VRIJE
UNIVERSITEIT
BRUSSEL

Graduation thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Applied Sciences and Engineering: Toegepaste Informatica

COMPARISON OF SLAM SOLUTIONS WITH EXCLUSIVE USAGE OF LIDAR OBSERVATION DATA

BEN BERGS

Academic year 2020–2021

Promoter: Prof. Dr. Beat Signer
Advisor: Maxim Van de Wynckel

Faculty of Sciences and Bio-Engineering Sciences

VRIJE UNIVERSITEIT BRUSSEL

Afstudeer eindwerk ingediend in gedeeltelijke vervulling van de eisen voor het behalen van de graad Master of Science in de Ingenieurswetenschappen: Toegepaste Computerwetenschappen

COMPARISON OF SLAM SOLUTIONS WITH EXCLUSIVE USAGE OF LIDAR OBSERVATION DATA

BEN BERGS

Academiejaar 2020–2021

Promoter: Prof. Dr. Beat Signer
Advisor: Maxim Van de Wynckel

Faculteit Wetenschappen en Bio-ingenieurswetenschappen

Abstract (EN)

Simultaneous Localisation And Mapping (SLAM) is a problem in which a map of an unknown environment is continuously updated or created while keeping track of the location of a moving entity within this environment. As different kinds of algorithms exist which try to approximate a solution to this problem, different solution implementations exist.

Many of the known SLAM solutions commonly require different types of input data such as distance measurements, odometry and orientation. This thesis revolves around the benchmarking and comparison of a selection of existing SLAM solutions when LiDAR observations are the only available data.

First an overview is provided on manners to perform indoor mapping and modelling which is then followed by an explanation of different algorithms which provide a solution to the SLAM problem. After this theoretical clarification of possible approaches, the selected SLAM solutions and their implementations are introduced.

Further an explanation is given on how the SLAM solutions were technically implemented and how the LiDAR observations should be interpreted.

A proposal for evaluating the mapping results is introduced and applied on the output generated by the SLAM implementations.

Based upon the information gathered within this thesis a conclusion is provided on the strengths and weaknesses of each of the SLAM solutions.

Abstract (NL)

Simultaneous Localisation And Mapping (SLAM) is een rekenkundig probleem waarbij een kaart gecreëerd wordt van een onbekende omgeving, gelijktijdig wordt de locatie bepaald van een entiteit dat zich voortbeweegt binnen deze omgeving. Er bestaan verscheiden algoritmes die een oplossing van dit probleem proberen te benaderen. Hierdoor bestaan er ook verschillende implementaties gebasseerd op deze algoritmes.

Verscheiden uitwerkingen van het SLAM probleem die publiek beschikbaar zijn, gebruiken meerdere soorten data als input. Het betreft dan onder andere afstandsmetingen, odometrie en oriëntatie. Deze thesis draait om het vergelijken en beoordelen van bestaande SLAM implementaties wanneer zij louter afstandsmetingen, gemaakt door een LiDAR, als input krijgen.

De thesis begint met een oplijsting van manieren voor het uitvoeren van 'indoor mapping and modellinging'. Vervolgens worden verschillende algoritmes toegelicht die een oplossing bieden voor de SLAM probleemstelling. Na deze theorie te hebben bestudeerd, worden de geselecteerde SLAM oplossingen voorgesteld en hun specifieke implementatie wordt kort uiteengezet.

Verder wordt er een toelichting gegeven rond de technische implementatie van de SLAM oplossingen alsook een overzicht van de technische omgeving die werd opgezet hiervoor. In hetzelfde hoofdstuk wordt een voorstelling gegeven rond de fysieke omgeving waarbinnen observaties werden gemaakt met een LiDAR. Eveneens wordt er uitgelegd hoe de LiDAR observaties dienen te worden geïnterpreteerd.

Een voorstel rond het evalueren van de resultaten die voortkomen uit de geïmplementeerde SLAM oplossingen zal dan worden toegelicht. De elementen uit dit voorstel zullen dan worden toegepast en bekeken.

Gebasseerd op de bevindingen in de thesis zal er een conclusie geschreven worden. Deze zal de sterktes en zwaktes van de geselecteerde implementaties toelichten en een advies proberen te geven rond het maken van een doordachte keuze bij de selectie van een oplossing.

Acknowledgements

During the creation of this thesis I received help from different people. Within this acknowledgement I want to thank them for their direct or indirect contribution.

I wish to express my sincere gratitude to Maxim Van De Wynckel and prof. Beat Signer for supporting the idea of this thesis. An extra note of appreciation goes to Maxim for his close guidance during the entire life cycle of this thesis. He has helped me with his valuable guidance to direct the thesis into its final form.

I am much thankful towards all teachers and professors who have educated me during the past years. They provided me the knowledge needed to construct this thesis.

A special thanks goes to my wife and children who have provided me with moral support and understanding during the making of this work.

Contents

1 Introduction

Due to the increase of machine-to-machine interfaces and the Internet of Things (IoT) over the last years, the desire for the ability to determine a location within an unknown environment has increased as well [1]. Global Positioning System (GPS) is widely known and used for outdoor navigation. However, in many environments and situations, GPS signals are not available or reliable: under water, indoor, remote planets, or urban canyons [2].

Indoor localisation often relies on infrastructure present within the building such as wireless access points, Bluetooth/radio beacons, or specifically designed hardware [3]. These systems have multiple constraints: buildings need to equip themselves with infrastructure (a repeated cost for every building), infrastructure should remain stable, and offline training is required [3].

Due to the above, simultaneous localisation and mapping (SLAM) is a problem which has attracted much attention in the robotics research community. SLAM addresses the problem of a robot being placed in an unknown environment and having to build a map, using only relative observations of the environment. While building an accurate map, the robot would simultaneously use this map to navigate through its surroundings. Being able to combine these properties, would make a robot truly autonomous [4, 5].

To be able to create relative observations of the surrounding, a sensor needs to be used. While a wide range of usable sensors is available, in this thesis it is opted to explore the use of a LiDAR (Light Detection And Rang-

ing). LiDARs are used to determine ranges of distance by using a laser to travel to a target object. Different approaches can be used to determine the distance between the laser pulsing sensor and an object. Pulsed ranging, records the travel time of a laser pulse until it reaches a target, while continuous wave ranging, records a phase change in the transmitted sinusoidal signal by broadcasting a continuously emitting laser [6].

As a fair amount of research has happened already about the SLAM problem, several solutions and possible theoretical approaches to it exist within the known literature. One can find different packages online which provide an implementation for the SLAM problem which can be implemented onto multiple platforms and sensor configurations. Most theoretical and technical solutions to the SLAM problem use both the odometry data of the motion sensors (usually attached to a robot) and the scanning results generated by a distance sensor. Due to circumstances (e.g. connection problems between motion sensors and receiver, or absence of motion sensors), it might occur that no odometry data is available. This means that the SLAM solution can only rely on readings of the distance sensor. The goal of this thesis is to compare different online available packages which provide a SLAM solution without the use of odometry data and decide which performs best within our testing environment.

The results of this research will be presented within this thesis. Within Chapter 2 an overview is provided of three main topics. As mapping and modelling an environment is an important part of solving the SLAM problem, this will be discussed first. Secondly different algorithms and approaches for solving SLAM are presented. The chapter ends by introducing the selected SLAM solutions and their specific implementation.

Chapter 3 provides the technical implementation of the SLAM solutions and an overview of the environment on which this is executed. The environment scanned by the selected LiDAR is described within this chapter as well. Different LiDAR observations and their interpretation will be explained.

When reading Chapter 4, one will find a proposal for evaluating the output of the different implemented SLAM solutions. This evaluation technique is then applied to the selected solutions so that data exists on which conclusions can be drawn. The conclusions and possible future work is then provided within Chapter 5.

2

Background and Related Work

2.1 Indoor Mapping and Modelling (IMM)

To be able to navigate inside a building, a map can be a helpful tool. Ideally the user (human or non-human) of a map is provided with a plan which is as accurate and real time as possible. Creating and maintaining a map manually requires a lot of labour such as measuring, drawing, and time. Whenever the infrastructure of a building changes, the manual process needs to be fully or partially repeated and changes to the original plan need to be made. As redesigning a floorplan is a mostly repetitive task, this makes a good candidate to be automatised. Most of the solutions for outdoor map generation cannot be applied to indoor situations as signals used for outdoor map generation (e.g. satellite) have difficulties travelling through building walls.

To determine one's location within a map, this map needs to exist. Different approaches exist for building such a map. As the focus of this thesis is to be able to find the most optimal solution for the indoor SLAM problem, the focus will be on the indoor map generation and not on other variants of automatically generated maps. Different research and implementations exist nowadays to provide a solution to this problem. Each of these solutions are using different approaches on how signals and sensors can be used to create an accurate map of an indoor environment.

2.1.1 Radio Propagation

Radio propagation uses devices which transmit radio signals and other devices which receive radio signals. By measuring the strength of the signals picked up by the receivers, the distance between the emitter and receiver can be calculated. When applying this to indoor mapping applications, there are two basic approaches to this design. A building equipped with devices in known locations are broadcasting radio signals wile mobile devices listen to the emitted signals. The strength of each of the signals is measured and used to determine the distance to each of the known radio transmitters. In the other approach, a building is equipped with radio signal receivers at known locations in which a mobile device navigates through the building while broadcasting radio signals. By measuring the signal strengths, the signal receivers can determine the location of the mobile device. In both approaches, mobile device is followed as it moves through the building. By assuming that device cannot move through walls, it is possible to make a plan of the building.

Yiming Ji, et al. [7] state that all radio propagation models can grossly be grouped into three categories: Simple attenuation model, Partition model, and Site-specific model.

Simple attenuation model is the base of all other models and uses Equation 2.1.

$$P\left(d\right)\left[dB\right] = P\left(d_0\right)\left[dB\right] - 10 \times n \times log_{10}\left(\frac{d}{d_0}\right) \tag{2.1}$$

where $P\left(d\right)$ is the power of the radio signal at distance d to the transmitter in metres; $P\left(d_0\right)$ is the power at a reference distance d_0. n is the attenuation exponent, which is often statistically determined to provide a best fit with measurement readings.

The partition model considers the attenuation effect from buildings such as the loss of signal strength due to walls and floors. The idea is to reduce the effect of the attenuation exponent n from the Equation 2.1

The site-specific model is based on the same concept as partition models but considers site-specific conditions such as wall thickness, floortype and materials.

Yiming Ji, et al. [7] state as well that all three categories hold the same shortcomings:

- Determining the attenuation exponent requires extensive measuring.
- The dynamic behaviour of radio signals are not taken into account. Even when the mobile device does not move within the building, the signal strength which is transmitted/received will vary.
- Signal attenuation is only considered along the direct path between signal transmitter and receiver.

2.1.2 Range Sensors

Maps of indoor environments can be created by the use of range sensors as well. These sensors will scan the environment and measure the distance between itself and nearby obstacles (e.g. walls, furniture). By making measurements at different locations within a building and combining this data, an accurate map can be generated. Many different kinds of range sensors exist, each varies in e.g. complexity, size, accuracy and range. An overview of a few of the most popular types of range sensors can be found below together with a short summary explaining the used approach for measuring.

Ultrasonic Sensors

Ultrasonic distance sensors are built to create non-contact distance measurements. They exist out of a transmitter which broadcasts ultrasonic sounds and a receiver which captures ultrasonic sounds. These kind of sensors create measurements based on the time-of-flight principle. The transmitter sends out ultrasonic sound signals, these signals will be reflected to the receiver when they collide with an object as almost all materials reflect sound waves. The distance between the sensor and an object is then calculated by considering the time taken by the ultrasonic wave to be reflected to the receiver. This principle is illustrated in Figure 2.1.

Ultrasonic sensors do not require any contact with the target object and are therefore classified as non-intrusive. In contrast to vision or light based sensors, they are able to detect transparent or shiny objects. Overall these kind of sensors are not expensive and have a reliable precision. As a downside, these sensors are sensitive to environmental temperature and humidity as these affect the speed of sound. The object reflecting the sound waves to the sensor should be perpendicular to the receiver. When a flat object is placed within an angle to the sensor, sound waves might be reflected towards

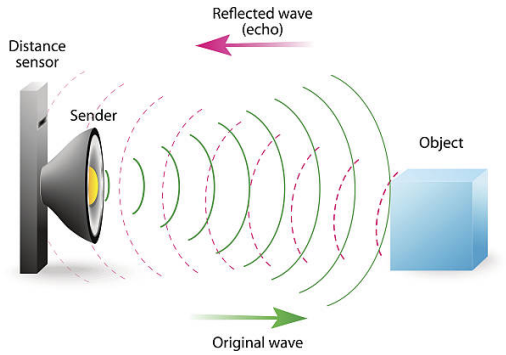


Figure 2.1: Illustration of ultrasonic distance sensors [8]

another direction then the sensor. A similar problem arises when the object reflecting the sound waves is not a flat surface. Another disadvantage when using ultrasonic sound sensors exist in the inability of detecting objects which are very close to the sensor as they deflect the wave before the receiver is ready [9].

Infrared Sensors

Like ultrasonic sensors, infrared (IR) sensors are non-intrusive distance sensors. Though different techniques exist to use IR sensors for distance measurements, only two will be discussed shortly in this thesis: distance calculation using the reflective technique and distance calculation using triangulation.

The reflecting technique of IR distance measurement is very similar to the technique used for ultrasonic (US) distance measurements. An infrared light source such as a LED is used to send out an infrared beam. The light which collides with an object is sent back to the sensor as most materials reflect IR light. Where US sensors use the time-of-flight to determine the distance the sound wave travelled, IR sensors use the strength of the reflected light to calculate the distance of the object reflecting the IR light. When using this technique, the measurements are depending on the reflectivity of the obstacle. On the other hand, small orientation of the object is not a problem to determine the distance between the sensor and the object [10].

