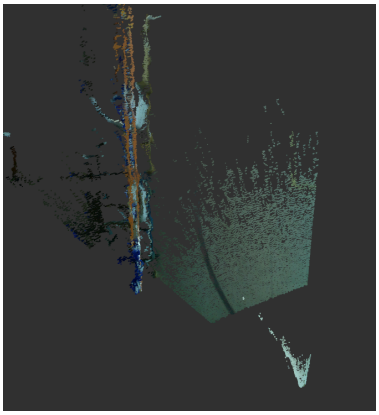


(a) Dense noise near camera on bottom left



(b) Noise in the right side of the image, chair on the left

(c) Noise cloud seen from the side



(d) Top view of noise near the L515 on bottom right

Figure 3.2: Various instances of noise represented in the point cloud

3.1.2 Velodyne VLP-16

The stated maximum range of the VLP-16 puck (Figure 3.3) is 100 m. Unlike the L515, this lidar has a 360° field of view and outputs point clouds consisting of sixteen rings at different angles (the vertical FOV is -15° to +15°). Update frequency is 10 Hz.

Figure 3.3: The Velodyne VLP-16.

3.1.3 Inertial Measurement Unit (IMU)

The IMU is a part of the L515. Both accelerometer and gyroscope outputs show drift over time. Given we are integrating both linear acceleration and angular velocity, drift increases exponentially. The gyroscope outputs angular velocity using the yaw, pitch and roll axis. Many 3D SLAM approaches use IMUs to recognize the gravity vector and provide the initial estimates for scan matching.

3.2 Software Tools

The ROS (Robot Operating System) framework is used to implement the navigation system using various open source packages. Several additional ROS nodes are written during the project using Python. During the work, added improvements regarding point cloud processing are made utilizing Matlab and the corresponding ROS Toolbox for direct connection to a ROS master from Matlab.

3.2.1 Robot Operative System

The ROS is a framework widely used in robotics, it facilitates communication between different programs, offers various tools for interaction, and a library with various state of the art open source programs already integrated. Since a significant part of

the implementation is done using ROS, basic knowledge regarding the framework is presented next.

In the ROS website one can find several open source packages. A package is a directory that may contain several programs (known as ROS nodes), as well as launch files, custom messages and services.

Launch files are used to ease the initiation of several nodes as well as setting its parameters.

Nodes communicate following a publisher/subscriber pattern by sending messages referenced by topics. Nodes send outputs structured as defined by a message type to a topic with a certain name. Any other node in the network can access the information by referring to the topic's name.

As an example, the driver running Intel Realsense L515 would output the lidar pointcloud (as a *sensor_msgs/PointCloud2* ROS message) in topic */camera/depth/-color/points*, and any program utilizing this information has to search for this topic to receive the data.

Finally, the ROS has a coordinate frame system that requires configuration according to our robot. Frames are displayed as a tree, where each junction represents a component of the robot and the edges represent the relative position and orientation between the junctions. This allows ROS to take into account the relative positions between sensors when using their data.

3.3 Scan matching

Before going further into the implementation of the system, it is worth to delve into scan matching, a process that has already been mentioned above. Lidar serves two main purposes: information about the environment through measurements that not only build the map and correct the pose, but also as a source of virtual odometry. In order to accomplish these goals, scan matching algorithms such as Iterative Closest Point (ICP) are used.

Scan matching algorithms take two point clouds or 2D laser scans and try to find

the rigid transformation that results in the minimum error between points or, in other words, that results in both being overlapped. The process is divided in two components: data association and error minimization. Data association refers to having for each point in a scan a corresponding point in the second scan. Ideally, these two points represent the same point in the object that the scan is representing. The error minimization takes these pairs of points and attempts to minimize the euclidean distance between them.

To take a look into the error minimization component let us assume two point clouds containing the same object model and same scale, but with an arbitrary difference in position and orientation. We now assume that we have the correct data association, meaning we know which points in the first point cloud correspond to those in the second.

The goal can be expressed by:

$$E(R,t) = \sum ||y_n - \bar{x}_n||^2 p_n \rightarrow min, \tag{3.1}$$

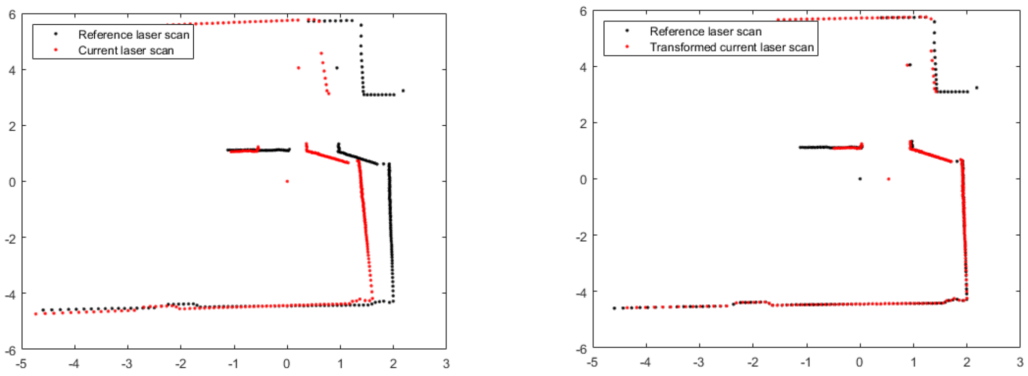
where $e_n = y_n - \bar{x}_n$ is the error vector regarding the pair of points with index n and $\bar{x}_n$ is a point in the first scan after it has been applied the rigid transformation: $Rx_n + t$; p_n is an optional weighting factor.

In this case there is an optimal solution that is obtained by a translation that overlaps both point clouds' centers of mass and a rotation that may be obtained using Singular Value Decomposition (SVD) for the least squares fitting. However, in a real world application there is no optimal solution. The corresponding pairs of points from each subsequent point cloud or scan can not be assumed to be known, because the scans are intrinsically noisy and different objects may enter and leave its field of view.

The ICP algorithm is used to align two 2D scans (Figure 3.4) or 3D point clouds by repeatedly associating pairs of points following a criteria (like associating the closest points in the different scans) and minimizing the difference between both scans utilizing non-linear least square methods. This process loops until error minimization is reached,

$$E(R,t) = \sum ||y_n - \bar{x}_{n,j}||^2 p_n \rightarrow min, \tag{3.2}$$

where j represents the previous iteration's guess.



(a) Scans taken at different times **(b)** Scans after scan matching

Figure 3.4: Visualization of scan matching[22].

Errors in finding the corresponding pairs of the two scans converge to a wrong result (i.e., a local minimum). This means that having a good initial estimate is critical to have the error minimization process converge correctly since the basic ICP algorithm finds pairs based on the closest neighbour.

Over the years several variants have been developed, approaching the algorithm with different strategies:

1. Selection of subsets of points to be used;
2. Criteria for matching pairs;
3. Weighting different values to selected pairs;
4. Rejection of certain pairs using a criteria;
5. Error metric.

Depending on the case, several variants may be better than others: if the environment has a lot of smooth surfaces, if there is a need to keep processing power low, etc. In this work the most important difference between the basic ICP and the one used was regarding item 2, where instead of using point-to-point association, point-to-line

association[5] was used (by employing the *laser_scan_matcher package*). Using a point to line metric leads to a faster convergence (less iterations).

3.4 Local Localization

A localization component starts each iteration in order to be fed into the SLAM node. It takes a 2D point-to-line ICP method that outputs 2D pose (using *laser_scan_matcher* ROS package) from the L515's and the Velodyne's VLP-16 data, IMU data and performs sensor fusion using an EKF (utilizing the *robot_localization* package). If wheel odometry is available it may easily be added. The results obtained by this step are expected to be accurate within a small time window. For long periods, drift accumulates noticeably, in other words, this pure localization step only allows for an accurate local localization estimation during short time intervals.

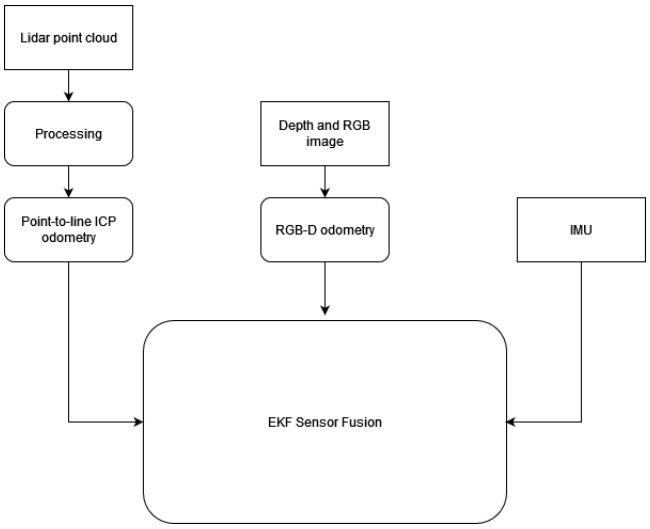


Figure 3.5: Localization component diagram.

3.4.1 Inputs

The Intel Realsense L515 outputs *sensor_msgs/Pointcloud2* ROS messages, which are converted into a 2D *sensor_msgs/Laserscan* message since the used ICP method seemed to work best with 2D scans. This is done by selecting a height interval in the depth image where the closest points are extracted (using *depthimage_to_laserscan* package). If at any point the robot rotates too rapidly, this might result in the two consecutive scans taken by the lidar being too different and applying ICP leads to

them converging to a local minima. As mentioned in Section 3.3, initial estimates are extremely important when applying ICP algorithms. To help prevent quick turns from ruining the scan matching, an Attitude and Heading Reference System (AHRS) filter is used to fuse IMU (accelerometer/gyroscope) data in order to have an initial heading estimate for the 2D ICP algorithm. Other odometry sources can be used for the same purpose, if available. The same scan matching and initial estimation process is done for the VLP-16 scan, except the conversion from pointcloud to scan, as the single horizontal scan is already published apart from the pointcloud.

The RGB-D odometry node is from the RTAB-Map package. The node gathers visual features (called GoodFeaturesToTrack), performs feature matching and computes the transformation using the Perspective-n-Point RANSAC approach. Pixels with invalid depth values are ignored. Both RGB image and an aligned depth image are fed to the node (as a *sensor_msgs/Image* message)

In parallel, an AHRS filter (implemented using the *imu_filter_madgwick* package) is used to obtain an estimate of the orientation; as time passes, this estimate will get less accurate as drift accumulates. While it can be used to obtain an estimate for position, given that drift is experienced by the accelerometer and we have to integrate twice, the error accumulates exponentially and diverges quickly.

An EKF is used to fuse the sources of odometry, it aims to output a more accurate 2D pose than it would be by utilizing any individual source.

3.4.2 EKF implementation

The state vector used by *robot_localization* contains 3D position, orientation as well as 3D linear, angular velocities and acceleration. For the purpose of this work, we take into account 2D movement as it is enough for most indoor situations.

The ICP and RGB-D odometry are inputted as */nav_msgs/Odometry* messages and the IMU as *sensor_msgs/Imu*. Since all odometry (except IMU) are expressed as a pose with the initial location as reference, over time these two poses might diverge to a point where their variances do not overlap, leading to the resulting estimated pose jumping between poses. To avoid this, one of the sources is selected to be derived to

use velocity instead, this is done by setting the "differential" parameter within the *robot_localization*'s configuration file.