When using the triangulation technique, a light source sends out an infrared beam. Just like the reflecting technique, the sensor is used to capture the light reflected by an obstacle. Where the reflecting technique uses the strength of the received light, the triangulation technique uses the angle in which the light is returned to the sensor as displayed in Figure 2.2. The angle in which the emitted light is returned to the sensor is dependent on

the distance to the object [11]. The reflective capability of the object is irrelevant with this technique as long as the obstacle reflects the light towards the sensor. The angle in which a flat object is positioned towards the sensor has an impact on the calculations.

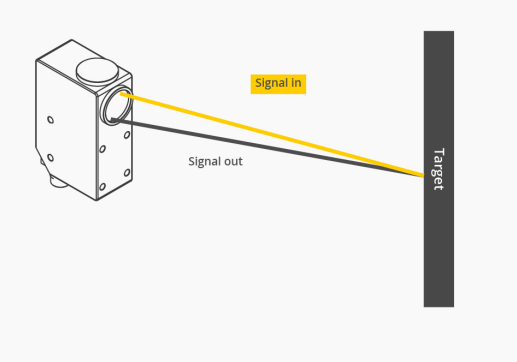


Figure 2.2: Image of a signal sent and received by a sensor using the triangulation technique [12]

Overall, IR sensors are reliable and provide a good resolution [10, 11]. However, this type of sensor is expensive and has problems detecting objects which are transparent or heavily reflecting such as glass or mirrors.

Stereo Cameras

Stereo cameras have been widely used for measuring depth in photogrammetry. Within this domain, measurements are mainly taken from static scenes. In augmented reality and robotics, accuracy of depth measurements have become more important [13].

Stereo cameras try to mimic the behaviour of human eyes. Figure 2.3 displays the technique of stereo cameras. By capturing two images of the same scene at slightly different locations, it becomes possible to calculate the distance towards the object. This approach allows a 3D view of the scene to be created as well. Both cameras will capture a certain point in a different projection line. By comparing the angle in which this point is captured from both of the cameras and the use of triangulation, the distance is determined. By making this calculation for every point of the scene, a 3D image can be generated. The challenge for stereo cameras lies in matching the points captured in both of the images [14].

Stereo cameras have problems detecting transparent object (such as glass), they are sensitive to the lighting of the scene (as this influences the point-matching-process). Typically these sensors appear to be more expensive

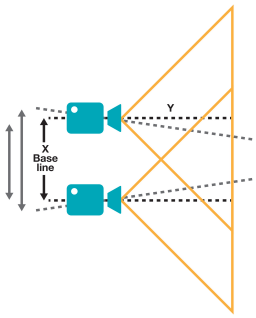


Figure 2.3: Stereo cameras for distance measurement [15]

than ultrasonic and infrared sensors and are usually quite large. On the other hand, these types of sensors provide very good resolution and are very reliable.

Laser Range Finders

Laser range finders or Light Detection and Ranging (LiDAR) systems are possibly the most accurate sensors for distance measurements. The concept behind LiDAR systems is similar to some of the other discussed range sensors. A signal, in this case a light signal, is sent into the environment. When the signal collides with an object, it is reflected to the receiver of the sensor. Although the concept is fairly simple, multiple approaches exist to broadcast the light signal and determine the distance to the detected obstacle. The two main existing techniques are: Pulsed modulation and Continuous Wave (CW).

Pulsed Modulation is based on the time-of-flight principle just like the ultrasonic sensors [16]. The sensor emits a laser beam which is reflected by an object to the receiver. By measuring the time between transmitting and receiving of the signal, the distance between the sensor and the object is calculated. Keep in mind that light travels at an approximate speed of $300'000'000m/s$ while sound travels at an approximate speed of $343m/s$. It is easy to see that the response time of light based technology is much faster than the response time of sound based technology.

Continuous Wave can be split into two mainly used approaches: Amplitude Modulation Continuous Wave (AMCW) and Frequency Modulation Continuous wave (FMCW).

AMCW sends out a continuous modulated signal. The light returned to the sensor has the same amplitude but by bouncing of an obstacle, the

phase has shifted. By determining the phase shift between the broadcasted and reflected signal, the distance between the sensor and the obstacle can be calculated [17].

FMCW sends out a frequency modulated signal. The signal bounces of an obstacle and is sent back to the sensor. The reflected signal is then mixed with the signal which is send. The result is the difference in frequency between the send and the received signal. By using this result, the distance of the obstacle can be calculated [18]. Figure 2.4 shows the difference between ToF, FMCW and AMCW.

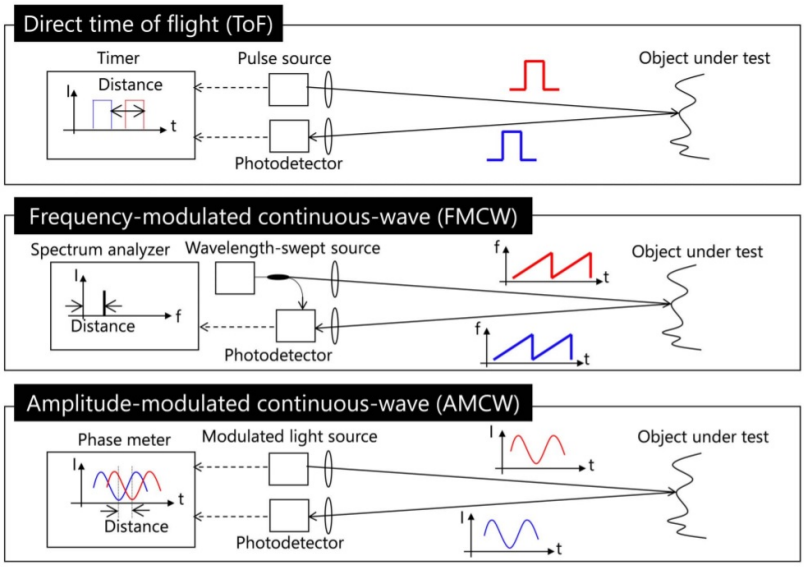


Figure 2.4: Ranging concept of direct ToF, FMCW, and AMCW laser scanner [19]

.

Overall LiDAR systems are reliable and have a good resolution. Just like IR red sensors, they have problems detecting transparent objects. Typically these types of sensors are quite expensive in comparison to the other sensors.

2.1.3 Wi-Fi Fingerprinting

Wi-Fi fingerprinting is based on the same theory as radio propagation. As it is a technique used in many papers and research, it is separately mentioned within this thesis. Nowadays, most buildings are equipped with several Wi-Fi Access Points (AP) which broadcast signals to receivers (mostly smartphones and laptops). These devices receive signals from different APs with a certain received signal strength (RSS). When combining the MAC address and the

RSS of at least three APs, it is possible to determine the location of a certain receiver within the building.

A *followed path* can be generated by tracking a device as it moves within a building. When combining many of the paths e.g. by using crowd sourcing technique, an accurate floorplan can be generated [1].

This approach has several challenges when implementing: Wi-Fi fingerprints tend to be noisy as mobile devices are dynamic and noisy; a decent crowdsourcing algorithm needs to be created and; the quality of the map is depending on the amount of data generated from the crowd [20].

2.1.4 Combining Sensors

Research and papers exist [e.g. [21, 22]] in which many sensors of devices are combined. Often these papers focus on smartphone devices which have a wide range of sensors e.g. GPS, accelerometer, Wi-Fi, Bluetooth. By using the data received from these sensors and using a combination of techniques such as cell-ID, fingerprinting, crowd sourcing, GPS tracking, and Bluetooth propagation, it is possible to generate accurate maps [23]. The discussion of these methods is out of scope for this thesis.

2.2 Simultaneous Localisation and Mapping (SLAM)

Many research and papers exist already about SLAM as it is a problem which has attracted much attention within the robotics research community. Within this section, a selection of different research is summarised together with their unique implementation to provide a solution for the SLAM-problem.

The idea of SLAM is to estimate the position of the robot and the position of the landmarks (i.e. the map) in the environment. Figure 2.5 is an example of a map generated by applying SLAM. The process of estimating a map cannot be decoupled from the process of estimating the trajectory as both variables are dependant of each other. This means that both problems need to be solved at the same time. This is also known as the chicken-or-egg problem in SLAM.

SLAM starts from a sequence of control commands and from sensor observations. The sequence of control commands which the robot executed are expressed as variables $u_{1:T}$. These can be either raw commands (e.g. move at a certain velocity) or these can be odometry readings (e.g. rotary encoders on the wheels of a robot). The sensor observations are expressed as $z_{1:T}$. These

Figure 2.5: Example of a map generated by applying SLAM [24]

observations are readings from a sensor attached to the robot (e.g. LiDAR) which provide information about the environment of the robot.

By using the sequence of control commands and sensor observations, SLAM tries to generate a map m of the environment and the path of the robot expressed as $x_{0:T}$. The variable x exists out of a x and y coordinate when using SLAM in a two dimensional space. Depending on the estimation technique used for SLAM, only x_T is of any interest, this is called online slam.

2.2.1 Extended Kalman Filter

The extended Kalman filter is a popular algorithm used in SLAM applications. To be able to explain the extened Kalman filter, a short introduction of the Bayes filter and the Kalman filter will be provided.

Bayes Filter

A Bayes Filter, is a general probabilistic approach for estimating an unknown probability density function. The filter is recursive and can be split up into two steps: the prediction step and the correction step. The prediction step can be noted by using following formula [25]:

$$\overline{bel}\,(x_t) = \int p\,(x_t|u_t, x_{t-1})\,bel\,(x_{t-1})\,dx_{t-1} \tag{2.2}$$

Whenever the robot has executed a command u_t, the algorithm will estimate its state x_t. This estimation is the so called believe of the state of the robot, which is noted as $\overline{bel}\,(x_t)$. As the robot can never be a hundred percent sure of the correct execution of a command, this believe of the state is a

distribution. For example: The command to the robot might be 'move two metres forward'. It is nearly impossible to move exactly two metres. It might be $1,999999$ metres or $2,0000001$ metres. As for every command the robot executes this uncertainty exists, the deviation of the expected state and actual state increases. The prediction step tries to calculate how the believe $bel\,(x_{t-1})$ of the robot changes when executing the command u_t by performing the command u_t for every x_{t-1} of the distribution of $bel\,(x_{t-1})$ and determining the probable state x_t of the robot based upon this input. This will end up in a new distribution indicating the new believe of the state of the robot $bel\,(x_t)$.

When the algorithm finished the prediction step, it will continue with the correction step. This step is expressed by following formula:

$$bel\,(x_t) = \eta p\,(z_t|x_t)\,\overline{bel}\,(x_t) \tag{2.3}$$

Within this step, the algorithm takes the observations made by the robot, z_t, in account. By relating the actual observations of the robot to the predicted observations $\overline{bel}\,(x_t)$ and determining the mismatch between these two parameters, a correction can be executed to the believe of the state of the robot [26].

Kalman Filter

The Kalman filter is an estimator for unknown variables which requires the model to be linear and the distribution to be Gaussian. The linear model assumes the motion model discussed in Equation 2.4 and the observation model of Equation 2.5 are linear functions.

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t \tag{2.4}$$

$$z_t = C_t x_t + \delta_t \tag{2.5}$$

Within the motion model (Section 2.4), A_t represents a matrix which describes how the state changes from $t-1$ to t without any controls. B_t represents a matrix which indicates how the command u_t changes the state from $t-1$ to t. ϵ_t is a variable which represents the noise.

Within the observation model (Section 2.5), C_t represents a matrix which maps the state x_t to an observation z_t. δ_t is a variable which represents the noise.

The Kalman filter describes the linear motion model with the Gaussian conditional distribution (which will not be explained within this thesis as it is out of scope) and describes the linear observation model with the Gaussian

conditional distribution. By plugging in the linear motion model and the linear observation model into the Bayes filter, the Kalman filter specifies algorithm 2.6 for estimation:

$$\begin{aligned} \overline{\mu}_t &= A_t \mu_{t-1} + B_t u_t \\ \overline{\Sigma}_t &= A_t \Sigma_{t-1} A_t^T + R_t \\ \\ K_t &= \overline{\Sigma}_t C_t^T \left(C_t \overline{\Sigma}_t C_t^T + Q_t \right)^{-1} \\ \mu_t &= \overline{\mu}_t + K_t \left(z_t - C_t \overline{\mu}_t \right) \\ \Sigma_t &= \left(I - K_t C_t \right) \overline{\Sigma}_t \end{aligned} \tag{2.6}$$

The steps in which $\overline{\mu}_t$ and $\overline{\Sigma}_t$ are calculated represent the prediction step. The Calculation of K_t, μ_t and Σ_t represent the correction step. μ_t and Σ_t are respectively the mean and covariance matrix. K_t is the so-called Kalman gain and indicates the certainty of the observations in respect to the executed command. Another way to look at K_t is as a formula to compute a weighted mean between the prediction and observation [27, 28].

Extended Kalman Filter

The Kalman filter assumes the distributions and noise to be Gaussian as it assumes as well that the motion and observation model are linear. In reality, the motion and observation models are rarely linear and often contain an angle or rotation. This results to a non-linear functions in which the motion model 2.7 and observation model 2.8 need to be rewritten by using the non-linear functions g and h.

$$x_t = g\left(u_t, x_{t-1}\right) + \epsilon_t \tag{2.7}$$

$$z_t = h\left(x_t\right) + \delta_t \tag{2.8}$$

The non-linearity of the motion and observation models lead to the destruction of the Gaussian distribution. This results in the Kalman filter becoming unusable as it assumes both functions to be linear for its implementation.

The extended Kalman filter (EKF) resolves this issue by using local linearisation. The linearisation is done by applying the Taylor approximation to our best estimate μ. The details of the Taylor approximation are out of scope of this thesis. It is important to be aware that the linearisation is computed in only one point of the function. When this point changes, the computation needs to be redone.