3.5 SLAM

Several SLAM packages have been tested and compared in this work: HectorSLAM, Gmapping, Cartographer, RTAB-Map, LOAM_Velodyne and SLAM Toolbox. The first three are being run on a laptop with the ROS Noetic version, while the last three were run on a Jetson Xavier NX with ROS Melodic.

3.5.1 HectorSLAM

Hector-SLAM[13] is a 2D SLAM algorithm that relies on robust scan matching (scan to map) and has the option to use the IMU for the initial estimate. It is fully based on scan matching and it does not do any kind of optimization with detected loop closures.

It does not require any sort of continuous localization to function. However, this leads to quick turns being fatal to the localization, since when scan matching fails there is no source of odometry to rely on.

The scan to map scan matching process is very robust as it keeps several maps with different resolutions, where scan matching starts on the coarsest resolution and is used for the initial estimate for the scan matching on the next finer resolution map. Since it does not perform loop closures, errors accumulate and might consistently have bad results in vast maps.

3.5.2 Gmapping

Gmapping[10] is a FastSLAM 2.0 approach for grid maps. Each sample comprises its pose and a map estimate, which is created taking into account the particle's trajectory and observation.

The proposal distribution considers both odometry information as well as the current scan, which increases its accuracy since range finder information tends to be more

precise than solely using odometry. It further seeks to reduce particle depletion by doing selective resampling. Resampling only occurs when particles have significantly different weights,

$$\frac{1}{\sum_i (w_t^{[i]})^2} < N/2, \tag{3.3}$$

where i refers to each particle, N is the total number of particles and w_t is the respective weight.

3.5.3 RTAB-MAP

Real-Time Appearance-Based Mapping[14] (RTAB-Map) is a graph-based SLAM approach that uses RGB features to detect loop closures. Visual features are extracted from RGB images and used to classify each image with a bag of words approach. By linking images in a virtual edge connecting their respective poses, the associations are then used by error minimization algorithms.

The memory management ensures the system to function within real-time constraints by carefully selecting weights to each nodes, discarding consecutive nodes of the same place and only keeping the most distinct ones (higher weight) in the designated "working memory". The nodes kept inside the "working memory" are used to detect loop closures.

If both RGB and pointclouds (or laser scans) are available, the pointclouds can be used to further refine the transformation between the detection.

Three graph optimization processes are selectable: g2o, GTSAM and TORO.

The package includes two virtual odometry nodes, one for RGB-D and another is ICP-based (for 2D and 3D). RTAB-Map has Octomap integrated, which means that the octomap is updated according to graph optimizations, and allows map serialization.

3.5.4 LOAM Velodyne

Lidar odometry and mapping in real-time[30] (LOAM) is designed to be used with lidar. It is divided in two algorithms, one performs low accuracy odometry at a high frequency to estimate velocity while the second provides a finer scan matching and performs registration of the pointcloud into the map, at a lower frequency.

None of the algorithms perform scan match with the entire scan and instead feature points are extracted using a metric that defines smoothness; both points within locations of maximum smoothness (planes) and minimum smoothness (edges) are extracted. With the second algorithm, the odometry given by the first is used for the initial estimation of the pose for the current scan on the map. The scan to map scan matching in this step utilizes ten times more the feature points used in the odometry step. It accepts IMU to further help movement estimation, but it does not perform loop closure.

3.5.5 Cartographer

Cartographer[11] is a highly customizable package that was used by Google, however, it is currently unmaintained. It is divided into a local SLAM and the global SLAM. The former creates submaps by scan matching paired with odometry sources and, if the sources are not trustworthy, a more expensive matching method can be selected. Local submaps are expected to drift and because of this, after a selectable amount of range data, the submap closes. For each submap, the drift is supposed to be unnoticeable (under the map's resolution). Global SLAM is a typical graph-SLAM, with direct constraints and virtual constraints. The second scan matching method mentioned in local SLAM, which is robust and computationally expensive, is the same used to detect loop closures. Google's Ceres is utilized in both graph optimization and scan matching. It allows for 2D and 3D SLAM, one uses 2D laser scans as input, and the other 3D point clouds.

3.5.6 SLAM Toolbox

SLAM Toolbox[21] is the most recent package in this list and could be considered the next generation of Karto SLAM, which is one of the first graph-based approaches in ROS. It is built on an improved version of the SRI International's Open Karto library.

Significant differences from the original Open Karto are: utilizing Ceres instead of sparse bundle adjustment to perform graph optimization for increased speed, improving scan matching by allowing multi threading and supporting map serialization, to perform SLAM while using maps from previous sessions.

The last point, map (and pose graph) serialization, is part of a feature named "LifeLong mapping", which aims to map an environment over time and being able to partially update the pose graph, including deleting nodes with dated information and consistently refine the map.

3.6 Mapping

3.6.1 Octomap

Octomap[12] does 3D occupancy grid mapping using octrees. The capability to update previously seen places thanks to probabilistic gridmaps makes it extremely valuable in dynamic environments. Poses are taken as inputs as well as the respective point clouds, which are inserted into the map. The pose may be given by a SLAM algorithm such as those previously mentioned.

Many of these methods build only 2D maps and so, using Octomap could be valuable if 3D maps are desired, but 3D SLAM is too computationally expensive. While a SLAM algorithm is running, Octomap can be utilized to do mapping in parallel, using the SLAM's pose estimates. However, in the case of using particle filter based SLAMs such as Gmapping, Octomap would be required to map for each single particle. Also, SLAM methods that execute graph optimization change the prior poses and therefore also the map, requiring Octomap to be updated accordingly. RTAB-Map

already has the option to use an integrated Octomap which updates accordingly. As for HectorSLAM, since it does not perform any type of change to the prior poses, Octomap could be used in parallel.

3.7 Pathfinding

Pathfinding has a simple objective: it takes a destination point or a goal, a map and the robot's localization within to trace what would be, ideally, the shortest path the robot needs to take to reach the goal.

The **Dijkstra's algorithm** and its variants are commonly used in robotics to perform pathfinding. It aims to return the shortest path between nodes connected by edges. The nodes represent a state, while edges represent the difficulty of traversing between the two states:

1. Initializing a list that contains the distances of shortest path from the starting node A to every other node in the network, giving every node the infinity value except the starting node;
2. Starting on node A, the distance to the adjacent nodes is checked and the list updated;
3. The next node with the smallest distance between itself and the starting node A is selected and the previous step is repeated;
4. The algorithm shall stop when every node is updated on the list or when the remaining unchecked nodes are not reachable from the starting node.

For path planning, the package *move_base* is selected and it employs the Dijkstra's algorithm. A destination is selected either by using the Rviz (the visualization tool in ROS) plugin, which allows us to select with the mouse where to move in the map, or by manually sending a *geometry_msgs/PoseStamped* message through the *move_base_simple/goal* topic. As a result, *move_base* outputs angular and linear velocity matrixes, which may be used by a movement system, as well as a useful Rviz visualization of the planned path.

Two costmaps are created, one global and one local. A costmap is a concept widely used in navigation; it is a map that contains the information of the difficulty to travel through each cell. In this case, any location up to a defined distance (e.g., 50 cm) surrounding objects is selected to have maximum difficulty. This way, the planner does not approach any object under the selected distance (these cells are given maximum difficulty). The global costmap uses the map as it is created by the SLAM, while the local costmap uses the realtime laser scans from the sensors. This way, the global plan is a general direction taking into account the various previously seen structures in the map. The local plan is set to update more frequently and takes into account obstacles seen in realtime; its purpose is to guide the robot towards the global planned path.

During development, an algorithm that detects traffic lights, and which light is on, was available; it publishes the color to a ROS topic. A short node was written to pause the publishing of the control commands whenever the red light was on. The commands would resume when the green light was back on.

The first prototype to be built by Active Space Technologies is expected to be a four mecanum wheeled robot. Therefore, the velocity output was converted to the individual movement of each wheel by applying inverse kinematics. Mecanum wheels (Figure 3.6) are a type of omnidirectional wheels that have rollers placed on the circumference of the wheel at 45° to the wheel plane. The node subscribes to the *cmd_vel* topic outputted by the *move_base* package and publishes the angular velocities, ω, for each wheel using the following equations[27]:

$$\begin{aligned}
\omega_{FL} &= \frac{1}{r}(v_x - v_y - (l_x + l_y)\omega_z) \\
\omega_{FR} &= \frac{1}{r}(v_x + v_y + (l_x + l_y)\omega_z) \\
\omega_{RL} &= \frac{1}{r}(v_x + v_y - (l_x + l_y)\omega_z) \\
\omega_{RF} &= \frac{1}{r}(v_x - v_y + (l_x + l_y)\omega_z)
\end{aligned} \tag{3.4}$$

The first letter in the subscript refers to front or rear wheels and the second to left

and right wheels, r expresses the radius of the wheel, l_x is half the distance between the front (or rear) wheels and l_y is half the distance between left (or right) wheels.

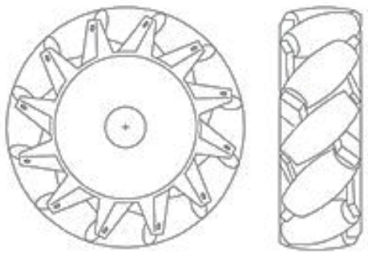


Figure 3.6: Schematic of a mecanum wheel.

3.8 Movement and people detection

Doing SLAM in dynamic environments is an arduous task as the moving features may be significant enough to trick localization sources that depend on range finders.

If, for example, ICP-based localization is being used, moving features can deceive the algorithm by having it minimize the error of moving features instead of static ones. The more important the feature is the more likely it is for the mistake to occur. A lidar such as the Velodyne's VLP-16 might not easily experience this effect thanks to its range and 360° angle, which is capable of gathering much more information on the environment. Scan matching using the L515's scans are more prone to be affected by people in movement. Since it only has a 9 m maximum range and a 70° horizontal field of view, individuals moving within the scan make a relevant portion of what is represented by the scan. The downside of removing any person from the L515's scans, used for odometry, is when situations arise where a static person may be the only major feature in the scan, and removing it results in scan matching being infeasible. Still, the data redundancy from the IMU and VLP-16 scans keep the localization functioning.

Removing people from the scan or point cloud, used for mapping, is not as critical since many SLAM algorithms use probabilistic grids, and a moving person is not immediately inserted in the map.

To improve the system on this front and, at the same time, to add a foundation for

future interaction with humans (such as turning the UV-C light off), RGB object detection techniques have been explored, implemented and paired with two developed methods (one removes clusters and the other filters the depth image). The detection is accepted as an argument by these methods, which remove humans from point clouds and scans, with the purpose of eliminating the errors induced by people moving around the robot.

3.8.1 Object detection

To detect objects using RGB images, a deep learning model for semantic image segmentation called Deeplab[6] has been used with the PASCAL VOC 2012 dataset. The dataset does not allow for an extremely diverse object detection, but since there is only interest in detecting humans, it suffices.

Given that ROS has its own image format, the package *cv_bridge* is utilized in order to convert */sensor_msgs/Image* to OpenCV images. An example of segmentation of a RGB image is seen in Figure 3.7.

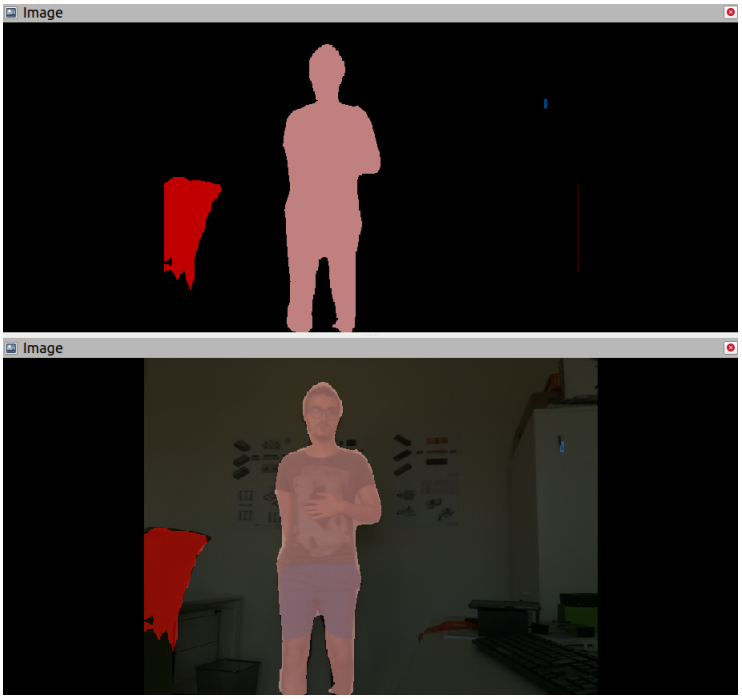


Figure 3.7: Segmentation of a person and a chair.

Other methods[20] had been explored: a full-body HOG (histogram of oriented gradients) detector and a 2D laser scan leg detector. The first had trouble detecting people

that are partially covered and, in other cases, it would also present several false positives. The leg detector would accurately detect people but only if moving at a certain speed. When walking too fast or taking long steps the detector would fail to recognize the person.

3.8.2 Using point clouds

In this subsection, approaches that identify people or moving objects in a point cloud are demonstrated.

The following components have been implemented using Matlab and take advantage of the ROS Toolbox, which allows communication between Matlab and ROS. Matlab might not be the most suitable tool to perform these operations, the most commonly used tool for point cloud processing is the Point Cloud Library (PCL). Since we are only validating the process and thus not concerned with performance, Matlab was still chosen for its simple and quick implementation of basic operations such as cluster segmentation.

Human segmentation

For each detection done by the RGB object detection algorithm, the position of the pixel at the center of the detection is sent to the */people_centers* topic. The Matlab code subscribes to this topic as well as the */camera/depth/color/points* topic, which contains the point cloud.

First, the ground is segmented from the rest of the point cloud. This is done by calculating the elevation angle between a point and its neighbours[2]. If the elevation angle is below a certain threshold, the adjacent point is considered to be in the same ground plane. The method assumes a floor with minimal curvature. A simpler alternative is to remove every point in the point cloud that is under a certain height. This requires the lidar to only experience rotation around the yaw axis. Finally, the remaining points are divided into clusters, which are defined considering a maximum distance between points of the same cluster.

The center pixel of the box bounding the person is translated to polar coordinates by

performing the conversion considering the field of view of the camera and resolution. A line is traced aiming at the human cluster in the point cloud. To remove the person, one now needs to know the distance, r, to the human cluster.

To get the cluster representing the person, every cluster intercepted by the line defined by the polar coordinate has their dimensions evaluated. Considering its height and width we can verify if it is the detected human, for the most part.

After having obtained the cluster, the points within it are removed from the point cloud. This happens to every human detected by the object detection algorithm.

Algorithm 3: People Filter

Result: Filtered Pointcloud

Initiate ROS communication;

while *True* **do**

  PointCloud = Receive(L515 Point Cloud);

  CentersList = Receive(Centers List) % *List of pixel coordinates of the people*;

  if *People are detected* **then**

    while $i < n^o$ *people* **do**

      $[\theta, \phi] = PixelToPolarCoord(CentersList(i))$

      while r<*MaxRange* **do**

        px = r*sin(θ), $py = r * cos(\phi)$

        Cluster = CheckForClosestCluster([px,py],PointCloud);

        if $r <$ *MaxDist* ***and*** *MinSize*<*Cluster*<*MaxSize* **then**

          HumanToRemove =+ ObtainClusterIndexes()

        end

        r = r + step

      end

      i=i+1

    end

    FilteredPointCloud = RemovePoints(PointCloud,HumanToRemove);

    Send(FilteredPointCloud);

  end

end

Movement detection

A method that attempts to detect movement, regardless of being a person or not, was also written. It takes as input the point cloud after the ground removal and cluster segmentation, and assumes the lidar to be static. For each cluster in the point cloud at time t, it searches for the closest cluster of the point cloud at time $t-1$ that has similar dimensions. If the distance between these two clusters is over a certain threshold, the object or person that the cluster is representing is considered to be moving.

The distance value computed is between the centers of the cuboid that fits the cluster. The minimum distance necessary to assume movement was set taking into consideration the average walking speed (around 1.4 m/s).

Since the L515 updates at a rate of 30 Hz, performing the computation between every frame yielded bad results, because the cluster's centers slightly fluctuate thanks to noise. Calculating the distance between the closest similar cluster every fifth frame means that the center of the cluster needs to move around 0.25 m and, this way, the algorithm manages to perform with less false positives. To further reduce false positives, the object is required to be in motion during three or more iterations before being considered to be moving.

Algorithm 4: Movement Detection

Result: Filtered point cloud

```
Initiate ROS communication;
while True do
    Receive(L515's PointCloud);
    PointCloud = RemoveGroundPlane(L515's PointCloud);
    ListofClusterInfo = ClusterSegmentation(PointCloud) %contains dimensions
     and position;
    while i<size(ListofClusterInfo) do
        SameCluster = SearchForClosestSimil-
         arCluster(ListofClusterInfo(i),PreviousListofClusterInfo) % searches for
         close clusters in the previous point cloud with similar height, width,
         volume;
        if SameCluster != NaN then
            FilteredPointCloud =
             RemoveCluster(PointCloud,ListofClusterInfo(i)) %if removing
             cluster in movement
        end
        PreviousListofClusterInfo = ListofClusterInfo;
    end
    Send(FilteredPointCloud)
end
```

3.8.3 Using depth image

Using Deeplab's semantic image segmentation, a mask is placed on top of the detected features, human or otherwise. The model outputs a matrix with the size of the camera's resolution, where each position refers to a pixel. The positions where the pixel is not containing a detected object have the value "0" where the others might have one of several values that refer to different objects (in the case of the algorithm being used, to know where a human is in the RGB image we search for pixels with number "15").

In order to remove any unwanted objects from the depth image, the corresponding

numbers are searched for in the matrix and their coordinates (row and column) are saved. These pixels are given a NaN value in a depth image that is aligned with the RGB camera, effectively performing segmentation in the 3D space with this extremely simple process.

Afterwards, the depth image is converted into a point cloud (using *depth_image_proc* package) or laser scan (using *depth_image_to_lasercan* package) and the resulting point cloud or laser scan can be used by the developed navigation system. This ensures the moving individuals do not affect the systems localization and mapping.

3.8.4 Noise removal through cross-referencing sensors

As mentioned in Section 3.1.1, the Intel RealSense L515 lidar shows significant masses of noise as the robot points at a few specific places in the test setting. This noise is dense and with a size that no typical filter can trim out. A solution has been found, which consists in utilizing another range sensor type, a depth camera, and use it to discriminate objects from noise.

Two methods have been tried, both starting with:

1. Place both point clouds within the same referential frame;
2. Perform cluster segmentation in both point clouds.

Method 1 (Algorithm 5) takes the geometric center of each cluster in the lidar's point cloud and checks the distance to the geometric center of the closest cluster in the depth camera's point cloud. If this distance is over a certain threshold, it is assumed that the lidar's cluster does not represent a real object, and instead is noise.

Method 2 takes the depth camera's point cloud and within the XY plane, the area where objects are detected is defined taking into account the observed clusters. Any point in the lidar's point cloud that is outside this area is removed.

Algorithm 5: Noise Filter: Method 1

Result: Filtered point cloud

Initiate ROS communication;

while *True* **do**

  Receive(L515's PointCloud);

  Receive(D435i's PointCloud);

  ListofClusterL515 = ClusterSegmentation(L515's PointCloud);

  ListofClusterD435i = ClusterSegmentation(D435i's PointCloud);

  while *i<size(ListofClusterL515)* **do**

    Dist =

     DistanceToClosestCluster(ListofClusterL515(i),ListofClusterD435);

    if *Dist > ThresholdDistance* **then**

      FilteredL515's PointCloud = RemoveCluster(ListofClusterL515(i))

    end

  end

  Send(FilteredL515's PointCloud)

end

A high-level system diagram, containing all the components mentioned above, can be visualized in Figure 3.8.

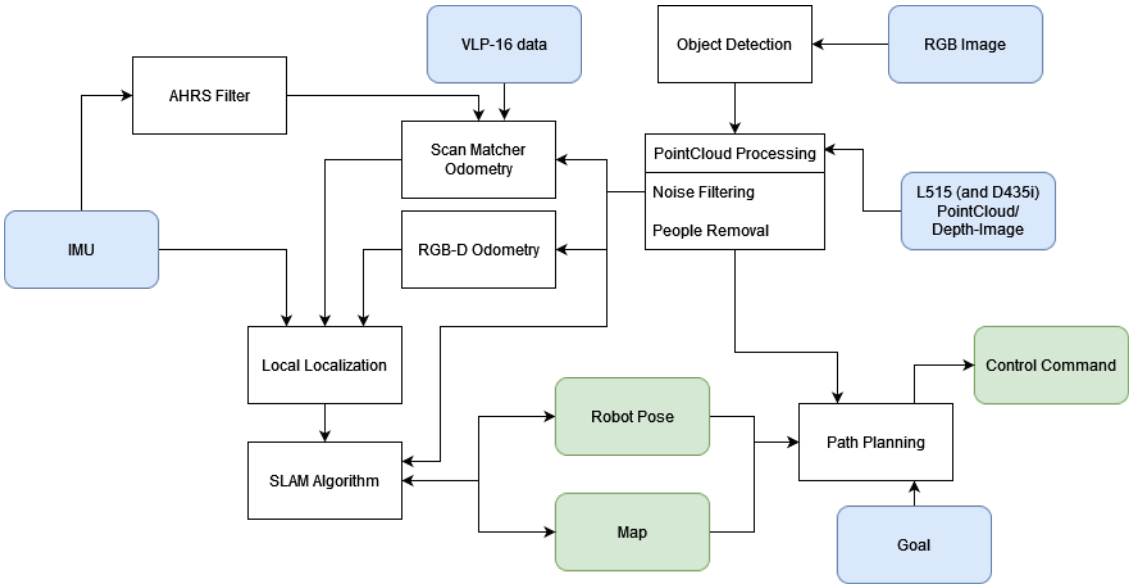


Figure 3.8: The system diagram.

3.9 Experimental apparatus

The objective of the tests are to evaluate the several SLAM packages with the Velodyne's VLP-16 and the Intel RealSense L515 lidar, separately. As the location of the tests is ample and considering the lack of encoders, when mapping with the L515, the VLP-16 scan may be used for odometry.

In this section, the experiments done to validate the SLAM approaches are explained, as well as the methods used to compare both maps and trajectories to the ground truth.

3.9.1 Platform

In order to test the SLAM approaches, both VLP-16 and the L515 were placed on top of an AGV (Figure 3.9).

Figure 3.9: AGV with a VLP-16, L515 and a D435i.

The L515 was pointed forwards and around 0.7 m in front of *base_link* (a virtual frame set in the ROS), while the VLP-16 was 0.4 m above the *base_link*.