EKF specifies Equation 2.9 for estimation based upon the Kalman filter and the linearisation of the motion and observation model:

$$\begin{aligned}
\overline{\mu}_t &= g\left(u_t, \mu_{t-1}\right) \\
\overline{\Sigma}_t &= G_t \Sigma_{t-1} G_t^T + R_t \\
\\
K_t &= \overline{\Sigma}_t H_t^T \left(H_t \overline{\Sigma}_t H_t^T + Q_t\right)^{-1} \\
\mu_t &= \overline{\mu}_t + K_t \left(z_t - h\left(\overline{\mu}_t\right)\right) \\
\Sigma_t &= \left(I - K_t H_t\right) \overline{\Sigma}_t
\end{aligned} \tag{2.9}$$

The EKF algorithm differs from the Kalman filter in several points. The formula for $\overline{\mu}_t$ is expressed as a non-linear function g which is evaluated at the linearisation point μ_{t-1}. A_t has been replaced by G_t which represents a Jacobian matrix of the motion which is used within the Taylor approximation for linearisation. C_t has been replaced by the non-linear function h_t for the calculation of μ_t and C_t has been replaced by the Jacobian matrix of the observation H_t in the computation of the Kalman gain [27, 29, 28].

EKF and SLAM EKF is used as an algorithm in solving the SLAM problem. EKF allows the estimation of the robot's state and the locations of the landmarks within the environment. The state of the system is represented by following matrices:

$$\begin{pmatrix} x_R \\ m_1 \\ \dots \\ m_n \end{pmatrix} \begin{pmatrix} \Sigma_{x_R x_R} & \Sigma_{x_R m_1} & \dots & \Sigma_{x_R m_n} \\ \Sigma_{m_1 x_R} & \Sigma_{m_1 m_1} & \dots & \Sigma_{m_1 m_n} \\ \dots & \dots & \dots & \dots \\ \Sigma_{m_n x_R} & \Sigma_{m_n m_1} & \dots & \Sigma_{m_n m_n} \end{pmatrix} \tag{2.10}$$

x_R represents the robot's pose which can be rewritten to (x, y, θ) for a 2D environment. m_1 to m_n represent n landmarks of the environment which can be rewritten as $(m_{1,x}, m_{1,y})$ to $(m_{n,x}, m_{n,y})$ for a 2D environment. The first matrix within Equation 2.10 represents the mean μ of the EKF, the second matrix represents the covariance matrix Σ of the EKF.

When a robot moves through the environment, EKF will create a solution to the SLAM problem as follows. After the execution of a movement, the location of the robot is adapted (take note that it will not change the location of the landmarks). This means that x_R within Equation 2.10 is changed. This change will then lead to an update of the covariance matrix Σ for every entry in which the state of the robot is used. This results in an update of the first row and the first column of the covariance matrix represented in Equation 2.10. This update has a linear complexity in the number of landmarks.

As the robot made a prediction of its state, it is then able to compute a prediction of the measurements it is expecting to obtain. After these predictions, the robot will obtain actual measurements of the environment which can then be compared to the estimated results of the observations. The difference between these two paramaters can then be used to make an update of the mean matrix and covariance matrix of the EKF. This computation has a complexity of $O(n^2)$ [27, 29, 28].

2.2.2 FastSLAM

Another popular SLAM algorithm is FastSLAM. This technique combines particle filters and EKF. Before explaining the FastSLAM technique, a short introduction to *particle filters* will be provided.

Particle Filters

Within this section an introduction will be provided of *particle filters* which can be used for localisation, assuming that the map is already known and does not need to be build. The Kalman filter focuses on linear models and probability distributions which are Gaussian. However, when estimating the pose of a robot within a map, the probability distribution often tends to be arbitrary. To deal with these kinds of distributions, particle filters show to be a good approach. The approach of particle filters is to use multiple samples to represent arbitrary distributions. These samples can be seen as 'points' which live within the x coordinates of the distribution. The more samples within a certain area, the higher the probability of that area region. Figure 2.6 shows how the samples, proposal distribution and target distribution relate. As an addition to the amount of samples, a weight for these samples is used. This way samples with large 'certainty' have a higher probability [30]. Representing the posterior can be done as follows [31]:

$$\chi = \left\{ \left\langle x^{[j]}, w^{[j]} \right\rangle \right\}_{j=1,\ldots,J} \tag{2.11}$$

$$p\left(x\right) = \sum_{j=1}^{J} w^{[j]} \delta_{x^{[j]}}\left(x\right) \tag{2.12}$$

Equation 2.11 represents the set of weighted samples. Every sample can be seen as a state hypothesis (i.e. a possible state the system may be in). This hypothesis is represented by $x^{[j]}$. The importance weight of a state hypothesis is represented by $w^{[j]}$. The higher the amount of samples or the higher the weight of a sample, the higher the probability of the true state

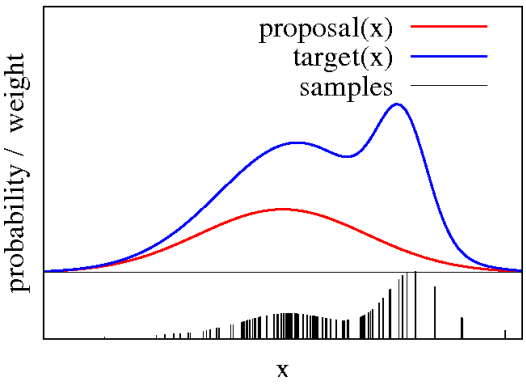


Figure 2.6: Approximation of the target distribution by samples [32]

of the robot matches with these samples. Equation 2.12 shows how the probability distribution obtained by the representation of the sample set.

Drawing samples from a distribution is key within the particle filter. Several techniques exist on how to efficiently draw samples from certain distributions such as the Gaussian distribution. These techniques are out of scope for this thesis. Within the particle filter, the bigger interest is in how to draw samples from arbitrary distributions. *The Importance Sampling Principle* is a technique to do this. This principle defines the possibility to generate samples from a distribution f by using another distribution g. The differences between f and g are accounted for by using a weight, w, in which $w = f/g$ [30, 31].

Particle Filter Algorithm is in fact a recursive Bayes filter with three main steps. First the proposal distribution g is used to create samples. Secondly the importance weights are calculated by comparing the proposal distribution g and the target distribution f. The last step includes resampling all particles by using the calculated weight [31].

Feature-Based SLAM with Particle Filters (FastSLAM)

SLAM tries to estimate the position of a robot, together with all landmark locations. Using Equation 2.13 in which $x_{1:T}$ represents the poses of where the robot has been and $m_{1,x}, m_{1,y}, \ldots, m_{M,x}, m_{M,y}$ represents all landmark locations. It is clear that the SLAM problem is presented in a high-dimensional state space.

$$x = (x_{1:T}, m_{1,x}, m_{1,y}, \ldots, m_{M,x}, m_{M,y})^T \tag{2.13}$$

When using particle filters, the likely regions of the state space need to be covered with samples. Therefore particle filters are effective in low

dimensional spaces. Slam being a high-dimensional state space is a limitation of why particle filters are difficult to be used to solve this problem.
Rao-Blackwellisation states that when variables have dependencies amongst each other, factorisation can be used to exploit these dependencies [33, 31].

$$p(a,b) = p(b|a)\,p(a) \tag{2.14}$$

Equation 2.14 explains that when $p(b|a)$ can be computed efficiently then $p(a)$ can be represented by samples and for every sample, $p(b|a)$ should be computed. Looking at Equation 2.13, it is possible to say that the locations of the landmarks are depending on the pose of the robot. This statement makes is possible to apply Equation 2.13 to the Rao-Blackwell theorem [5].

$$p(x_{0:t}, m_{1:M}|z_{1:t}, u_{1:t}) = p(x_{0:t}|z_{1:t}, u_{1:t})\,p(m_{1:M}|x_{0:t}, z_{1:t}) \tag{2.15}$$

Equation 2.15 shows the probability distribution $p(x_{0:t}, m_{1:M}|z_{1:t}, u_{1:t})$ of the poses, $x_{0:t}$, and the map, $m_{1:M}$, given the observations, $z_{1:t}$, and movements, $u_{1:t}$. Two posteriors are provided when factorising this distribution. $p(x_{0:t}|z_{1:t}, u_{1:t})$ is the map posterior and estimates the path of the robot. $p(m_{1:M}|x_{0:t}, z_{1:t})$ is the map posterior and depends on the trajectory the robot has taken. Note that $u_{1:t}$ is ignored in the map posterior as it is irrelevant. Given the poses of the robot, landmarks are conditionally independent. Therefore it is possible to rewrite Equation 2.15 as the following equation:

$$p(x_{0:t}, m_{1:M}|z_{1:t}, u_{1:t}) = p(x_{0:t}|z_{1:t}, u_{1:t}) \prod_{i=1}^{M} p(m_i|x_{0:t}, z_{1:t}) \tag{2.16}$$

Rewriting Equation 2.15 as Equation 2.16 shows that it is possible to maintain M two dimensional EKFs instead of one large dimensional EKF. As the past poses are not revised within FastSlam, there is no need to maintain these within the sample set. To build the map, only the current state is of any interest. This means only the current pose of the robot is represented within the sample set. Because of this, every particle only needs to maintain three dimensions for the pose and $2 \times n$ dimensions for the landmarks in which every landmark is a 2×2 EKF [5].

FastSLAM is a process which uses four main steps to build a map from the surrounding.

- The path posterior is extended by sampling a new pose for each sample
- Computing the weight of each particle. This is done by using the importance sampling principle.

- The belief of each observed landmark is updated
- Resample

FastSLAM complexity can be concluded to be $O(NM)$ in which N represents the number of particles and M represents the number of map features. This is beacause the resampling is done in a complexity linear to the number of landmarks. By using shared binary search trees between particles, it possible to improve the FastSLAM algorithm to have a computational complexity of $O(NlogM)$ [5]. The explanation of this improvement is out of scope of this thesis.

FastSLAM 2.0

The FastSLAM algorithm previously explained within this thesis, is often refered to as FastSLAM 1.0. This because an improvement to the algorithm has been found which is referred to as FastSLAM 2.0. The concept behind FastSLAM 2.0 will only be explained briefly within this thesis. It has been found that when using an accurate sensor, it is useful to take in consideration the measurements during sampling. This will result in a more peaked proposal distribution and less particles compared to FastSLAM 1.0. It proves that FastSLAM 2.0 is more robust and accurate but more complex [34].

2.2.3 Grid-Based SLAM

The previously discussed solutions to the SLAM problem are based on *features*. Feature-based SLAM assumes that the environment exists out of landmarks and it is possible to detect these landmarks using a sensor. By using the data generated by the sensor, it is possible to calculate the location of the landmarks within a map. Using this data it is possible to provide a solution to the SLAM problem. Another solution to the SLAM problem is grid-based SLAM. To be able to understand this solution, a short overview to grid maps will be provided on which the grid-based SLAM solution is based.

Grid Maps

When using grid maps, the environment is represented by a grid which divides the 'world' into independent cells of a certain size. A cell can either be fully empty or fully occupied [35]. Figure 2.7 shows an example of a grid with occupied and free cells. Using this approach, obstacles detected by a sensor can be rendered directly onto the grid. It is clear that grid maps need more memory than feature based maps as the grid map needs to store for every

cell within the grid if it is free or occupied while feature-based maps only store the locations of the landmarks [36]. When creating grid maps, every cell is assumed to be a binary random variable. This means that a cell can only exist out of two possible values: 1 (occupied) or 0 (not occupied or free). The idea is to estimate the probability of the cell being occupied. If a cell is occupied, the occupancy probability is equal to 1. If the cell is not occupied, the occupancy probability equals to 0. If there is no knowledge of the cell, the occupancy probability equals to $0,5$. Combining the measurements of the sensor and the log odds model (explaining this model is out of scope for this thesis), it is possible to quickly compute the occupancy probability of a cell [37]. When displaying the grid and filling out the information which cell is probably occupied, which cell is probably not occupied and the cells of which no knowledge exists, results in a map of the environment. For building a grid map and using the sensor observations, it is crucial for the robot to know its exact location within the map while interpreting the sensor observations [36]. The assumption of the robot knowing its exact location fails as motion is noisy. Therefore the method described above fails to produce a usable map of the surroundings. It is possible to state that however the motion of a robot is unreliable, the observations done by a sensor are more precise (the precision is dependent of the type of sensor). To resolve the known-poses-failure of the robot, a technique called scan-matching can be used. This technique tries to incrementally align the current information observations made by the sensor, the generated map, ... to previously generated information in combination with the robot's motion [36]. Many different scan-matching variations exist such as scan-to-scan, scan-to-map, map-to-map, ICP.

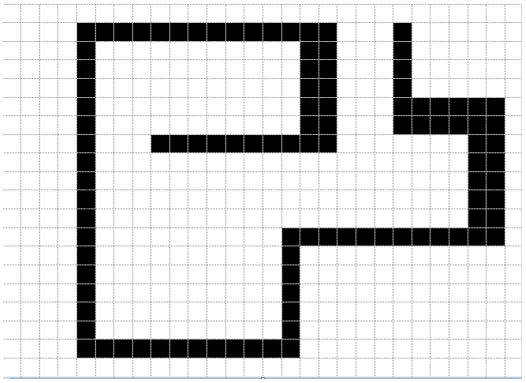


Figure 2.7: A grid with occupied and free cells

Grid-Based FastSLAM

The FastSLAM 1.0 approach explained above, can be applied on grid maps as well. FastSLAM 1.0 created a map of the environment by using the observed landmark information. As the pose of the robot can be determined, it becomes possible to calculate the map. The same concept can be applied for building a map by using a grid map in which a map is built by determining which cells within the grid are occupied or not occupied. FastSLAM is based on the Rao-Blackwellisation principle in which the believe of the pose of the robot and the map given the observations and odometry can be split into two parts. The path posterior which is a particle filter and the map posterior given the path of the robot. As stated previously, it is fairly easy to create a grid map when the poses of the robot are known. Every particle represents a possible path of the robot. Because of this every particle needs to maintain and update its own map [36]. The method described above fails to produce a usable map of the surroundings. Just as with normal grid maps, the motion of the robot is very noisy. This can be resolved by using more samples but memory consumption will increase as each particle needs to maintain its own map. A grid map is quite large, therefore it is difficult to use more samples [36]. Another solution to this problem is to improve the pose estimate before the particle filter is applied. As discussed previously, the pose estimation can be improved by using scan matching. Using scan matching will result in a usable map [36].