The AGV followed a predefined path, which is not in any way influenced by our current system. The course consists in a loop around two pillars in a wide room, followed by a passage through three rooms and ends with a lemniscate around the previously mentioned pillars. The path was chosen in order to give the opportunity for the approaches to perform loop closures after the first loop, after passing through the smaller rooms and during the lemniscate.

Considering the limitations of the range and field of view of the L515, several objects were carefully placed around the rooms with the goal of making sure there was at least one feature at any point in time observed by the lidar. Because of the small height of said objects, some of these do not appear in the horizontal plane scan taken by the VLP-16.

During the course, the data given by both the VLP-16 and the L515 was recorded to a *rosbag* file (.bag). This file contains the point clouds, scans, IMU data, RGB video as well as the transformations between frames paired with timestamps, which can be replayed at a later time.

Two courses were done, the first was done in a static environment, meaning that no humans moved within the scan of the L515 nor did the configuration of objects change at any point in time. In the second course, several humans were walking within the range of the scan and the object layout was changed between the loop and the lemniscate.

While playing the same *rosbag* files, the following SLAM packages have been used to perform mapping and localization: hector_slam, gmapping, cartographer, slam_toolbox, loam_velodyne and rtabmap_ros.

3.9.2 Map and trajectory evaluation metrics

Both resulting maps as well as trajectories are compared with ground truth data. The ground truth trajectory is given by the localization system currently used by Active Space Technologies.

To evaluate the trajectory, both Absolute Trajectory Error (ATE) and Relative Pose

Error (RPE) are calculated. First, both the ground truth trajectory and the obtained trajectories must be aligned, which is achieved by applying ICP-based scan matching.

May P' be the list containing all 3DOF poses that make up the ground truth trajectory given by the localization algorithm, and P the list of poses given by the SLAM algorithm to be evaluated. These might have different lengths since the frequency of which the trajectory updates is different for different algorithms

$$P' = \{p'_1, p'_2, p'_3, ..., p'_{N_{P'}}\},\ P = \{p_1, p_2, p_3, ..., p_{N_P}\}, \text{where } p_i = (x_i, y_i, \theta_i), \tag{3.5}$$

Where P' corresponds to the list containing ground truth poses, and P refers to the current algorithm that is under test.

The ATE is obtained from

$$ATE(t) = |P'_{t'} - P_t|, \text{where } t' = \frac{t}{N_P} * N_{P'}. \tag{3.6}$$

As for RPE, it is expressed by

$$RPE(t) = |(P'_{t+\Delta'} - P'_t) - (P_{t+1} - P_t)|, \text{where } \Delta' = \frac{1}{N_P} * N_{P'}. \tag{3.7}$$

This metric demonstrates the drift of the trajectory, meaning it does not consider the accumulated error up until point t.

Separately, the resulting maps are also compared to a ground truth map. Like trajectories, maps may have to be aligned; Matlab Image Processing Toolbox is used to perform the alignment. The map is saved as an image file (.pgm or .jpg) and SURF features are extracted (Figure 3.10), followed by applying MSAC (a RANSAC variant) to filter out outliers and obtaining the transformation between features. In some cases, the maps are too distinct and this method does not produce the desired results. In those cases, ICP-based scan matching has been used after manually giving an initial estimate. To evaluate how good the obtained map is, the coordinates of the occupied cells in the ground truth map and the map obtained from the algorithm

are selected. For each selected occupied cell in the map under test, the distance to the closest cell in the ground truth map is computed.

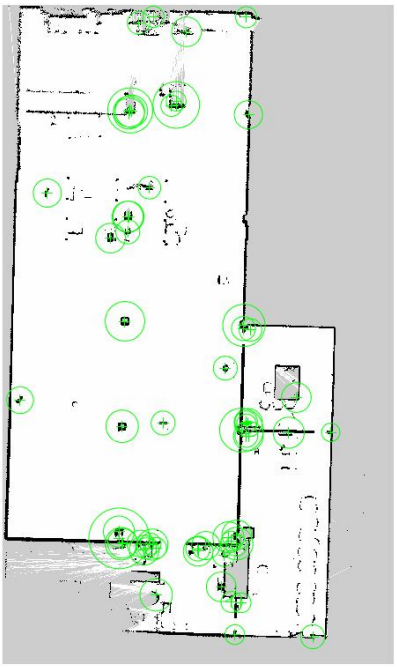


Figure 3.10: SURF features detected in a map.

The ground truth trajectory is obtained from the resulting map of a SLAM algorithm paired with the VLP-16, which is deemed as accurate enough as seen in Chapter 4.

Chapter 4

Results

For each package, the generated maps are presented, as well as the trajectory, followed by an overview, where each package is compared.

While processing the *rosbag*, an unpredicted error manifested itself, which consisted on a desynchronisation between the Intel RealSense L515's scan and the VLP-16's scan when publishing an */odom* frame. The */odom* frame is generated by the local localization segment and is used by the SLAM package. The temporal offset leads to the localization given by the ICP using the VLP-16 scan to be delayed in relation to the mapping by the L515 scan; the outcome is extremely noticeable during curves as the scans move before the pose of the robot does. This problem is overcome by running the local localization once and saving the data in a text file. The text file contains a table with the position and orientation resulted from the ICP odometry, which is published to ROS after removing the first few rows to counter the delay. Also, because of it, every experiment that used the odometry from the VLP-16 uses the same values (these would vary each time the algorithm is run).

Every map presented has a resolution of 0.05 m. Smaller resolutions have been experimented but values below 0.05 m have not showed consistently good results by every SLAM package.

The chosen ground truth map is from using the VLP-16 alongside the SLAM Toolbox package (Figure 4.1).

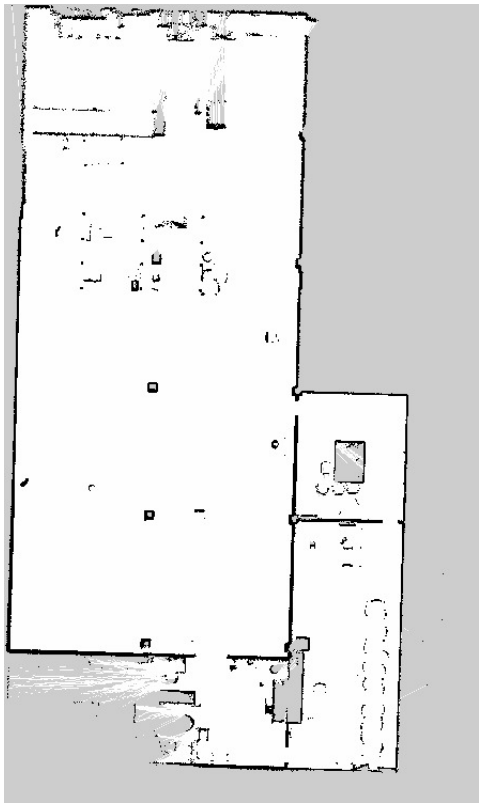


Figure 4.1: The ground truth map (roughly $650m^2$) from SLAM Toolbox.

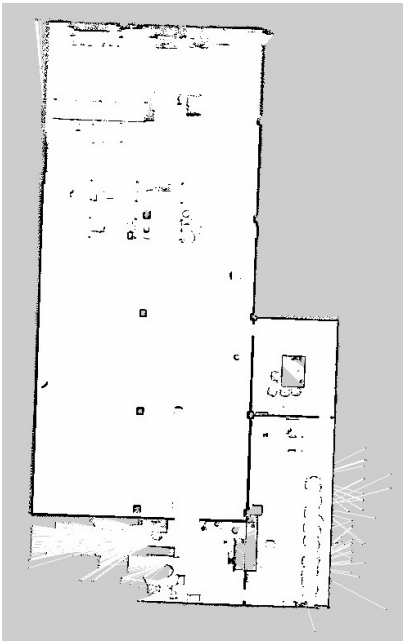
For future reference, the biggest room is the basement, the smallest on its right is the conference room, the one below is the canteen and, finally, the one on the bottom left is the laboratory. While two routes have been performed, one with a static environment and other with changes in the environment, barely any difference was seen between them. Future tests might require more changes and moving obstacles.

4.1 HectorSLAM

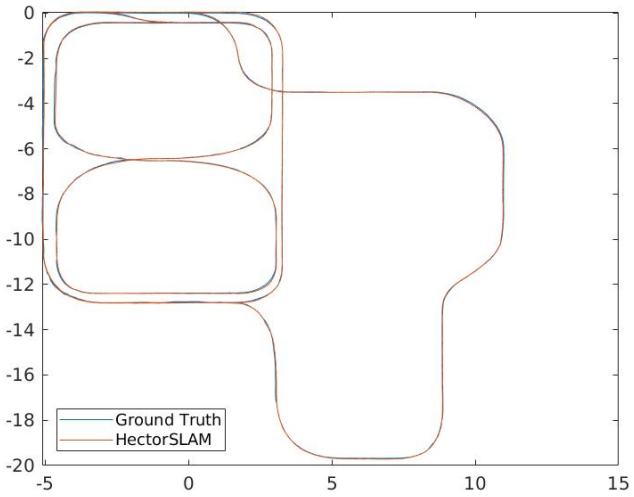
In this section the map and trajectory obtained from HectorSLAM are presented (Figures 4.2 and 4.3). The trajectory is obtained from the transformation between the map referential and the referential frame *base_link*.

Using Velodyne VLP-16

The default parameters were used.



(a) The map



(b) The trajectory

Figure 4.2: Results given by HectorSLAM and VLP-16.

Using Intel RealSense L515

The small range and field of view of the Intel RealSense L515 were causing the robot to lose itself and jump around the map after a point. Because the default parameters have proven to be insufficient, certain parameters were changed according to the Table 4.1.

Table 4.1: HectorSLAM parameter table.

Parameter	Default Value	Custom Value
map_update_distance_thresh	0.4	0.2
map_update_angle_thresh	0.9	0.3

By changing these parameters the map is expected to update more frequently and reduce the likeliness of losing itself in the map. While the basement was correctly mapped, localization jumped while passing through the conference room.

(a) The map with default parameters **(b)** The map after changing parameters

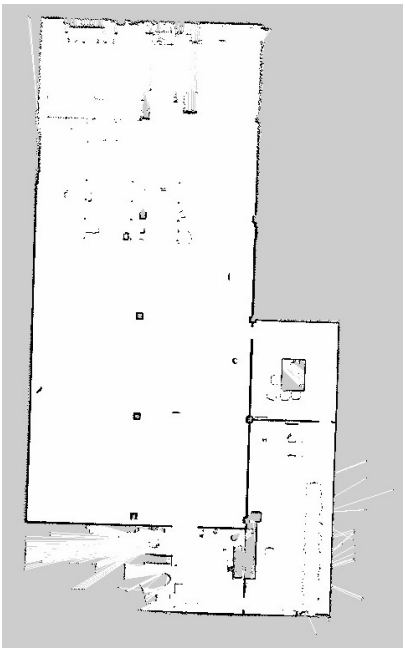
Figure 4.3: Results given by HectorSLAM and L515.

Neither settings were able to pass through the laboratory without HectorSLAM losing itself, so the maps were paused at that point. The terminal printed the warning "SearchDir angle change too large", which refers to a failure of the scan matcher at finding a valid solution.