Grid-Based FastSLAM 2.0

Just as with feature based solutions, it can be useful within grid-based FastSLAM to take in consideration the measurements during sampling which is the key theory behind FastSLAM 2.0. This will result in a more peaked proposal distribution and less particles compared to FastSLAM 1.0.

2.3 SLAM Solutions

As many different approaches to the SLAM problem exist, many different solutions and implementations are created. When reading Section 2.2 it becomes clear that feature-based SLAM provides the most accurate solution to the SLAM problem. Obviously feature-based SLAM is based upon the presence of odometry. This research focusses on finding a high quality solution when odometry is abscent. Within this section, several SLAM solutions are presented which provide a working SLAM solution based on the grid-SLAM approach discussed in Section 2.2.3.

2.3.1 Cartographer

The Cartographer implementation is created by Google to provide a working indoor SLAM solution by using a LiDAR attached to a backpack. This way, a person can walk around a building and generate a map of this building at the same time. Scan-matching is used to minimise errors within the generated map. By using this approach, the Cartographer solution is not dependent on odometry data (although odometry data can be used to improve the result of the solution). Google added an Inertial Measurement Unit (IMU) to its backpack to improve the data generated by the LiDAR. Using the IMU module is not mandatory and can be ignored when implementing the algorithm. The Cartographer solution for 2D SLAM is explained within [38]. Based upon [38] and the walkthrough published on the official Cartographer website [39], a basic introduction to the algorithm will be provided.

Cartographer describes their solution as the cooperation of two components: *local SLAM* and *global SLAM*. These components consume input sensor data to generate their output. A high level overview of the Cartographers system is provided by Figure 2.8.

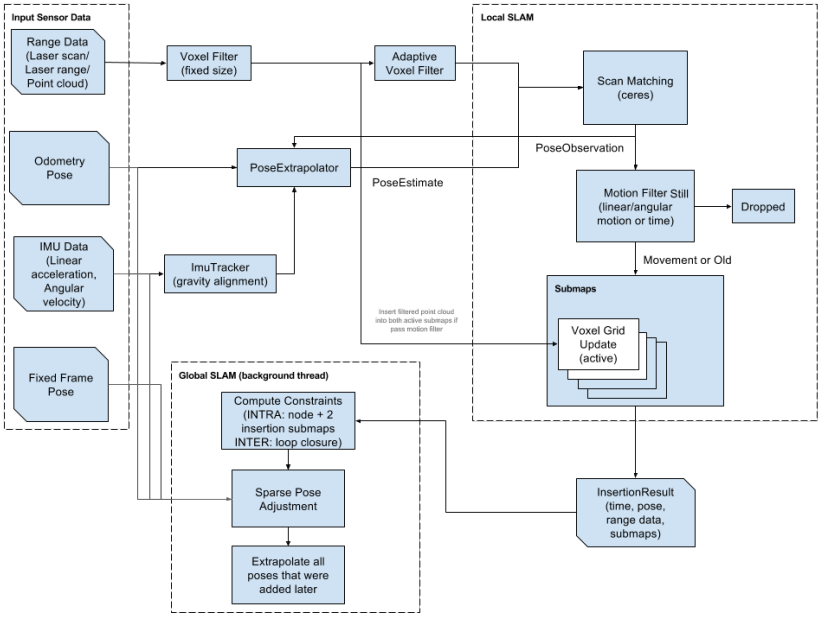


Figure 2.8: High level overview of Cartographers system [39]

Local SLAM

The goal of Cartographer's local SLAM component is to create a succession of submaps. These are to be locally consistent but it is accepted that local SLAM drifts over time. Each consecutive observation of our LiDAR, referred to as a scan, is matched against a small chunk of the world which is called a submap.

By using scan matching, the cartographer solution inserts a new scan into its current submap. The scan matchers responsibility maximise the probability of the scans within the submap. The Cartographers scan matcher is based upon the Ceres Solver [1] which is a library developed by Google for solving non-linear least squares problems with bounds constraints and general unconstrained optimisation problems. This scan matcher solution will not be discussed any further within this thesis. The submaps take the form of probability grids. Whenever a scan is to be inserted into the probability grid, the algorithm decides upon a set of grid points to be occupied and a set of grid points which are free. For every scan, the closest grid point is added to the occupied grid point set, for every grid point which intersects the ray of our LiDAR between origin and scan is added to the free grid point set. This way the Cartographer knows which cells are occupied, free and unknown.

A submap is considered as complete when the local SLAM has received a given amount of range data. Local SLAM drifts over time, global SLAM is used to fix this drift. Submaps must be small enough so that the drift inside them is below the resolution, and so that they are locally correct. On the other hand, they should be large enough to be distinct for loop closure to work properly [39].

Global SLAM

While local SLAM generates a succession of submaps, global SLAM tries to build a general map by rearranging submaps between each other. The global SLAM is a kind of "GraphSLAM", it is essentially a pose graph optimisation which works by building constraints between nodes and submaps and then optimising the resulting constraints graph [39]. When a node and a submap are considered for constraint building, they go through a first scan matcher. This scan matcher, based on a "branch and bound" mechanism, has been specifically designed for Cartographer and makes real time loop closure scan matching possible [39]. Once the branch-and-bound-based scan matcher has a good enough proposal (above a minimum score of matching), it is then fed into a Ceres Scan Matcher to refine the pose [39].

[1] ttp:// ceres-solver.org.

2.3.2 Hector SLAM

Hector SLAM is created by 'Team Hector'. With this implementation, they try to provide a flexible and scalable solution to the SLAM problem. Hector SLAM consumes low computational resources meaning it can be used on modest hardware. The approach uses ROS (see Section 3.1.2) as middleware and is available as open source software. The theoretical details of this solution, can be retrieved within [40]. Within this section, a small summary is given.

[40] states that a fair amount of SLAM solutions already exist. Most of these solutions rely on accurate odometry data and do not leverage the accuracy and high update rate of modern LiDAR systems. Many systems make a distinction between 'front-end' and 'back-end' SLAM. Front-end SLAM is used to estimate the movement of the robot online in real time, back-end SLAM is used to perform optimisations between poses generated by the front-end. It can be stated that the Google Cartographer, explained within Section 2.3.1, is a system which works in a similar structure. [40] explains that their solution serves as a SLAM front-end and does not provide any back-end optimisation. It is the believe of Hector SLAM that in real world conditions, optimisations from the front-end generated map are not needed.

To be able to represent arbitrary maps, a grid map is used. The discrete nature of occupancy grid maps limits the precision that can be achieved and also does not allow the direct computation of interpolated values or derivatives [40]. Section 2.2.3 shows that when using grid maps, the usage of scan matching is needed to generate optimised maps as an output. The approach described in [40] is based on optimisation of the alignment of scan observations with the map learnt so far. Using this implementation, there is no need for any further data association. Another advantage is that this way the latest reading is implicitly mapped with all previous scans.

Hector SLAM is able to deal with 6 degrees of freedom (6DOF). To improve the interpretation of LiDAR scan readings, navigational subsystems such as an IMU or GPS can be used. For estimating the 6D pose of the platform, Hector SLAM uses an Extended Kalman Filter (EKF). As these devices will not be used within the research of this thesis, this will not be discussed any further within this section.

3 Solution

When searching for SLAM solutions, one can find several implementations to this problem. Many of these solutions exist out of the usage of odometry data of the motion sensors and scan results of distance sensors. As odometry data might not always be available, the goal of this thesis is to compare and benchmark different SLAM solutions which are exclusively based on distance sensor readings.

The first step of testing SLAM solutions is to actually make a working implementation within a certain environment. Section 3.1 presents the technical environment which is set up and presents the technical set-up of the selected SLAM solutions.

To make sure that none of the selected solutions is having an advantage based upon the distance sensor readings, it is opted to generate new data. Within Section 3.2 a presentation is done of the selected LiDAR sensor and the environment being scanned by the LiDAR.

3.1 Technical Set-Up

3.1.1 Ubuntu 18.04

Ubuntu is an operating system which is composed out of free and open source software. It is a popular Linux distribution which exists in three flavours:

desktop, server and core. For convenience, Ubuntu 18.04 is installed as a virtual machine on a Windows 10 host using Oracle VM Virtualbox. This version of the operating software was released on 26 April 2018 and will be supported until April 2023. Ubuntu 18.04 is not the latest version available at the moment of writing, the choosing of this installation is based upon the documentation of the selected LiDAR. The YDLIDAR X4 installation guide describes the installation procedure for Ubuntu 18.04/16.04/14.04 LTS and Windows 7/10. As Ubuntu 18.04 is the most recent Ubuntu version which is described within the documentation, it became the selected installation.

3.1.2 ROS Melodic Morenia

Robotic Operating Software (ROS) is a so-called robotics middleware suite. It can be described as a flexible framework for writing robot software by providing a collection of tools, libraries and conventions for simplifying the creation of complex robot behaviour. ROS uses a graph structure to represent processes as nodes. These nodes are connected by edges which are called topics. The topics make it possible for nodes to communicate and pass data to each other. This principle is shown by Figure 3.1. Projecting this structure to the solution of this thesis, it means that the LiDAR will be a node and the selected SLAM package will be another node. Passing the data from our LiDAR to the SLAM package will be done by setting up a topic between both nodes. By using this structure, ROS makes it possible to interchange LiDAR models and SLAM packages in a very flexible manner.

For this thesis, the ROS 'Melodic Morenia' version was selected as it is, according to the official ROS website, the version supported for Ubuntu 18.04. More information can be retrieved at the official ROS website [1].

3.1.3 SLAM Packages

Hector SLAM

Hector SLAM is provided as a ROS package. Therefore it can easily be installed by launching the following one-liner within the terminal:

```
sudo apt-get install ros-melodic-hector-slam
```

Bear in mind that 'melodic' is used as this is the chosen ROS distribution. If any other distro is used, melodic has to be changed by the corresponding term.

[1] https://www.ros.org/

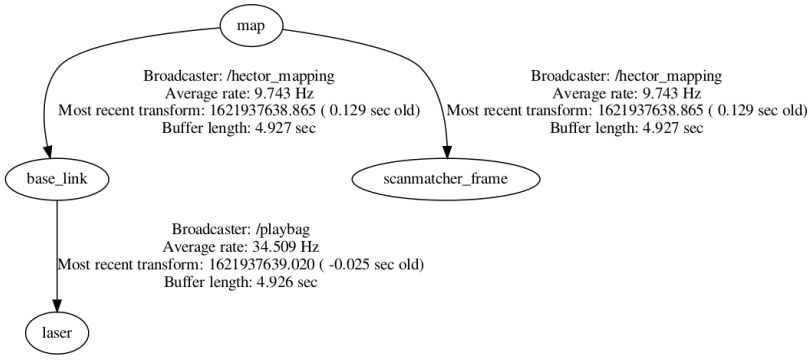


Figure 3.1: Illustration of ROS by topics connected nodes

A launch-file is created within the Hector-package folder *hector_slam_launch/launch*. The created launch-file is an adapted version of the file '*tutorial.launch*' which is present in the same folder after installing the Hector SLAM package. The details of the *hector_slam_launch/launch* can be retrieved within the appendices Section A.1. Two lines from this file need to be highlighted.

```
<param name="/use_sim_time" value="true"/>
```

By setting the value of 'use_sim_time' to 'true', it becomes possible to work with non-live data. It is the goal to replay formerly made recordings of LiDAR observations to make sure the exact same data is used between all executions of the SLAM solutions.

```
<include file="$(find hector_mapping)/launch/my_mapping.launch"
   />
```

The mapping configurations can be found within the specified file.

Within the folder *hector_mapping/launch*, the file 'my_mapping.launch' is created. Its details can be retrieved within the appendices Section A.2.This file is based on the file 'mapping_default.launch' which is present after installation of the Hector SLAM package. Two main adaptations are executed within the newly created file.

```
<arg name="base_frame" default="base_link"/>
<arg name="odom_frame" default="base_link"/>
```

When one does not want to make use of the odometry capabilities of the Hector SLAM solution, the parameters 'base_frame' and 'odom_frame' need to be set to 'base_link'. This method is described within the package documentation.

```
<node pkg="tf" type="static_transform_publisher" name="
    base_link_laser_broadcaster" args="0 0 0 0 0 0 /base_link /
    laser 100"/>
```

Hector SLAM obviously needs LiDAR readings as an input. Previous statement creates the listener by which the Hector SLAM node 'base_link' listens to the data broadcast on the node 'laser'. As mentioned in Section 3.2.1, this is the name of the node which broadcasts the LiDAR observational data.

To launch the actual Hector SLAM package, the one-liner below needs to be launched from the terminal. Keep in mind that this will only work when the LiDAR data is being broadcasted at the same time.

```
roslaunch hector_slam_launch my_launch.launch
```

Google Cartographer

On the official Google Cartographer website, a guide is published on how to install their solution. The steps followed from this guide are published within Section A.3 of the appendices.

The cartographer solution uses two main files. One launch-file in which all nodes to be started are programmed. The other file holds a 'lua' extension and contains all configuration settings for the Cartographers algorithm.

The launch-file used within this implementation is named 'my_carto.launch' and is put inside of the folder '*cartographer_ros/launch*'. Its details can be retrieved in Section A.4 of the appendices. The file is based upon the Cartographers provided file 'backpack_2D.launch'. The most important change within the new launch-file is the call towards the custom created lua-file:

```
<node name="cartographer_node" pkg="cartographer_ros" type="
    cartographer_node" args="-configuration_directory $(find
    cartographer_ros)/configuration_files -
    configuration_basename my_carto.lua" output="screen" />
```

The newly created lua-file within this solution is called 'my_carto.lua'. The content of this file can be found within the appendices at Section A.5 and is mainly based upon the file 'backpack_2d.lua' which comes with the package. Many configuration options can be used within the Google Cartographer. When the exact specification of the moving entity would be known, one is able to create an optimal file. Within this solution the preference went to a more global configuration as different kinds of observations will be performed. The most important changes/additions to the newly created lua-file are the following:

```
...

options = {
  ...
  provide_odom_frame = false,
  ...
}

...

TRAJECTORY_BUILDER_2D.min_range = 0.12
TRAJECTORY_BUILDER_2D.max_range = 10
TRAJECTORY_BUILDER_2D.missing_data_ray_length = 10

TRAJECTORY_BUILDER_2D.num_accumulated_range_data = 1

TRAJECTORY_BUILDER_2D.use_imu_data = false

TRAJECTORY_BUILDER_2D.use_online_correlative_scan_matching =
    true
...
```

These lines tell Google Cartographer that no odometry will be used to solve the SLAM problem. The minimal and maximal scanning distance of the LiDAR is provided and a default distance in case a measurement is out of these boundaries. No IMU will be used to provide any extra rotational data. According to Cartographer's documentation, the RealTimeCorrelativeScanMatcher should be enabled if no other sensors are used or if you do not trust them. By doing this, scans are matched against their submap. the best prior is then used for the scan matching algorithm.

To run the Google Cartographer the one-liner below needs to be launched from the terminal. Keep in mind that this will only work when the LiDAR data is being broadcasted at the same time.

```
roslaunch cartographer_ros my_carto_bag.launch
```

3.2 Observations

3.2.1 LiDAR

The obtained observations are measured by the use of a LiDAR. The LiDAR used within this thesis is a YDLIDAR X4. This specific model is a relatively affordable device which uses triangulation to measure the distance between the device itself and the object reflecting the laser. More information about triangulation can be found at Section 2.1.2. The YDLIDAR X4 is able to rotate 360 degrees to scan and range its surrounding environment. It is able to scan objects at a distance of minimally 0.12m up to maximally 10m. Table 3.1 shows all measurement characteristics of the LiDAR. The manufacturer provides a GIT with a full SDK, example-code, drivers for several goals (e.g. ROS, Arduino, Apollo) [2].

Table 3.1: YDLIDAR X4 Measurement Characteristics

Subject	Performance
Sample Frequencey	$5000Hz$
Scanning Frequency	$6-12Hz$
Range	$0.12-> 10m$
Angular Range	$0-360$ Degrees
Life Span	$1500h$

Installing the ROS-package of the selected LiDAR is fairly easy as the manufacturer exposes the package as a GIT which can be cloned. The detailed installation instruction followed, can be retrieved within the appendices at Section A.6.

To generate actual LiDAR readings, a launch-file need to be created within the folder 'ydlidar_ros launch'. Several example files are already present after installing the YDLIDAR ROS-driver. As the used device is a YDLIDAR model X4, the file created is based upon the file 'X4.launch'. The detailed file can be retrieved within the apendices at Section A.7. The most important change to mention compared to the original file is the line in which the frame-id of the broadcasted node is 'laser'. As this is the name used within the listener for laser scan input of the installed SLAM solutions.

```
<param name="frame_id" type="string" value="laser"/>
```

[2] https://github.com/YDLIDAR

To initiate the LiDAR a one-liner within the terminal is used. This can be found below.

```
roslaunch ydlidar_ros_driver [launch file]
```

3.2.2 BAG Files

To be able to make a fair comparison between the results generated by different packages, it is optimal to provide each algorithm with the exact same data. ROS allows users of the operating system to create 'bags' by using a tool like 'rosbag'. These tools subscribe to one or more ROS topics and store the serialised message data in a file. These bag files can then be played back to the same (or other) ROS topics. By using this approach it is possible to record laser scan data of an environment and use the exact same data as input for each SLAM package. This is initiated by executing following line within the terminal:

```
rosbag record rosout tf scan
```

One needs to make sure the LiDAR is making observations as described in Section 3.2.1 at the same moment than the rosbag is being created. This will generate a bag-file as an output which can be played at any given time.

The one-liner below is used to play the bag-file so that the data can be picked up by the SLAM solutions.

```
rosbag play --clock [bag-filename]
```

3.2.3 Environment

Though several suppliers of SLAM implementations provide example data to work with, for this research it is opted to generate new readings. As it is the goal to judge the performance of the SLAM solutions within real life situations, the first floor of a house is chosen. Each of the rooms within this floor is provided with a name: 'room 1', 'room 2', 'room 3' and 'corridor'. Their location within the building can be found on Figure 3.2. Different obstacles are being expected to have an influence on the LiDAR observations. Each of the rooms is equipped with a window. As lasers travel through glass, it is foreseen to observe distance measurements which lay behind the location of the windows. Figure 3.2 clearly indicates the locations of the windows. The same theory can be applied to the mirror which is present in 'room 2'. Both within 'room 2' and 'room 3' a closet is present. Both closets are indicated by black rectangles within Figure 3.2. The same figure shows the location of the mirror by a black triangle.

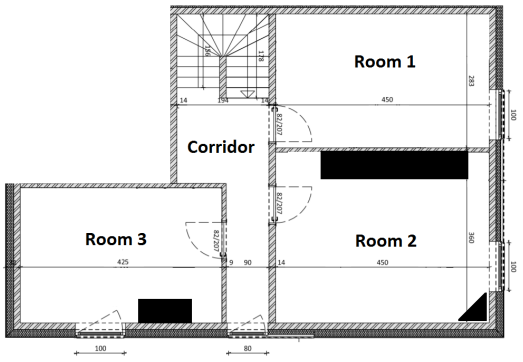


Figure 3.2: Architectural floor-plan of the scanned environment with named rooms and obstacle indications.

3.2.4 Observation Sets

Defining the performance and characteristics of the selected SLAM solutions, requires different types of tests. Three different LiDAR observation datasets are created in which the laser scanner moves in a specific scenario. The three sets are named 'observation 1', 'observation 2' and 'observation 3'. Whenever this document mentions one of these names, it refers to a specific group of observational data gathered in a manner as described further in this section. More information about the environment is provided within Section 3.2.3.

- *Observation 1*: This observation visits every room of our selected building floor only once. The LiDAR observations start from the corridor and move to 'room 1'. From there it turns around slowly to return to the corridor and move to 'room 2'. Once again the LiDAR will turn around slowly to move to 'room 3' by passing through the corridor. When 'room 3' is scanned, the observations are stopped.

- *Observation 2*: The LiDAR will move exactly the same in 'observation 2' as it does in 'observation 1' except for the rooms being visited a second time so more distance measurements are made.

- *Observation 3*: Just like 'observation 2', this observation will move twice through all the rooms of the building floor. However, the LiDAR will move in a faster pace then during 'observation 2' and one swift rotation of the LiDAR around its vertical axis is performed in each of the rooms.

4
SLAM Results

Each of the SLAM packages discussed within Section 3.1.3 has successfully been implemented within the technical set-up discussed within Section 3.1. The implementations of the SLAM packages are optimised according to the package documentations to generate the best possible map by the exclusive use of LiDAR observational data. Different sets of observational data have been made, see Section 3.2.4, to benchmark the behaviour of the SLAM solutions. All of these readings were in the same environment but have their own characteristics. Within this chapter a proposal for evaluating the SLAM results of each solution will be presented. Based upon this proposal, the performance of each SLAM implementation within each observational dataset will be discussed.

4.1 SLAM Evaluation Technique

Evaluating SLAM techniques is unfortunately not straight forward. Different techniques and visions exist on how to benchmark SLAM solutions.

One of the most straight forward metrics is a visual evaluation of the generated map. This is a metric which is very sensitive to subjective opinions. To perform an objective visual evaluation of the floorplan, it is opted to limit this evaluation to defining and indicating the (in)correct presence or absence of walls and obstacles.

[41] describes an evaluation technique in which the generated maps are compared to the ground truth. The ground truth is defined by a map of the surrounding which is a perfect representation of the reality. An architectural map of the selected environment exists but this is however not an exact representation of the reality. Construction workers are allowed to deviate within certain boundaries from the plans designed by the architect. The selected mapping area posses a set of obstacles which are not indicated on the architects floorplan (e.g. closets, mirror) which is another reason why the architectural floorplan is not suitable. However, the idea of comparing the generated maps against the ground truth is an interesting approach and will be followed within this thesis. Manually measuring and creating a complete map which represents the reality is error prone as well. Therefore it is opted within this research to not manually create a complete floorplan but measure certain areas within the map. It is then possible to make an aspect ratio between several walls and see how these relate to each other. By then applying the same technique to the maps generated by the SLAM implementations, it is possible to compare the correctness of the maps.

SLAM algorithms involve many mathematical calculations. Obviously these calculations require CPU power and memory of the hardware on which the SLAM solutions are implemented. As the CPU and memory is very divergent and depending on how the available hardware is set up, the usage of these is an important metric. It is assumed that when the algorithm does not have sufficient capacity to perform the mathematical calculations needed for its algorithm, the map generation will lag behind and will be sensitive for errors. Memory and CPU usage will be monitored during the runtime of the different SLAM solutions.

4.2 Visual Evaluation

Within this section, a visual evaluation is made of the maps generated by the implemented SLAM solutions. All generated sets of LiDAR data are used to produce their corresponding map using the selected SLAM solutions. Each solution will run multiple times on the same dataset to make sure that the created maps are consistent. As a visual inspection can easily become subjective, the following metrics are chosen to be evaluated within the generated map:

- Correct or incorrect presence of walls and obstacles
- Shape of walls and obstacles
- Consistency between the maps generated by the same dataset

Figure 4.1 represents an architectural floorplan by which the building floor is constructed. This map is not an exact representation of the reality as obstacles such as closets, mirrors and doors are present within the scanned rooms. The map will however provide a decent foundation for evaluating the mapping results. Any deviations which were expected between the architectural plan and the maps created by the SLAM solutions, will be explained.

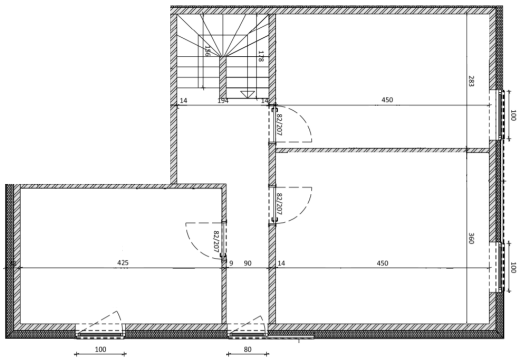
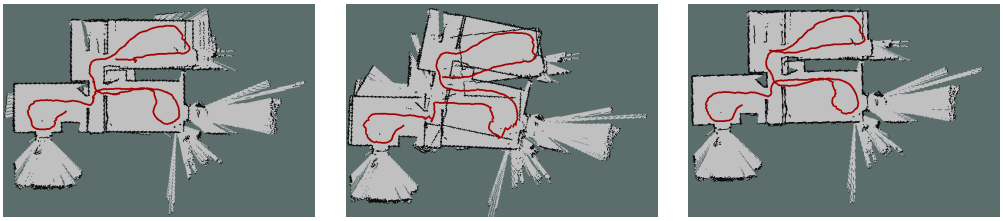


Figure 4.1: Architectural floorplan of the scanned environment

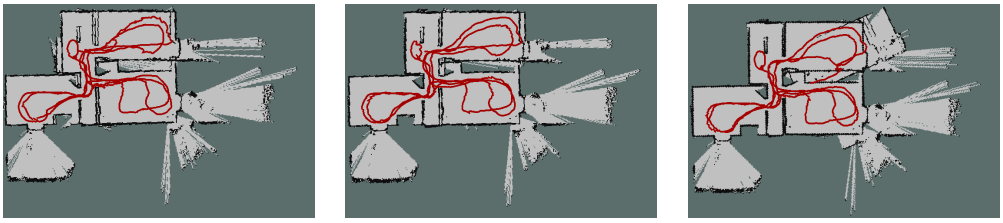
4.2.1 Hector SLAM



(a) Mapping result 1 (b) Mapping result 2 (c) Mapping result 3

Figure 4.2: Mapping results of running 'observation 1' with Hector SLAM

Figure 4.2, Figure 4.3 and Figure 4.4 show a clear inconsistency between all three maps generated of one and the same dataset. When observing



(a) Mapping result 1 (b) Mapping result 2 (c) Mapping result 3

Figure 4.3: Mapping results of running 'observation 2' with Hector SLAM

(a) Mapping result 1 (b) Mapping result 2 (c) Mapping result 3

Figure 4.4: Mapping results of running 'observation 3' with Hector SLAM

the process of generating a map using Hector SLAM, it becomes clear that the mapping of the 'room 1' is a cause for errors and inconsistencies. As a consequence the 'double wall' is created as can be seen in Figures 4.2 (a), (b) and Figures 4.3 (a), (c). Another example is the 'observed' closet space. No data is retrieved within the closet but when inspecting all three maps of Figure 4.3, one can notice that some areas of the closet are observed as empty. This is the consequence of 'room 1' being put at a different angle than the rest of the map. The map eventually recovered from this error but the closet space shows as if it has been observed while it wasn't. When looking at Section 2.3.2, Hector SLAM tries to relate the observations of the LiDAR against the map created so far. As in the beginning of the map generation this map does not exist, the solution cannot relate the data points. As the map grows bigger, the better the mapping algorithm of the Hector SLAM solutions performs. As more data points are generated and the bigger the map becomes, the smaller the differences between the generated maps becomes as well. This can clearly be observed when comparing the maps generated in Figure 4.2 and Figure 4.3.

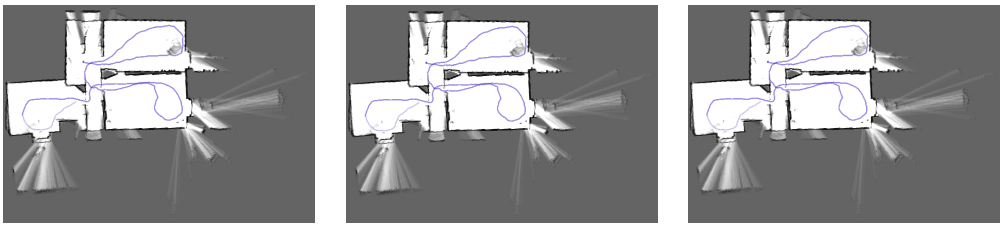
Figure 4.4 shows three unusable maps which do not remotely resemble the architectural floorplan. Hector SLAM shows to have difficulties when the LiDAR makes swift rotations around its vertical axis. If this situation would only occur once, then the map would be able to recuperate from this problem when more data points become available. Whenever the algorithm is faced with a LiDAR which is often rotated around its vertical axis, the generated map will be unusable.