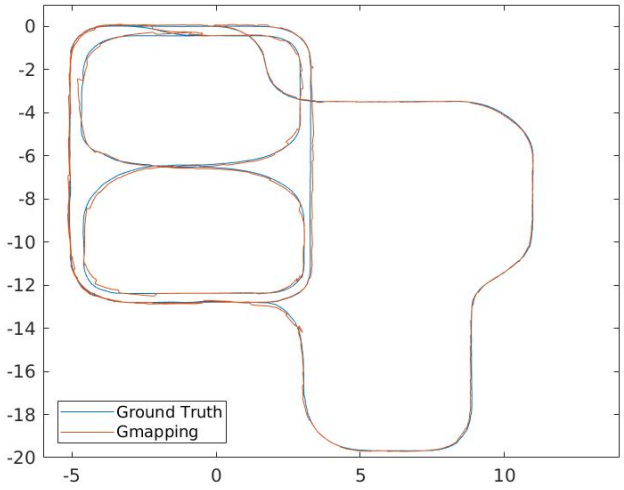
4.2 Gmapping

In both cases Gmapping was run with the default parameters except for the number of particles, which is set to 60 instead of the default 30 (Figures 4.4 and 4.5).

Using Velodyne VLP-16



(a) The map



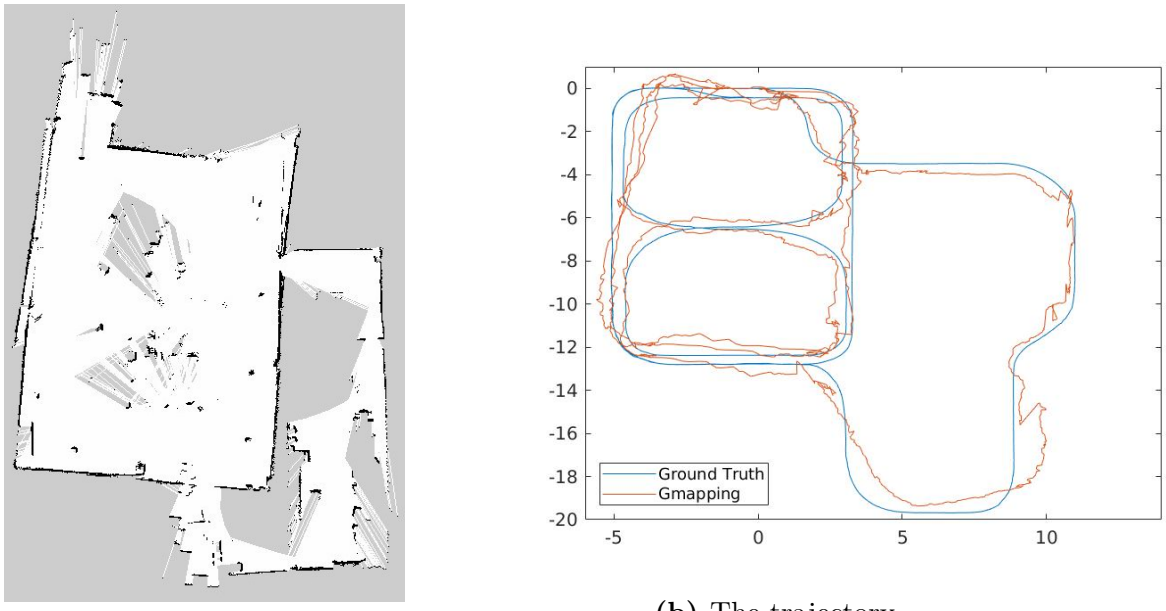
(b) The trajectory

Figure 4.4: Results given by Gmapping and VLP-16.

By observing the trajectory, which is obtained from the transform between the frame *map* and *base_link*, we can see that during the course the pose barely drifted.

Using Intel RealSense L515

The odometry used came from the VLP-16 and the IMU. The ICP odometry using the L515 would repeatedly fail. In this case, the angular drift is noticeable in the map, specially in the basement. By observing the trajectory one can see that the map was being built correctly, but as the robot was passing through the canteen the winning particle changed, mistakenly; this resulted in the canteen being pushed to the right.



(a) The map

(b) The trajectory

Figure 4.5: Results given by Gmapping and L515.

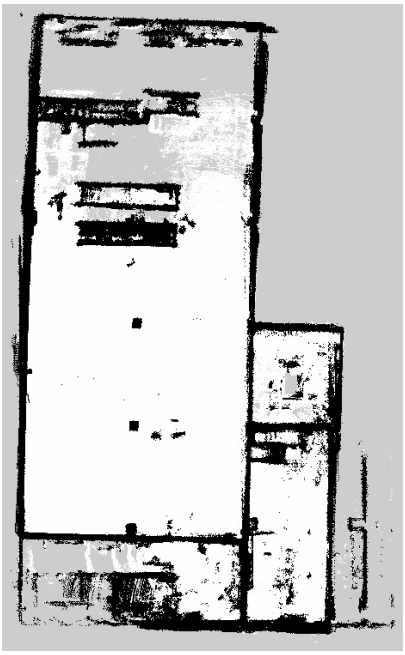
4.3 RTAB-Map

RTAB-Map was run on a Jetson Xavier NX, and the libraries that RTAB-Map makes use of are not built to be support CUDA (a parallel computing platform). This means the processes are being run by the CPU, resulting in the crash of the SLAM. This happens because the algorithm can not keep up with the data at the speed that it is being played (the original speed that it was taken). Given how slow the processing is, the rate at which the *rosbag* data is being played is cut by half (0.5). The trajectory is given by RTAB-Map after applying graph optimizations (Figures 4.6-4.8).

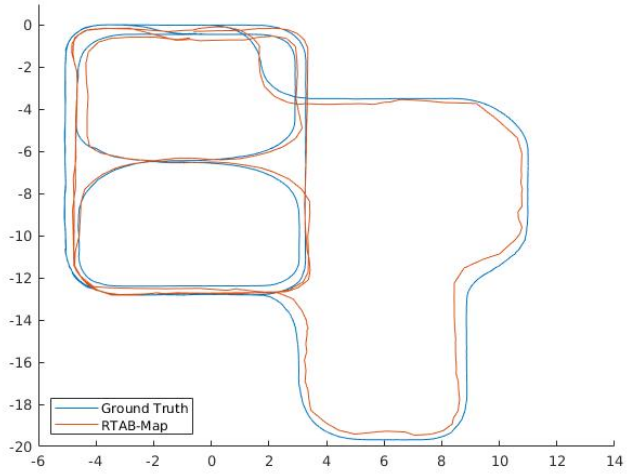
Both SLAM and odometry nodes in this package are limited to 2D movement, which is set by the "Reg/Force3DoF" parameter.

Using Velodyne VLP-16

The launch file (containing the parameters) utilized is tailored for the Velodyne VLP-16 by the author of RTAB-Map[23]. Unlike the algorithms above, the odometry (done by the *icp_odometry* node in RTAB-Map package) and mapping was done using the 3D point cloud instead of a 2D scan.



(a) The map



(b) The trajectory

Figure 4.6: Results given by RTAB-Map and VLP-16.

The map resulting from the second rosbag shows a moving person in the canteen.

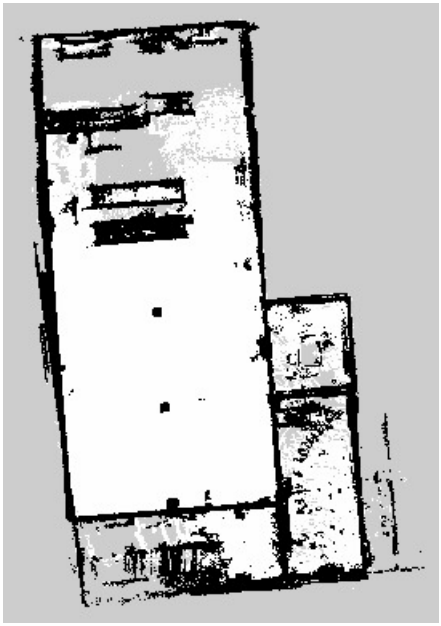
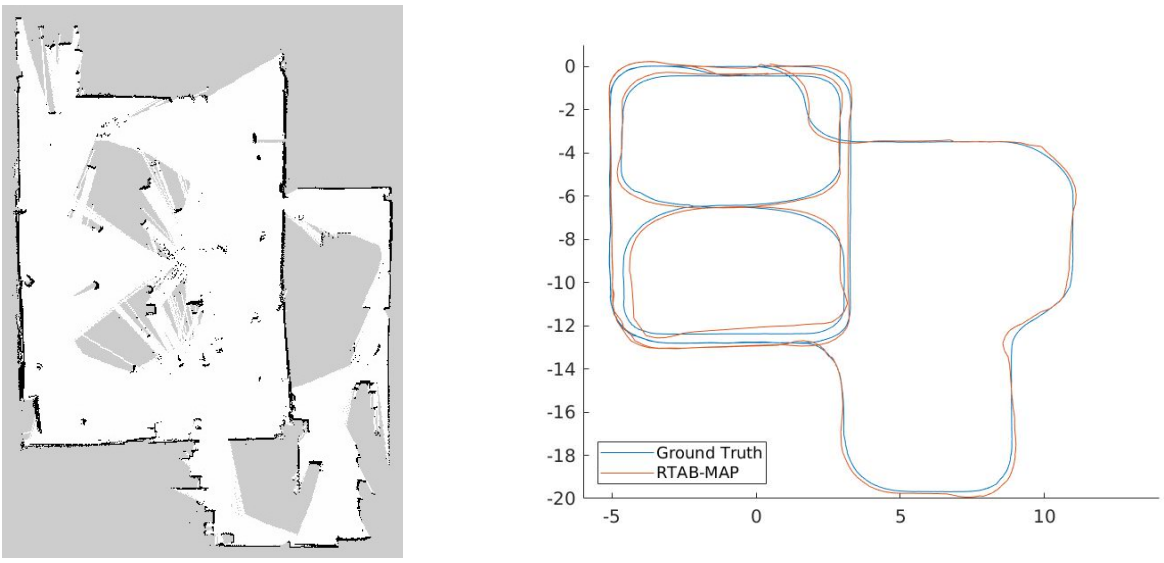


Figure 4.7: Map resulting for RTAB-Map and VLP-16.

Using Intel RealSense L515

In this experiment, instead of using ICP odometry from the VLP-16, odometry was

given by the RGB-D odometry node (and the IMU) supplied in the RTAB-Map package to test its performance.



(a) The map **(b)** The trajectory

Figure 4.8: Results given by RTAB-Map and L515.

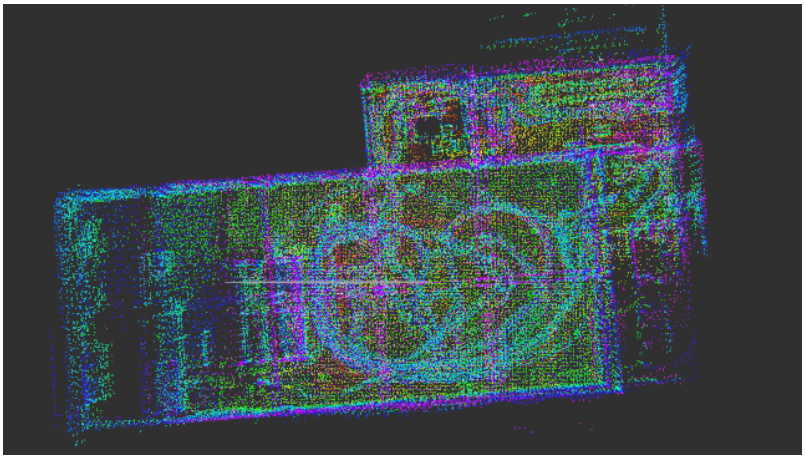
While the RGB-D odometry performed nicely during this test, most times it was too unstable and unable to recover. The difference between grid maps acquired from 2D SLAM and grid maps resulting from projecting a 3D map is also visible; the walls in the former seem sharper.