Figure 4.2 (b) shows a map in which the ground truth can still be retrieved. However this map is not usable for humans nor for robots as walls are overlapping or at a wrong axis. Figure 4.3 shows that by generating more data points and revisiting the same locations multiple times, the map improves. Though one needs to take in account that Hector SLAM is not always a reliable solution when using the algorithm in spaces which are only visited once.

From these observations, several conclusions can be made about Hector SLAM:

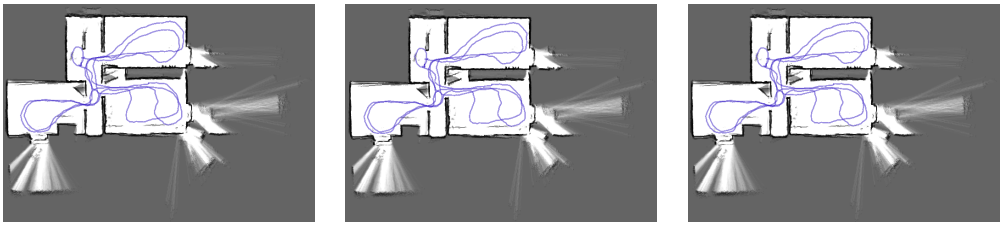
- Generating maps from LiDAR data which rotates often quickly around its vertical axis might generate unusable maps
- The mapping result can deviate even when using the same observational data
- Observed spaces might be a result of improving the map with new observational data instead of actual observations
- The map improves by using more data points and revisiting locations which were already visited

4.2.2 Cartographer



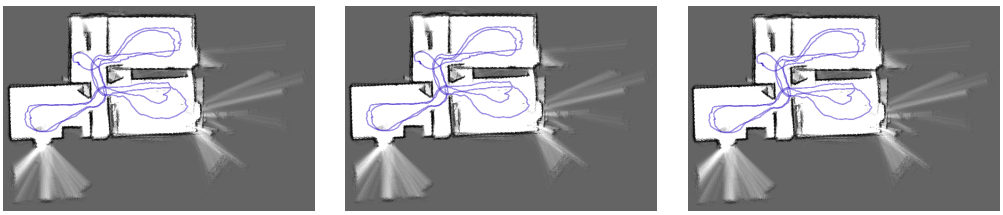
(a) Mapping result 1 (b) Mapping result 2 (c) Mapping result 3

Figure 4.5: Mapping results of running observation 1 with Google Cartographer



(a) Mapping result 1 (b) Mapping result 2 (c) Mapping result 3

Figure 4.6: Mapping results of running observation 2 with Google Cartographer



(a) Mapping result 1 (b) Mapping result 2 (c) Mapping result 3

Figure 4.7: Mapping results of running observation 3 with Google Cartographer

An observation of Figures 4.5, 4.6 and 4.7 learns that the Cartographer algorithm performs an identical output every run of a particular dataset. This shows the reliability of the Cartographer algorithm.

Figure 4.5 shows several errors such as the closet space of 'room 2' which is too small, the corner of 'room 3' and the corridor overlap, and several data point observations are placed outside of the map on locations where it does not make sense. However it must be said that the map based on 'observation 1' outputs already a usable map for humans and robots. When comparing Figure 4.5 and 4.6, it becomes clear that the Cartographer map improves when more data points become available and rooms are visited more frequently. As a result the map corrects the errors which were made when the rooms were only visited once. The closet space which was too narrow in Figure 4.5 is clearly more realistic in Figure 4.6. Another example is the overlap of 'room 3' with the corridor which is more correctly aligned in Figure 4.6.

The robustness of the Cartographers algorithm becomes the most clear when observing Figure 4.7. Even when swiftly rotating the LiDAR around its vertical axis, this SLAM solution is capable of generating a usable output.

When looking at Section 2.3.1, one can learn the two components of the algorithm. The local SLAM algorithm creates smaller submaps based upon a range of LiDAR observational data. These submaps are then matched and rearranged by the global SLAM component to build one global map of the environment. When rotating the LiDAR, the Cartographer algorithm is therefore able to recover from this sudden movement as long as the submaps generated can be matched against the global map.

From these observations, several conclusions can be made about Google Cartographer:

- A robust solution is provided which can handle unexpected events of the LiDAR and sudden movements
- The solution is consequent in a way that it generating consistent maps of the same data
- The map improves by using more data points and revisiting locations which were already visited

4.3 Ground Truth Comparison

When judging the correctness of a map generated by a SLAM solution, it is important to evaluate the distance measurements against the reality. Ideally a map exists which is a perfect representation of the environment being mapped. This map can then be used within programmatic software to compare the created maps against this ground truth. Unfortunately, such a map does not exist for our scanned building floor. Creating such a floorplan manually is not only labour-intensive, it is prone to errors as well. To minimise the errors and the manual labour, a particular set of distance measurements is made by using a ruler and selecting several locations to be measured.

Figure 4.8 represents the locations of all distances which are measured within our actual building floor. Each distance location is provided with a number. An overview of these numbers against their measured distance is provided within Table 4.1. The same locations which are manually measured, will be measured within the maps generated by the implemented SLAM solutions. The maps are visualised within 'RVIZ' which is an out-of-the-box ROS package. This program is equipped with a measurement tool. The distance measures within the generated maps will be performed using this tool.

One may argue that comparing the actual measurements against each other might be insufficient to be able to make any conclusions. Deviations

Table 4.1: Distance measures of the reality using a ruler. Numbers can be related to the locations indicated at Figure 4.8.

number	distance (cm)
1	447
2	656
3	192
4	104.5
5	308
6	424.5
7	552.5
8	448.5
9	522.5
10	102.5
11	283.5
12	57
13	295
14	98.5
15	278.5
16	357
17	61.5
18	297
19	276

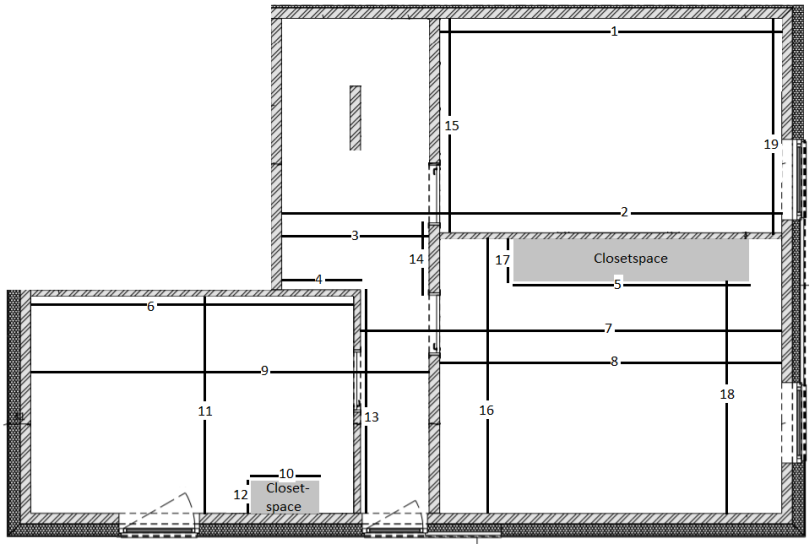


Figure 4.8: Floorplan indicating the manually measured distances

based upon these numbers can be caused by an inaccurate LiDAR or poor measurement skills of the person performing the manual measurements of the building floor. To reduce the effect of these problems, an evaluation of the aspect ratio between the measured distances will be made. Table 4.2 shows the aspect ratios of the measured distances of the manual measurements. By comparing the aspect ratios of the manual measurement against the aspect ratios of the SLAM solution distance measurements, conclusions about the generated map can be made. If the difference between aspect ratios of the SLAM solutions and the reality are very small, possible differences between the distance measurements of the reality and the SLAM solution maps could be caused by a wrong calibration of the LiDAR.

4.3.1 Hector SLAM

Table 4.3 represents the measured distances within the maps created by the Hector SLAM solution. As the map for 'observation 3' was unusable, no measurements have been taken from this map.

When examining Table 4.4 it becomes clear that most of the measured distances are shorter than the distances of the manual measurements. 'Observation 1' shows an average deviation of -6.69, while 'observation 2' shows an average difference of -5.18.

Tables 4.7 and 4.8 provide an overview of the differences between the aspect ratios of the manual distance measurements of the reality and the

Table 4.2: Aspect ratio between the measured distances from Table 4.1

number	2	3	4	5	6	7	8	9	10
1	0.68	2.33	4.28	1.45	1.05	0.81	1.00	0.86	4.36
2	x	3.42	6.28	2.13	1.55	1.19	1.46	1.26	6.40
3	x	x	1.84	0.62	0.45	0.35	0.43	0.37	1.87
4	x	x	x	0.34	0.25	0.19	0.23	0.20	1.02
5	x	x	x	x	0.73	0.56	0.69	0.59	3.00
6	x	x	x	x	x	0.77	0.95	0.81	4.14
7	x	x	x	x	x	x	1.23	1.06	5.39
8	x	x	x	x	x	x	x	0.86	4.38
9	x	x	x	x	x	x	x	x	5.10

number	11	12	13	14	15	16	17	18	19
1	1.58	7.84	1.52	4.54	1.61	1.25	7.27	1.51	1.62
2	2.31	11.51	2.22	6.66	2.36	1.84	10.67	2.21	2.38
3	0.68	3.37	0.65	1.95	0.69	0.54	3.12	0.65	0.70
4	0.37	1.83	0.35	1.06	0.38	0.29	1.70	0.35	0.38
5	1.09	5.40	1.04	3.13	1.11	0.86	5.01	1.04	1.12
6	1.50	7.45	1.44	4.31	1.52	1.19	6.90	1.43	1.54
7	1.95	9.69	1.87	5.61	1.98	1.55	8.98	1.86	2.00
8	1.58	7.87	1.52	4.55	1.61	1.26	7.29	1.51	1.63
9	1.84	9.17	1.77	5.30	1.88	1.46	8.50	1.76	1.89
10	0.36	1.80	0.35	1.04	0.37	0.29	1.67	0.35	0.37
11	x	4.97	0.96	2.88	1.02	0.79	4.61	0.95	1.03
12	x	x	0.19	0.58	0.20	0.16	0.93	0.19	0.21
13	x	x	x	2.99	1.06	0.83	4.80	0.99	1.07
14	x	x	x	x	0.35	0.28	1.60	0.33	0.36
15	x	x	x	x	x	0.78	4.53	0.94	1.01
16	x	x	x	x	x	x	5.80	1.20	1.29
17	x	x	x	x	x	x	x	0.21	0.22
18	x	x	x	x	x	x	x	x	1.08

distance measurements of the maps generated by Hector SLAM. Important to know is that map (c) from Figure 4.2 was used for the measurements of Table 4.7. The average aspect ratio deviation for Table 4.7 is 0.0118 while the average for Table 4.8 is 0.0307. Both of these deviations can be considered very low meaning the aspect ratio of the map is very similar to the reality. The most and biggest differences can be found in measurements number 12 and 17. These are both the depth measurement of the closet space. This might be an indication for different problems. The SLAM solution might have problems with these obstacles, or the manual measurements around this area might have been performed badly, or the LiDAR observations were poor. As the average is heavily influenced by the measurements of distance 12 and 17, one may want to look at the medians for 'observation 1' and 'observation 2'. These give us respectively the results 0.0003 and 0.0092. These values show once again that the aspect ratios of the generated by map by the algorithm is very close to the reality.

Table 4.3: Distance measures of Hector SLAM generated maps. Numbers can be related to the locations indicated at Figure 4.8.

number	observation 1 (cm)	observation 2 (cm)
1	447	442.8
2	640.6	652.3
3	186.8	183.5
4	96.8	100.0
5	300.3	306.9
6	417.4	415.6
7	543.9	547.2
8	438.8	440.6
9	514.2	512.6
10	103.5	105.2
11	277.1	277.2
12	60.1	53.6
13	285.4	292.0
14	100.1	100.2
15	258.7	272.1
16	349.3	350.6
17	55.3	58.5
18	287.3	287.2
19	272.2	265.4

Table 4.4: Deviation of Hector SLAM measurements Table 4.3 against reality measurements Table 4.1

number	observation 1 (cm)	observation 2 (cm)
1	0.00	−4.2
2	−15.40	−3.7
3	−5.20	−8.5
4	−7.70	−4.5
5	−7.70	−1.1
6	−7.10	−8.9
7	−8.60	−5.3
8	−9.70	−7.9
9	−8.30	−9.9
10	1.00	2.7
11	−6.40	−6.3
12	3.10	−3.4
13	−9.60	−3
14	1.60	1.7
15	−19.80	−6.4
16	−7.70	−6.4
17	−6.20	−3
18	−9.70	−9.8
19	−3.80	−10.6

Table 4.5: Aspect ratio between the measured distances from Table 4.3 'observation 1'

number	2	3	4	5	6	7	8	9	10
1	0.70	2.39	4.62	1.49	1.07	0.82	1.02	0.87	4.32
2	x	3.43	6.62	2.13	1.53	1.18	1.46	1.25	6.19
3	x	x	1.93	0.62	0.45	0.34	0.43	0.36	1.80
4	x	x	x	0.32	0.23	0.18	0.22	0.19	0.94
5	x	x	x	x	0.72	0.55	0.68	0.58	2.90
6	x	x	x	x	x	0.77	0.95	0.81	4.03
7	x	x	x	x	x	x	1.24	1.06	5.26
8	x	x	x	x	x	x	x	0.85	4.24
9	x	x	x	x	x	x	x	x	4.97

number	11	12	13	14	15	16	17	18	19
1	1.61	7.44	1.57	4.47	1.73	1.28	8.08	1.56	1.64
2	2.31	10.66	2.24	6.40	2.48	1.83	11.58	2.23	2.35
3	0.67	3.11	0.65	1.87	0.72	0.53	3.38	0.65	0.69
4	0.35	1.61	0.34	0.97	0.37	0.28	1.75	0.34	0.36
5	1.08	5.00	1.05	3.00	1.16	0.86	5.43	1.05	1.10
6	1.51	6.95	1.46	4.17	1.61	1.19	7.55	1.45	1.53
7	1.96	9.05	1.91	5.43	2.10	1.56	9.84	1.89	2.00
8	1.58	7.30	1.54	4.38	1.70	1.26	7.93	1.53	1.61
9	1.86	8.56	1.80	5.14	1.99	1.47	9.30	1.79	1.89
10	0.37	1.72	0.36	1.03	0.40	0.30	1.87	0.36	0.38
11	x	4.61	0.97	2.77	1.07	0.79	5.01	0.96	1.02
12	x	x	0.21	0.60	0.23	0.17	1.09	0.21	0.22
13	x	x	x	2.85	1.10	0.82	5.16	0.99	1.05
14	x	x	x	x	0.39	0.29	1.81	0.35	0.37
15	x	x	x	x	x	0.74	4.68	0.90	0.95
16	x	x	x	x	x	x	6.32	1.22	1.28
17	x	x	x	x	x	x	x	0.19	0.20
18	x	x	x	x	x	x	x	x	1.06

Table 4.6: Aspect ratio between the measured distances from Table 4.3 'observation 2'

number	2	3	4	5	6	7	8	9	10
1	0.68	2.41	4.43	1.44	1.07	0.81	1.00	0.86	4.21
2	x	3.55	6.52	2.13	1.57	1.19	1.48	1.27	6.20
3	x	x	1.84	0.60	0.44	0.34	0.42	0.36	1.74
4	x	x	x	0.33	0.24	0.18	0.23	0.20	0.95
5	x	x	x	x	0.74	0.56	0.70	0.60	2.92
6	x	x	x	x	x	0.76	0.94	0.81	3.95
7	x	x	x	x	x	x	1.24	1.07	5.20
8	x	x	x	x	x	x	x	0.86	4.19
9	x	x	x	x	x	x	x	x	4.87

number	11	12	13	14	15	16	17	18	19
1	1.60	8.26	1.52	4.42	1.63	1.26	7.57	1.54	1.67
2	2.35	12.17	2.23	6.51	2.40	1.86	11.15	2.27	2.46
3	0.66	3.42	0.63	1.83	0.67	0.52	3.14	0.64	0.69
4	0.36	1.87	0.34	1.00	0.37	0.29	1.71	0.35	0.38
5	1.11	5.73	1.05	3.06	1.13	0.88	5.25	1.07	1.16
6	1.50	7.75	1.42	4.15	1.53	1.19	7.10	1.45	1.57
7	1.97	10.21	1.87	5.46	2.01	1.56	9.35	1.91	2.06
8	1.59	8.22	1.51	4.40	1.62	1.26	7.53	1.53	1.66
9	1.85	9.56	1.76	5.12	1.88	1.46	8.76	1.78	1.93
10	0.38	1.96	0.36	1.05	0.39	0.30	1.80	0.37	0.40
11	x	5.17	0.95	2.77	1.02	0.79	4.74	0.97	1.04
12	x	x	0.18	0.53	0.20	0.15	0.92	0.19	0.20
13	x	x	x	2.91	1.07	0.83	4.99	1.02	1.10
14	x	x	x	x	0.37	0.29	1.71	0.35	0.38
15	x	x	x	x	x	0.78	4.65	0.95	1.03
16	x	x	x	x	x	x	5.99	1.22	1.32
17	x	x	x	x	x	x	x	0.20	0.22
18	x	x	x	x	x	x	x	x	1.08

Table 4.7: Difference of the aspect ratio of the measured distances Table 4.2 and the aspect ratio of Hector SLAM measured distances 'observation 1' Table 4.5

number	2	3	4	5	6	7	8	9	10
1	0.02	0.06	0.34	0.04	0.02	0.01	0.02	0.01	−0.04
2	x	0.01	0.34	0.00	−0.01	−0.01	0.00	−0.01	−0.21
3	x	x	0.09	0.00	0.00	0.00	0.00	0.00	−0.07
4	x	x	x	−0.02	−0.01	−0.01	−0.01	−0.01	−0.08
5	x	x	x	x	−0.01	−0.01	0.00	−0.01	−0.10
6	x	x	x	x	x	0.00	0.00	0.00	−0.11
7	x	x	x	x	x	x	0.01	0.00	−0.14
8	x	x	x	x	x	x	x	−0.01	−0.14
9	x	x	x	x	x	x	x	x	−0.13

number	11	12	13	14	15	16	17	18	19
1	0.04	−0.40	0.05	−0.07	0.12	0.03	0.81	0.05	0.02
2	0.00	−0.85	0.02	−0.26	0.12	0.00	0.92	0.02	−0.02
3	0.00	−0.26	0.00	−0.08	0.03	0.00	0.26	0.00	−0.01
4	−0.02	−0.22	−0.02	−0.09	0.00	−0.02	0.05	−0.01	−0.02
5	0.00	−0.41	0.01	−0.13	0.05	0.00	0.42	0.01	−0.01
6	0.01	−0.50	0.02	−0.14	0.09	0.01	0.65	0.02	0.00
7	0.01	−0.64	0.03	−0.18	0.12	0.01	0.85	0.03	0.00
8	0.00	−0.57	0.02	−0.17	0.09	0.00	0.64	0.02	−0.01
9	0.01	−0.61	0.03	−0.17	0.11	0.01	0.80	0.03	0.00
10	0.01	−0.08	0.02	−0.01	0.03	0.01	0.20	0.02	0.01
11	x	−0.36	0.01	−0.11	0.05	0.00	0.40	0.01	−0.01
12	x	x	0.02	0.02	0.03	0.01	0.16	0.02	0.01
13	x	x	x	−0.14	0.04	−0.01	0.36	0.00	−0.02
14	x	x	x	x	0.03	0.01	0.21	0.02	0.01
15	x	x	x	x	x	−0.04	0.15	−0.04	−0.06
16	x	x	x	x	x	x	0.51	0.01	−0.01
17	x	x	x	x	x	x	x	−0.01	−0.02
18	x	x	x	x	x	x	x	x	−0.02

Table 4.8: Difference of the aspect ratio of the measured distances Table 4.2 and the aspect ratio of Hector SLAM measured distances 'observation 2' Table 4.6

number	2	3	4	5	6	7	8	9	10
1	0.00	0.08	0.15	−0.01	0.01	0.00	0.01	0.01	−0.15
2	x	0.14	0.25	0.00	0.02	0.00	0.02	0.02	−0.20
3	x	x	0.00	−0.03	−0.01	−0.01	−0.01	−0.01	−0.13
4	x	x	x	−0.01	−0.01	−0.01	−0.01	0.00	−0.07
5	x	x	x	x	0.01	0.00	0.01	0.01	−0.09
6	x	x	x	x	x	−0.01	0.00	0.00	−0.19
7	x	x	x	x	x	x	0.01	0.01	−0.19
8	x	x	x	x	x	x	x	0.00	−0.19
9	x	x	x	x	x	x	x	x	−0.22

number	11	12	13	14	15	16	17	18	19
1	0.02	0.42	0.00	−0.12	0.02	0.01	0.30	0.04	0.05
2	0.04	0.66	0.01	−0.15	0.04	0.02	0.48	0.06	0.08
3	−0.02	0.06	−0.02	−0.12	−0.02	−0.01	0.01	−0.01	0.00
4	−0.01	0.03	−0.01	−0.06	−0.01	−0.01	0.01	0.00	0.00
5	0.02	0.32	0.01	−0.06	0.02	0.01	0.24	0.03	0.04
6	0.00	0.31	−0.02	−0.16	0.00	0.00	0.20	0.02	0.03
7	0.03	0.52	0.00	−0.15	0.03	0.01	0.37	0.05	0.06
8	0.01	0.35	−0.01	−0.16	0.01	0.00	0.24	0.02	0.04
9	0.01	0.40	−0.02	−0.19	0.01	0.00	0.27	0.03	0.04
10	0.02	0.16	0.01	0.01	0.02	0.01	0.13	0.02	0.03
11	x	0.20	−0.01	−0.11	0.00	0.00	0.13	0.01	0.02
12	x	x	−0.01	−0.04	−0.01	−0.01	−0.01	−0.01	0.00
13	x	x	x	−0.08	0.01	0.01	0.19	0.02	0.03
14	x	x	x	x	0.01	0.01	0.11	0.02	0.02
15	x	x	x	x	x	0.00	0.12	0.01	0.02
16	x	x	x	x	x	x	0.19	0.02	0.03
17	x	x	x	x	x	x	x	0.00	0.00
18	x	x	x	x	x	x	x	x	0.01

4.3.2 Cartographer

The measured distances within the maps generated by Cartographer are shown in Table 4.9. Using these values, Table 4.10 is constructed to provide an overview of the differences between the manual measurements of the building floor and the distances measured within the maps generated by the Cartographer solution. 'Observation 1', 'observation 2' and 'observation 3' provide a respective average difference of -12.58, -4.15 and -5.36.

Tables 4.14, 4.15 and 4.16 provide an overview of the differences between the aspect ratios of the manual distance measurements of the reality and the distance measurements of the maps generated by Cartographer. These tables provide respectively the following averages for the aspect ratio difference: 1.4344, 0.0229 and 0.1068. The respective medians of the aspect ratio deviation are 0.0130, 0.0008 and 0.0101.

Examining the average distance difference and average aspect ratio difference of 'observation 1', one may determine the map shows different than the reality. The median for the aspect ratio deviation however shows a small number. The difference between the averages and the median may indicate that the map is mostly a good representation of the reality but will contain several mismatches. The averages and medians generated from the distance measures of 'observation 2' and 'observation 3' show that the aspect ratios of the generated map by the Cartographer algorithm is very close to the reality.

4.4 CPU and Memory Usage

For a variety of reasons, manufacturers choose different hardware components to build their devices. Unfortunately, every hardware component has its limitations by which it can operate. In many occasions it is important to know that the used hardware is able to handle the SLAM solution implemented. Problems may arise when the map generation and trajectory estimation lags too far behind on the movement of the robot or human equipped with the LiDAR. Within this section a comparison is made between CPU and memory usage of the selected SLAM solutions. The percentage of used memory and CPU during the generation of a map is plotted on a graph for each observational set. The plots will be evaluated for each of the SLAM solutions and can be used to make a comparison between different solutions.

To generate the plots, a batch script is written which takes a process-id as input parameter. The script will then retrieve the percentage of used CPU and memory every second of the process with the provided process-id and write it to a temporary file. When the script is stopped, it will output

Table 4.9: Distance measures of Cartographer generated maps. Numbers can be related to the locations indicated at Figure 4.8.

number	observation 1 (cm)	observation 2 (cm)	observation 3 (cm)
1	440.5	442.1	447.1
2	649.0	649.0	645.6
3	188.6	188.5	186.8
4	100.2	101.8	101.8
5	301.9	302.0	302.2
6	416.2	422.0	422.7
7	545.6	543.9	542.8
8	438.8	443.8	454.7
9	509.7	512.2	514.7
10	107.0	108.4	103.4
11	276.0	280.4	275.6
12	53.5	53.4	53.4
13	247.1	288.7	283.8
14	108.5	100.2	98.5
15	258.8	275.4	272.1
16	307.2	347.1	342.8
17	15.1	58.4	51.8
18	290.5	292.1	290.0
19	268.7	273.7	270.4

Table 4.10: Deviation of Cartographer measurements Table 4.9 against reality measurements Table 4.1

number	observation 1 (cm)	observation 2 (cm)	observation 3 (cm)
1	−6.5	−4.9	0.1
2	−7	−7	−10.4
3	−3.4	−3.5	−5.2
4	−4.3	−2.7	−2.7
5	−6.1	−6	−5.8
6	−8.3	−2.5	−1.8
7	−6.9	−8.6	−9.7
8	−9.7	−4.7	6.2
9	−12.8	−10.3	−7.8
10	4.5	5.9	0.9
11	−7.5	−3.1	−7.9
12	−3.5	−3.6	−3.6
13	−47.9	−6.3	−11.2
14	10	1.7	0
15	−19.7	−3.1	−6.4
16	−49.8	−9.9	−14.2
17	−46.4	−3.1	−9.7
18	−6.5	−4.9	−7
19	−7.3	−2.3	−5.6

Table 4.11: Aspect ratio between the measured distances from Table 4.9 'observation 1'

number	2	3	4	5	6	7	8	9	10
1	0.68	2.34	4.40	1.46	1.06	0.81	1.00	0.86	4.12
2	x	3.44	6.48	2.15	1.56	1.19	1.48	1.27	6.07
3	x	x	1.88	0.62	0.45	0.35	0.43	0.37	1.76
4	x	x	x	0.33	0.24	0.18	0.23	0.20	0.94
5	x	x	x	x	0.73	0.55	0.69	0.59	2.82
6	x	x	x	x	x	0.76	0.95	0.82	3.89
7	x	x	x	x	x	x	1.24	1.07	5.10
8	x	x	x	x	x	x	x	0.86	4.10
9	x	x	x	x	x	x	x	x	4.76

number	11	12	13	14	15	16	17	18	19
1	1.60	8.23	1.78	4.06	1.70	1.43	29.17	1.52	1.64
2	2.35	12.13	2.63	5.98	2.51	2.11	42.98	2.23	2.42
3	0.68	3.53	0.76	1.74	0.73	0.61	12.49	0.65	0.70
4	0.36	1.87	0.41	0.92	0.39	0.33	6.64	0.34	0.37
5	1.09	5.64	1.22	2.78	1.17	0.98	19.99	1.04	1.12
6	1.51	7.78	1.68	3.84	1.61	1.35	27.56	1.43	1.55
7	1.98	10.20	2.21	5.03	2.11	1.78	36.13	1.88	2.03
8	1.59	8.20	1.78	4.04	1.70	1.43	29.06	1.51	1.63
9	1.85	9.53	2.06	4.70	1.97	1.66	33.75	1.75	1.90
10	0.39	2.00	0.43	0.99	0.41	0.35	7.09	0.37	0.40
11	x	5.16	1.12	2.54	1.07	0.90	18.28	0.95	1.03
12	x	x	0.22	0.49	0.21	0.17	3.54	0.18	0.20
13	x	x	x	2.28	0.95	0.80	16.36	0.85	0.92
14	x	x	x	x	0.42	0.35	7.19	0.37	0.40
15	x	x	x	x	x	0.84	17.14	0.89	0.96
16	x	x	x	x	x	x	20.34	1.06	1.14
17	x	x	x	x	x	x	x	0.05	0.06
18	x	x	x	x	x	x	x	x	1.08