4.4 LOAM Velodyne

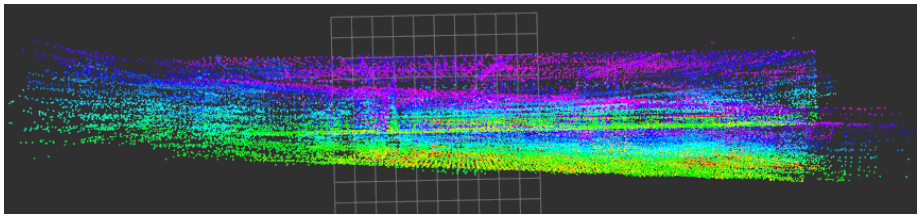
LOAM Velodyne is designed to work with Velodyne's models only (Figure 4.9), and therefore the L515 could not be tested.

Using Velodyne VLP-16

The default parameters were used.



(a) The map from above



(b) The map from the side

Figure 4.9: Results given by LOAM and the VLP-16.

While the map maintained its form when seen from above, every time the robot passed through the canteen its pose began to sink into the ground. Moving humans are also visible in the map seen from above, as one can even see their trajectories in the point cloud.

4.5 Cartographer

Cartographer takes as input both range finder information and IMU. The trajectory is given by Cartographer after taking into account any graph optimizations (Figures 4.10 and 4.11).

Using Velodyne VLP-16

Cartographer's localization consistently began jumping poses at the laboratory when mapping using 2D scans. Therefore, 3D point clouds were used. The odometry transformation is provided by Cartographer by setting the "provide_odom_frame" parameter in the .lua configuration file to true. Unfortunately, the trajectory of the robot after taking into account loop closures was not obtainable, as Cartographer

saves the information into a .pbstream file that has not been able to be read in this case.

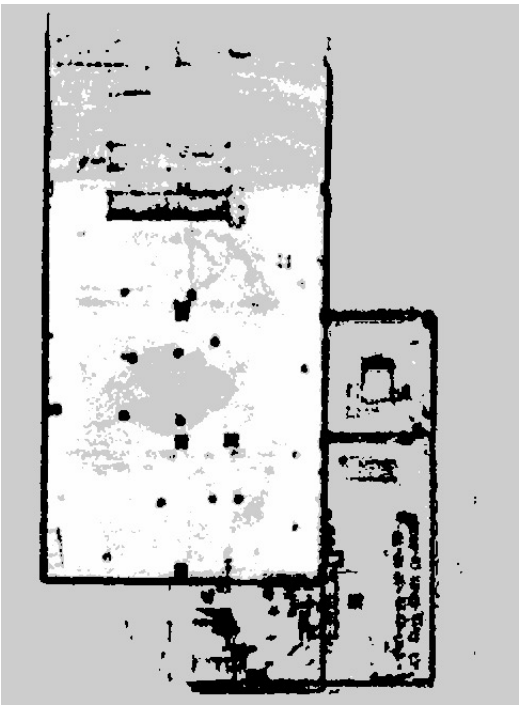
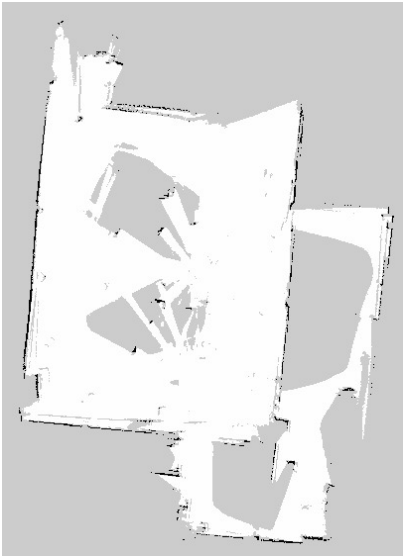


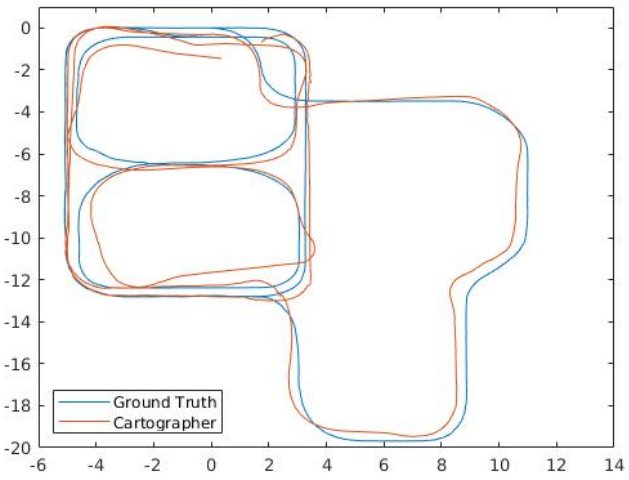
Figure 4.10: Map given by Cartographer and the VLP-16.

Using Intel RealSense L515

The odometry used was not provided by the L515, but instead from the VLP-16 and the IMU. This map was done using 2D scans with the default parameters.



(a) The map



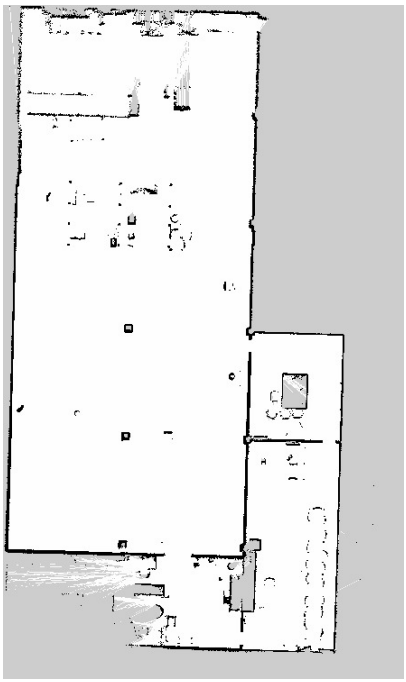
(b) The trajectory

Figure 4.11: Results given by Cartographer and L515.

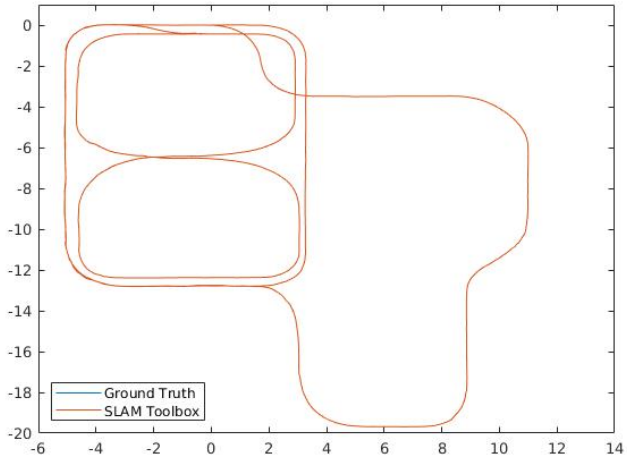
4.6 SLAM Toolbox

SLAM Toolbox is utilized with asynchronous mode, the parameters defined by the .yaml file stayed the default except the mapping of both frames and topics, and also the maximum range was adjusted for the VLP-16 (increased to 50 m). Map and trajectory are seen in Figure 4.12.

Using Velodyne VLP-16



(a) The map



(b) The trajectory

Figure 4.12: Results given by SLAM Toolbox and VLP-16.

4.7 Results overview

During the tests, the VLP-16 did not observe several of the objects in the horizontal scan, because it was placed at a higher height than those objects. Utilizing the raw Intel RealSense L515 maps to be compared to the ground truth would wrongly skew the results, since the later was built using the VLP-16. The objects that do not appear on the VLP-16 scan were manually removed (an example can be seen in Figure 4.13).

(a) The map **(b)** The map after

Figure 4.13: Example of manual changes to the map

The following tables use methods described in Section 3.9.2

Table 4.2: Error regarding maps built by Velodyne VLP-16.

Distance between closest points (cells)	HectorSLAM	Gmapping	RTAB-Map	Cartographer
Mean	0.122	0.7244	4.3246	2.8163
RMSE	0.840	0.9612	5.7875	4.1857

HectorSLAM produced the best results, followed by Gmapping, Cartographer and RTAB-Map. It can be seen that Table 4.2 corroborates the fact that 2D grid maps built from 3D maps seem to be less precise.

Table 4.3: ATE regarding trajectories using the Velodyne VLP-16.

Absolute Trajectory Error	HectorSLAM	Gmapping	RTAB-Map	SLAM Toolbox
Mean (m)	0.0245	0.0384	0.6862	0.0005
RMSE (m)	0.0359	0.0513	1.1618	0.0056
Mean (rad)	0.0199	0.0363	0.2208	0
RMSE (rad)	0.0284	0.0573	0.3430	0.0003

Table 4.4: RPE regarding trajectories using the Velodyne VLP-16.

Relative Trajectory Error	HectorSLAM	Gmapping	RTAB-Map	SLAM Toolbox
Mean (m/s)	0.0290	0.026	0.0532	0.0025
RMSE (m/s)	0.0430	0.0474	0.0985	0.0099
Mean (rad/s)	0.0160	0.0270	0.0575	0.0019
RMSE (rad/s)	0.0310	0.0644	0.0845	0.0012

The SLAM Toolbox performed outstandingly, HectorSLAM and Gmapping did decently while RTAB-Map did not manage to keep up as nicely.

Table 4.5: Error regarding maps built by Intel RealSense L515.

Distance between closest points (cells)	Gmapping	RTAB-Map	Cartographer
Mean	6.8823	2.7299	4.4019
RMSE	9.9719	4.0466	6.1173

RTAB-MAP showed the least error, followed by Cartographer and Gmapping. Considering that the resolution is 0.05 m, the error overall is in the decimeters.

Table 4.6: ATE regarding trajectories using the Intel RealSense L515.

Absolute Trajectory Error	Gmapping	RTAB-Map	Cartographer
Mean (m)	1.6056	0.3288	0.8716
RMSE (m)	1.9493	0.3992	1.0062
Mean (rad)	0.4186	0.2138	-
RMSE (rad)	0.5932	0.2873	-

RTAB-Map with RGB-D odometry performed very well, while Gmapping, as expected, did not do as well (Gmapping is not meant to be used to obtain trajectories).

Table 4.7: RPE regarding trajectories using the Intel RealSense L515.

Relative Trajectory Error	Gmapping	RTAB-Map	Cartographer
Mean (m/s)	0.2261	0.0309	0.1163
RMSE (m/s)	0.4862	0.0462	0.3947
Mean (rad/s)	0.1180	0.0493	-
RMSE (rad/s)	0.2201	0.0691	-

4.8 Movement and object detection

One of the points of filtering people or moving obstacles from scans is to validate if their removal improves ICP and RGB-D odometry in dynamic environments. Because during these tests the odometry given by the L515 was barely usable, only the segmentation and removal could be tested and not the effects on pose estimation.

4.8.1 Using point clouds

An example of a person being removed using Matlab and the process described in Section 3.8.2 can be seen in Figure 4.14.

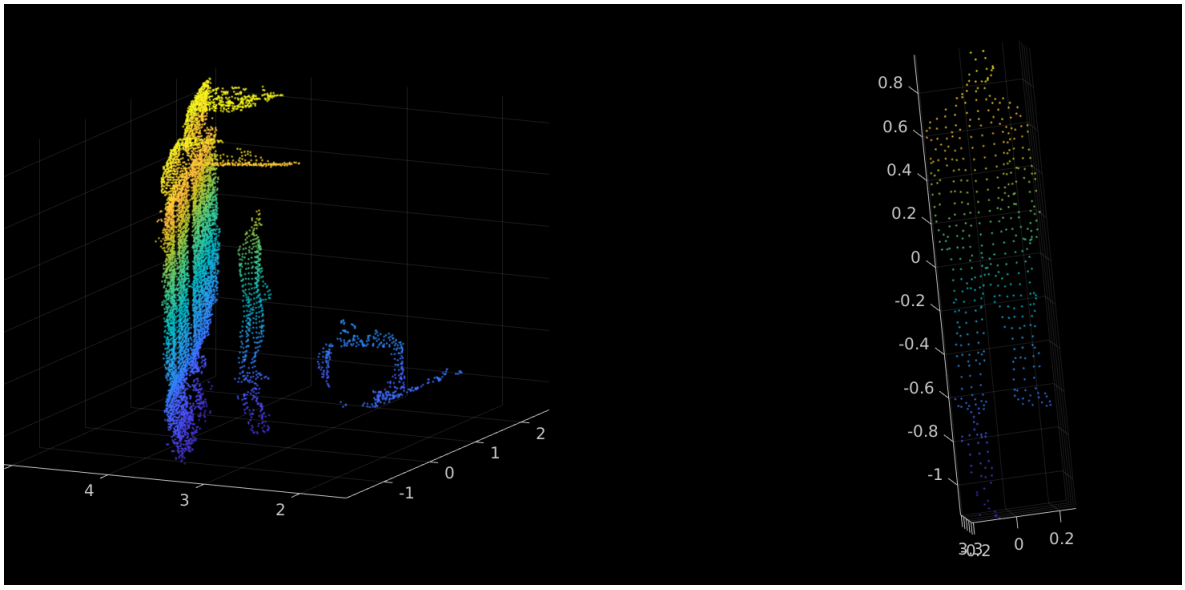


Figure 4.14: Extraction of person cluster in point cloud.

In the case above, the ground partially reflected the legs too, which is included in

the isolated cluster. As an exercise to test the segmentation in a real setting, a loop around the pillars has been done while removing humans from the point cloud (and laser scan) used for mapping. Two videos were recorded showing the process[15][16]. In these videos, the red scan (therefore the map) is created after removing people.

Matlab's processing speed, in this case, is poor; just transferring pointclouds from ROS to Matlab is a slow operation and tests indicated that transferring an Intel RealSense L515's pointcloud to Matlab followed by sending it back to ROS slows the frequency of scans from 30 Hz to around 20 Hz. A possible explanation might have to do with the high resolution of the Intel RealSense L515 lidar and the fact that the format of point clouds in both softwares are different and need conversion twice for each message.

This method is far from ideal, because to allow it to run at decent speeds, downsampling of point clouds must be done. Because of the downsample, increasing the minimum distance between points that is used to define clusters is required. If the person is too close to other objects, both may be removed for being a part of the same cluster, or the person is not removed at all since the cluster's dimensions do not fit the dimensions of a person.

4.8.2 Movement detection

In this subsection, the approach explained in Subsection 3.8.2 is showcased. Figure 4.15 shows a person, that is walking, being detected by this method (the bounding cuboid is shown), and a video[19], where its velocity is calculated, is also available.

The method performs decently in the test setting, although occasionally presenting a false positive. It is able to detect the moving person even when walking behind the couch (coloured yellow in Figure 4.15).

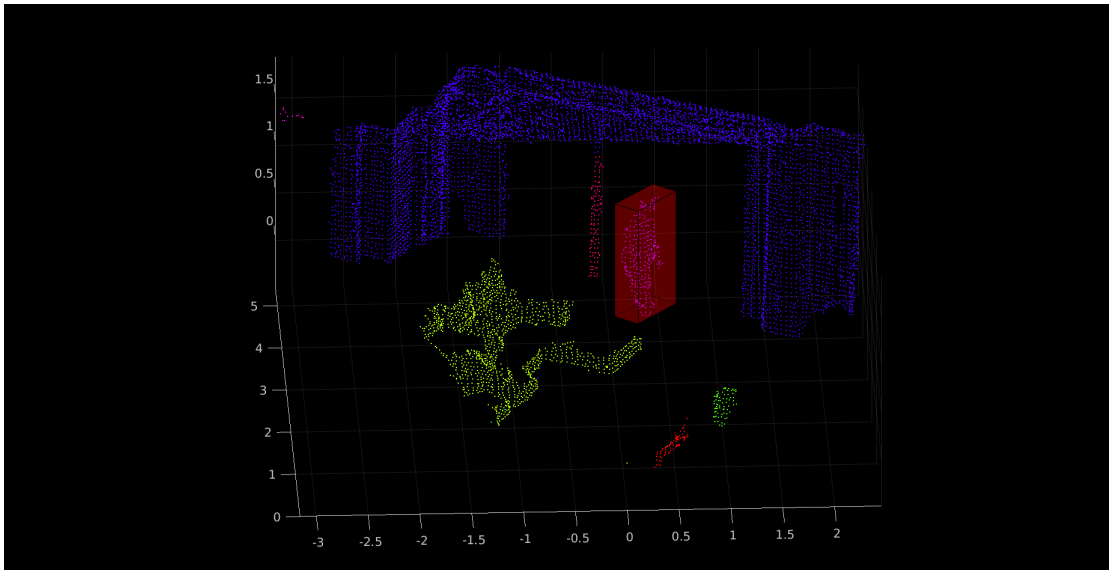


Figure 4.15: A person's movement being detected in a segmented point cloud.

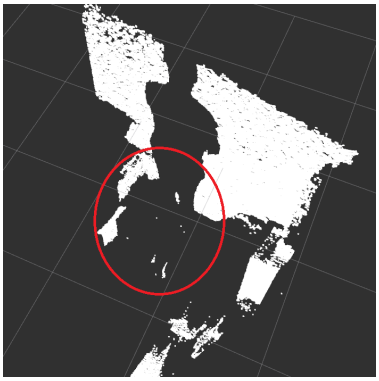
Lastly, this method requires changes if it is to be utilized alongside a moving lidar, since the transformation between the sensor's pose during time $t-1$ to t needs to be taken into account. The position between clusters belonging to point clouds taken at different times must be offset, which can be done by moving the second point cloud into the first's referential. This requires the localization method to exhibit low RPE values, if this error is higher than the velocity threshold selected (to assume movement), then static objects are considered to be moving. Since only short term localization is required, using the estimates given by the EKF that is fusing the odometry might be enough.

4.8.3 Using depth image

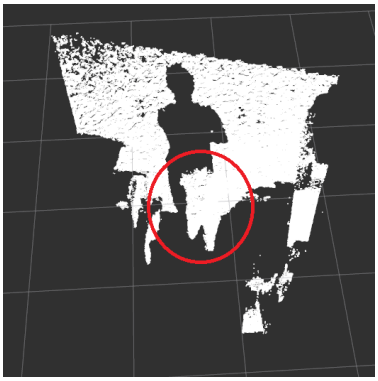
The removal of a person using the method demonstrated in Section 3.8.3 can be seen in Figure 4.16. There is a visible thin line where the person once was. This happens because the alignment between RGB and depth image is not perfect. This effect seems to be even more noticeable during movement, which might indicate the RGB image and depth image have an offset in time.

The publishing frequency of the filtered point cloud barely suffers any reduction. This and the fact that isolating people from other objects is easier, makes this method

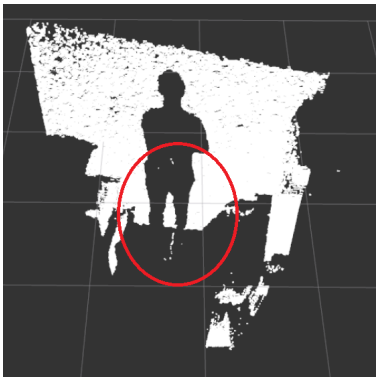
(a) Raw point cloud



(b) Point cloud after removing person



(c) Raw point cloud



(d) Point cloud after removing person

Figure 4.16: Two cases of before and after filtering a person out.

a better solution to filter out people or any objects than the above. However, it requires a calibrated RGB-D sensor.

4.9 Noise removal results

In Section 3.8.4 two methods aiming to remove noise from appearing in the scans were mentioned.

Method 1 presented one problem, which is that clustering is not a perfect method to isolate objects. This leads to certain objects being placed in the same cluster as a wall in one of the point clouds (either the L515 or the D435i). The same object might be correctly isolated as a cluster in the other and it would result in both objects not being associated.

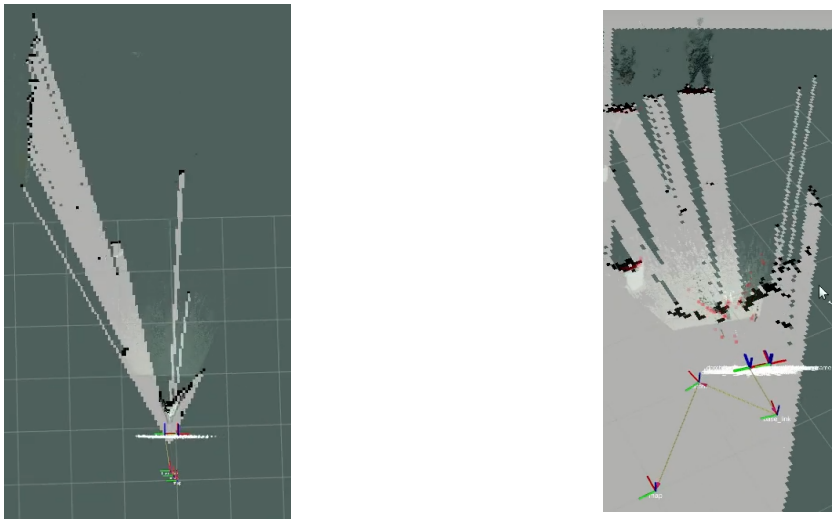
The second approach did not demonstrate the problem above, since it does not need to link objects and, instead, removes any points in the L515 scan that are not near

a cluster in the D435i's scan. The downside is that certain noise clusters may be only partially removed and not the whole cluster since a part of it is close to a real object.

To show how the removal of the noise affects mapping, Gmapping was used to map while a robot performed a loop in the basement. This time, both the Intel RealSense L515 and a Intel RealSense D435i (a depth camera) were placed on top of the robot.

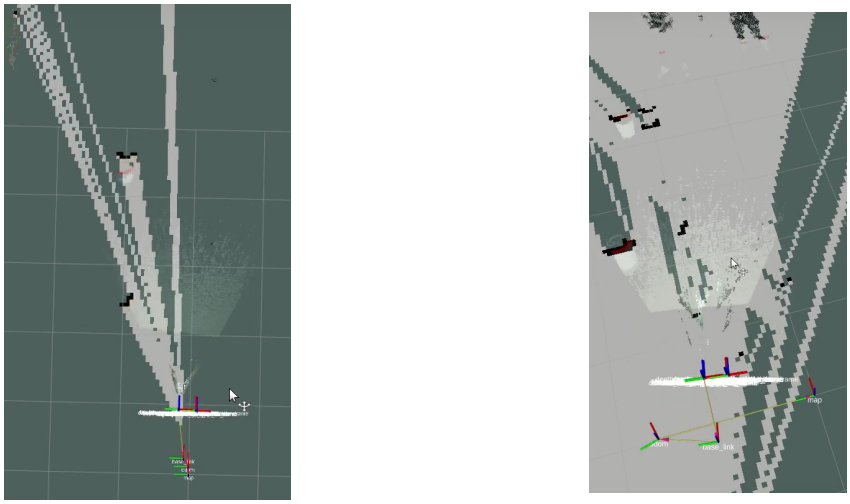
The same data was played back twice, once while performing SLAM using the raw L515 scan and the second time using a processed L515 scan after noise removal, according to the second method described in Section 3.8.4.

Images in Figure 4.17 and Figure 4.18 show the raw and processed point clouds of the L515, and the 2D map being built with and without noise, respectively.



(a) Example 1 **(b)** Example 2

Figure 4.17: Noise being inserted into the map in the first case.



(a) Example 1 **(b)** Example 2

Figure 4.18: Noise being ignored by the map during the second case.

Two videos showing both cases are available, one of the first lap[18], and to the second here[17].

Both implementations, as of now, are done in Matlab. Performing downsampling, cluster segmentation and filtering the noise reduces the frequency of the scan by two thirds (10 scans per second from 30).

Given that during this segment the D435i depth camera was utilized, a comparison with the L515 lidar camera is possible.

4.9.1 L515 and D435i comparison

Both sensors have similar characteristics and are roughly the same price. As it was done with the L515 in Table 3.1, Table 4.8 is obtained under the same conditions while using the D435i.

Real distance (m)	Mean measured distance (m)	Standard deviation (m)
0.5	0.5063	0.0007
2.5	2.6716	0.0174
5	5.3962	0.2211
7	7.3585	0.5594
9	10.9986	0.8732

Table 4.8: Intel RealSense D435i's test measurements.

The L515 is, clearly, both more accurate and precise. However, the lidar does not detect anything farther than 9 m, while the D435i does, even if not reliably. When performing mapping with the depth camera, it might be beneficial to limit the maximum range of the scan, since at 9 m the measurements get wildly uncertain.

The depth camera did not present the noise created by reflective materials and direct light sources seen by the L515 (which is why it was used to discriminate noise). Another difference is that the L515 can not see far through a window, while the D435i can.

In a companion thesis (Pires, 2021[24]), the D435i was used to build a map in the same situation as the tests in this Chapter. The packages used are HectorSLAM, Gmapping and RTAB-Map. The first algorithm did not manage to build a map at all, HectorSLAM could not localize itself right from the start. Gmapping and RTAB-Map made acceptable maps, the main difference seen is that the walls are significantly more blurred. In the case of RTAB-Map, since the map was done using the 3D point clouds, the walls were also less precise than in the Gmapping's case.

Chapter 5

Conclusion

The goal of this work was to accomplish a navigation system for autonomous mobile robots in dynamic environments, either for transportation in factories or disinfection of public spaces.

This work explored six options (HectorSLAM, Gmapping, RTAB-Map, LOAM Velodyne, Cartographer and SLAM Toolbox) to solve the core problem of simultaneous localization and mapping, and compared these different algorithms in a real world setting. Simultaneously, the performance of the Intel RealSense L515 lidar camera and the Velodyne VLP-16 was also evaluated.

The tests showed how powerful the VLP-16 lidar is, all six packages showed satisfactory maps and trajectories (except for LOAM_Velodyne), with SLAM Toolbox and HectorSLAM being the superior of the six. Being in either static or dynamic surroundings did not seem to have a noticeable impact on the result. The L515 lidar could not produce robust localization using ICP, as expected in an environment that includes ample spaces. RGB-D odometry was also tested, but in almost every case the algorithm would lose itself and not recover. ICP odometry using the VLP-16 had to be used in most cases. The case that returned the best results was RTAB-Map, followed by Cartographer. Having someone to move through the scan did not seem to affect the mapping in a discernible way.

Furthermore, a systemic noise problem was found in the data given by the L515, and a possible solution involving pairing it with the D435i depth camera and cross referencing the point clouds was provided and verified.

The last core piece for an autonomous system is the capacity to control its movement after defining a path to a goal. This was implemented and is working as expected.

The velocity for each wheel is computed utilizing inverse kinematics considering a four mecanum wheeled robot, one of the Active Space Technologies' future prototypes.

Lastly, object and people detection algorithms, using the DeepLab model, were integrated into ROS and might now serve as the basis for future features such as interaction with humans. The ability to associate people detected by RGB images to their representation in point clouds was tested. Using cluster segmentation is effective, unless the environment is cluttered, but slow. Filtering unwanted objects or people from the depth image was proven to be a better option performance-wise.

5.1 Future work

Further tests may be done to evaluate a cost-effective solution regarding the selection of sensors. While it is clear that for ample spaces such as warehouses the VLP-16 is superior, it might be excessive in rooms of a smaller scale. A combination of the low-end Intel RealSense lidars and depth cameras paired with wheel encoders might be adequate in these cases. On the topic of hardware, to truly evaluate the Jetson Xavier NX performance, the libraries used by the SLAM packages require inspection to verify if they are taking advantage of the GPU.

The control system has, unfortunately, not been tested in a real world setting. The path finding can be visualized and is seemingly working as expected. As soon as the prototype is built further tests may be performed.

Another test to be done is to investigate if the results given by algorithms that perform odometry, such as RGB-D and ICP-based odometry, benefit from the removal of people in dynamic settings. This needs to be carried out in a small scale environment (average rooms instead of warehouses) where the L515's data is viable for odometry. Better methods to associate similar clusters (beside dimensions) in the movement detection algorithm can be explored, in order to increase robustness.

Any prototype code done in Matlab, such as the noise removal, should be implemen-

ted as a ROS node and make use of libraries like the PCL. Making this change is, certainly, going to improve performance.

References

[1] I. Afanasyev and I. Ibragimov, 'Comparison of ros-based visual slam methods inhomogeneous indoor environment,' Oct. 2017 (cit. on p. 21).

[2] I. Bogoslavskyi and C. Stachniss, 'Efficient online segmentation for sparse 3d laser scans,' *Journal of Photogrammetry, Remote Sensing and Geoinformation Science*, 2017 (cit. on p. 41).

[3] O. K. Bruno Siciliano, *Springer Handbook of Robotics*. 2016 (cit. on p. 15).

[4] W. Burgard, S. Thrun and D. Fox, 'Probabilistic robots,' (cit. on pp. 6, 9).

[5] A. Censi, 'An icp variant using a point-to-line metric,' [Online]. Available: `https://www.researchgate.net/publication/4341089_An_ICP_variant_using_a_point-to-line_metric` (cit. on p. 31).

[6] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff and H. Adam, 'Encoder-decoder with atrous separable convolution for semantic image segmentation,' *arXiv:1802.02611*, 2018 (cit. on p. 40).

[7] M. Filipenko and I. Afanasyev, 'Comparison of various slam systems for mobile robot in an indoor environment,' (cit. on pp. 22, 23).

[8] D. Fox, W. Burgard, F. Dellaert and S. Thrun, 'Monte carlo localization: Efficient position estimation for mobile robots,' in *Proc. of the National Conference on Artificial Intelligence*, 1999 (cit. on p. 10).

[9] G. Grisett, R. Kümmerle, C. Stachniss and W. Burgard, 'A tutorial on graph-based slam,' (cit. on pp. 18, 19).

[10] G. Grisetti, C. Stachniss and W. Burgard, 'Improved techniques for grid mapping with rao-blackwellized particle filter,' (cit. on p. 33).

[11] W. Hess, D. Kohler, H. Rapp and D. Andor, 'Real-time loop closure in 2d lidar slam,' (cit. on pp. 22, 35).

[12] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss and W. Burgard, 'Octomap: An efficient probabilistic 3D mapping framework based on octrees,' *Autonomous Robots*, 2013. DOI: `10.1007/s10514-012-9321-0`. [Online]. Available: `http://octomap.github.com` (cit. on pp. 11, 23, 36).

[13] S. Kohlbrecher, O. von Stryk, J. Meyer and U. Klingauf, 'A flexible and scalable slam system with full 3d motion estimation,' 2011 (cit. on pp. 22, 33).

[14] M. Labbé and F. Michaud, 'Appearance-based loop closure detection for online large-scale and long-term operation,' (cit. on p. 34).

[15] P. Leal, 'Human being filtered out of map,' 2021. [Online]. Available: https://www.youtube.com/watch?v=L_-cDX3b2lw (cit. on p. 65).

[16] ——, 'Human being filtered out of the map 2,' 2021. [Online]. Available: https://www.youtube.com/watch?v=jPFoVmEGs90 (cit. on p. 65).

[17] ——, 'Map with filtered scans,' 2021. [Online]. Available: https://youtu.be/G0NlPFLYgec (cit. on p. 69).

[18] ——, 'Map with raw scans,' 2021. [Online]. Available: https://youtu.be/2S9ZgcpWZo8 (cit. on p. 69).

[19] ——, 'Movement detection using poin cloud,' 2021. [Online]. Available: https://www.youtube.com/watch?v=mx4EFJ23nZk (cit. on p. 65).

[20] T. Linder, S. Breuers, B. Leibe and K. Arras, 'On multi-modal people tracking from mobile platforms in very crowded and dynamic environments,' [Online]. Available: https://github.com/spencer-project/spencer_people_tracking (cit. on p. 40).

[21] S. Macenski and I. Jambrecic, 'Slam toolbox: Slam for the dynamic world,' (cit. on p. 36).

[22] Matlab. (). 'Estimate robot pose with scan matching,' [Online]. Available: https://www.mathworks.com/help/nav/ug/estimate-robot-pose-with-scan-matching.html (cit. on p. 30).

[23] matlabbe, 'Rtab-map's launch file for velodyne vlp-16,' [Online]. Available: https://github.com/introlab/rtabmap_ros/blob/master/launch/tests/test_velodyne.launch (cit. on p. 56).

[24] M. Pires, 'Natural navigation solutions for amrs and agvs using depth cameras,' 2021 (cit. on p. 70).

[25] J. M. Santos, D. Portugal and R. P. Rocha, 'An evaluation of 2d slam techniques available in robot operating system,' in *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, 2013, pp. 1–6. DOI: 10.1109/SSRR.2013.6719348 (cit. on p. 21).

[26] S. Se, D. Lowe and J. Little, 'Local and global localization for mobile robotsusing visual landmarks,' (cit. on p. 10).

[27] H. Taheri, B. Qiao and N. Ghaeminezhad, 'Kinematic model of a four mecanum wheeled mobile robot,' Mar. 2015 (cit. on p. 38).

[28] S. Thrun, M. Montemerlo, D. Koller, B. Wegbreit, J. Nieto and E. Nebot, 'Fastslam:an efficient solution to the simultaneous localization and mapping problem with unknown data association,' (cit. on pp. 16, 17).

[29] R. Yagfarov, M. Ivanou and I. Afanasyev, 'Map comparison of lidar-based 2d slam algorithms using precise ground truth,' (cit. on p. 23).

[30] J. Zhang and S. Singh, 'Loam: Lidar odometry and mapping in real-time,' (cit. on p. 35).