Table 4.12: Aspect ratio between the measured distances from Table 4.9 'observation 2'

number	2	3	4	5	6	7	8	9	10
1	0.68	2.35	4.34	1.46	1.05	0.81	1.00	0.86	4.08
2	x	3.44	6.38	2.15	1.54	1.19	1.46	1.27	5.99
3	x	x	1.85	0.62	0.45	0.35	0.42	0.37	1.74
4	x	x	x	0.34	0.24	0.19	0.23	0.20	0.94
5	x	x	x	x	0.72	0.56	0.68	0.59	2.79
6	x	x	x	x	x	0.78	0.95	0.82	3.89
7	x	x	x	x	x	x	1.23	1.06	5.02
8	x	x	x	x	x	x	x	0.87	4.09
9	x	x	x	x	x	x	x	x	4.73

number	11	12	13	14	15	16	17	18	19
1	1.58	8.28	1.53	4.41	1.61	1.27	7.57	1.51	1.62
2	2.31	12.15	2.25	6.48	2.36	1.87	11.11	2.22	2.37
3	0.67	3.53	0.65	1.88	0.68	0.54	3.23	0.65	0.69
4	0.36	1.91	0.35	1.02	0.37	0.29	1.74	0.35	0.37
5	1.08	5.66	1.05	3.01	1.10	0.87	5.17	1.03	1.10
6	1.50	7.90	1.46	4.21	1.53	1.22	7.23	1.44	1.54
7	1.94	10.19	1.88	5.43	1.97	1.57	9.31	1.86	1.99
8	1.94	10.19	1.88	5.43	1.97	1.57	9.31	1.86	1.99
9	1.83	9.59	1.77	5.11	1.86	1.48	8.77	1.75	1.87
10	0.39	2.03	0.38	1.08	0.39	0.31	1.86	0.37	0.40
11	x	5.25	0.97	2.80	1.02	0.81	4.80	0.96	1.02
12	x	x	0.18	0.53	0.19	0.15	0.91	0.18	0.20
13	x	x	x	2.88	1.05	0.83	4.94	0.99	1.05
14	x	x	x	x	0.36	0.29	1.72	0.34	0.37
15	x	x	x	x	x	0.79	4.72	0.94	1.01
16	x	x	x	x	x	x	5.94	1.19	1.27
17	x	x	x	x	x	x	x	0.20	0.21
18	x	x	x	x	x	x	x	x	1.07

Table 4.13: Aspect ratio between the measured distances from Table 4.9 'observation 3'

number	2	3	4	5	6	7	8	9	10
1	0.69	2.39	4.39	1.48	1.06	0.82	0.98	0.87	4.32
2	x	3.46	6.34	2.14	1.53	1.19	1.42	1.25	6.24
3	x	x	1.83	0.62	0.44	0.34	0.41	0.36	1.81
4	x	x	x	0.34	0.24	0.19	0.22	0.20	0.98
5	x	x	x	x	0.71	0.56	0.66	0.59	2.92
6	x	x	x	x	x	0.78	0.93	0.82	4.09
7	x	x	x	x	x	x	1.19	1.05	5.25
8	x	x	x	x	x	x	x	0.88	4.40
9	x	x	x	x	x	x	x	x	4.98

number	11	12	13	14	15	16	17	18	19
1	1.62	8.37	1.58	4.54	1.64	1.30	8.63	1.54	1.65
2	2.34	12.09	2.27	6.55	2.37	1.88	12.46	2.23	2.39
3	0.68	3.50	0.66	1.90	0.69	0.54	3.61	0.64	0.69
4	0.37	1.91	0.36	1.03	0.37	0.30	1.97	0.35	0.38
5	1.10	5.66	1.06	3.07	1.11	0.88	5.83	1.04	1.12
6	1.53	7.92	1.49	4.29	1.55	1.23	8.16	1.46	1.56
7	1.97	10.16	1.91	5.51	1.99	1.58	10.48	1.87	2.01
8	1.65	8.51	1.60	4.62	1.67	1.33	8.78	1.57	1.68
9	1.87	9.64	1.81	5.23	1.89	1.50	9.94	1.77	1.90
10	0.38	1.94	0.36	1.05	0.38	0.30	2.00	0.36	0.38
11	x	5.16	0.97	2.80	1.01	0.80	5.32	0.95	1.02
12	x	x	0.19	0.54	0.20	0.16	1.03	0.18	0.20
13	x	x	x	2.88	1.04	0.83	5.48	0.98	1.05
14	x	x	x	x	0.36	0.29	1.90	0.34	0.36
15	x	x	x	x	x	0.79	5.25	0.94	1.01
16	x	x	x	x	x	x	6.62	1.18	1.27
17	x	x	x	x	x	x	x	0.18	0.19
18	x	x	x	x	x	x	x	x	1.07

Table 4.14: Difference of the aspect ratio of the measured distances Table 4.2 and the aspect ratio of Cartographer measured distances 'observation 1' Table 4.11

number	2	3	4	5	6	7	8	9	10
1	0.00	0.01	0.12	0.01	0.01	0.00	0.01	0.01	−0.24
2	x	0.02	0.20	0.02	0.01	0.00	0.02	0.02	−0.33
3	x	x	0.04	0.00	0.00	0.00	0.00	0.00	−0.11
4	x	x	x	−0.01	−0.01	−0.01	0.00	0.00	−0.08
5	x	x	x	x	0.00	0.00	0.00	0.00	−0.18
6	x	x	x	x	x	−0.01	0.00	0.00	−0.25
7	x	x	x	x	x	x	0.01	0.01	−0.29
8	x	x	x	x	x	x	x	0.00	−0.27
9	x	x	x	x	x	x	x	x	−0.33

number	11	12	13	14	15	16	17	18	19
1	0.02	0.39	0.27	−0.48	0.10	0.18	21.90	0.01	0.02
2	0.04	0.62	0.40	−0.68	0.15	0.28	32.31	0.03	0.04
3	0.01	0.16	0.11	−0.21	0.04	0.08	9.37	0.00	0.01
4	−0.01	0.04	0.05	−0.14	0.01	0.03	4.94	−0.01	−0.01
5	0.01	0.24	0.18	−0.34	0.06	0.12	14.99	0.00	0.01
6	0.01	0.33	0.25	−0.47	0.08	0.17	20.66	0.00	0.01
7	0.03	0.51	0.34	−0.58	0.12	0.23	27.15	0.02	0.03
8	0.01	0.33	0.26	−0.51	0.09	0.17	21.77	0.00	0.01
9	0.00	0.36	0.29	−0.61	0.09	0.20	25.26	0.00	0.00
10	0.03	0.20	0.09	−0.05	0.05	0.06	5.42	0.02	0.03
11	x	0.19	0.16	−0.33	0.05	0.10	13.67	0.00	0.00
12	x	x	0.02	−0.09	0.00	0.01	2.62	−0.01	−0.01
13	x	x	x	−0.72	−0.10	−0.02	11.57	−0.14	−0.15
14	x	x	x	x	0.07	0.08	5.58	0.04	0.05
15	x	x	x	x	x	0.06	12.61	−0.05	−0.05
16	x	x	x	x	x	x	14.54	−0.14	−0.15
17	x	x	x	x	x	x	x	−0.16	−0.17
18	x	x	x	x	x	x	x	x	0.01

Table 4.15: Difference of the aspect ratio of the measured distances Table 4.2 and the aspect ratio of Cartographer measured distances 'observation 2' Table 4.12

number	2	3	4	5	6	7	8	9	10
1	0.00	0.02	0.07	0.01	−0.01	0.00	0.00	0.01	−0.28
2	x	0.03	0.10	0.02	−0.01	0.01	0.00	0.01	−0.41
3	x	x	0.01	0.00	−0.01	0.00	0.00	0.00	−0.13
4	x	x	x	0.00	0.00	0.00	0.00	0.00	−0.08
5	x	x	x	x	−0.01	0.00	−0.01	0.00	−0.22
6	x	x	x	x	x	0.01	0.00	0.01	−0.25
7	x	x	x	x	x	x	−0.01	0.00	−0.37
8	x	x	x	x	x	x	x	0.01	−0.28
9	x	x	x	x	x	x	x	x	−0.37

number	11	12	13	14	15	16	17	18	19
1	0.00	0.44	0.02	−0.13	0.00	0.02	0.30	0.01	0.00
2	0.00	0.64	0.02	−0.18	0.00	0.03	0.45	0.01	−0.01
3	0.00	0.16	0.00	−0.07	0.00	0.01	0.11	0.00	−0.01
4	−0.01	0.07	0.00	−0.04	−0.01	0.00	0.04	0.00	−0.01
5	−0.01	0.25	0.00	−0.11	−0.01	0.01	0.16	0.00	−0.01
6	0.01	0.46	0.02	−0.10	0.01	0.03	0.32	0.02	0.00
7	−0.01	0.49	0.01	−0.18	−0.01	0.02	0.33	0.00	−0.01
8	0.00	0.44	0.02	−0.12	0.00	0.02	0.31	0.01	0.00
9	−0.02	0.43	0.00	−0.19	−0.02	0.01	0.27	−0.01	−0.02
10	0.03	0.23	0.03	0.04	0.03	0.03	0.19	0.03	0.02
11	x	0.28	0.01	−0.08	0.00	0.01	0.19	0.01	0.00
12	x	x	−0.01	−0.05	−0.01	−0.01	−0.01	−0.01	−0.01
13	x	x	x	−0.11	−0.01	0.01	0.15	0.00	−0.01
14	x	x	x	x	0.01	0.01	0.11	0.01	0.01
15	x	x	x	x	x	0.01	0.19	0.01	0.00
16	x	x	x	x	x	x	0.14	−0.01	−0.03
17	x	x	x	x	x	x	x	−0.01	−0.01
18	x	x	x	x	x	x	x	x	−0.01

Table 4.16: Difference of the aspect ratio of the measured distances Table 4.2 and the aspect ratio of Cartographer measured distances 'observation 3' Table 4.13

number	2	3	4	5	6	7	8	9	10
1	0.01	0.07	0.11	0.03	0.00	0.01	−0.01	0.01	−0.04
2	x	0.04	0.06	0.01	−0.02	0.00	−0.04	0.00	−0.16
3	x	x	0.00	−0.01	−0.01	0.00	−0.02	0.00	−0.07
4	x	x	x	0.00	−0.01	0.00	−0.01	0.00	−0.03
5	x	x	x	x	−0.01	0.00	−0.02	0.00	−0.08
6	x	x	x	x	x	0.01	−0.02	0.01	−0.05
7	x	x	x	x	x	x	−0.04	0.00	−0.14
8	x	x	x	x	x	x	x	0.03	0.02
9	x	x	x	x	x	x	x	x	−0.12

number	11	12	13	14	15	16	17	18	19
1	0.05	0.53	0.06	0.00	0.04	0.05	1.36	0.04	0.03
2	0.03	0.58	0.05	−0.11	0.02	0.05	1.80	0.02	0.01
3	0.00	0.13	0.01	−0.05	0.00	0.01	0.48	0.00	0.00
4	0.00	0.07	0.00	−0.03	0.00	0.00	0.27	0.00	0.00
5	0.01	0.26	0.02	−0.06	0.00	0.02	0.83	0.01	0.00
6	0.04	0.47	0.05	−0.02	0.03	0.04	1.26	0.03	0.03
7	0.02	0.47	0.04	−0.10	0.01	0.04	1.50	0.01	0.01
8	0.07	0.65	0.08	0.06	0.06	0.07	1.49	0.06	0.06
9	0.02	0.47	0.04	−0.08	0.02	0.04	1.44	0.02	0.01
10	0.01	0.14	0.02	0.01	0.01	0.01	0.33	0.01	0.01
11	x	0.19	0.01	−0.08	−0.01	0.01	0.71	0.00	−0.01
12	x	x	−0.01	−0.04	−0.01	0.00	0.10	−0.01	−0.01
13	x	x	x	−0.11	−0.02	0.00	0.68	−0.01	−0.02
14	x	x	x	x	0.01	0.01	0.30	0.01	0.01
15	x	x	x	x	x	0.01	0.72	0.00	0.00
16	x	x	x	x	x	x	0.81	−0.02	−0.03
17	x	x	x	x	x	x	x	−0.03	−0.03
18	x	x	x	x	x	x	x	x	0.00

the plot. To retrieve the percentage of used CPU and memory, the 'top' command is used and manipulated. One needs to bear in mind that the top-command outputs the CPU usage as a percentage of a single CPU. As the system which is used to implement the SLAM solutions is equipped with two CPUs, the percentage of used CPU can rise to a number higher than 100. Much of the code for this script has been retrieved at a public GIT published by Nicolas Azrzak [42]. The code had been adapted to output the CPU and memory usage. The original file used the 'ps' command to retrieve the requested data. This was changed to use the 'top' command as this matches more to the needs of the required graph. The code used within this project can be found within the appendices at Section A.8.

4.4.1 Hector SLAM

Monitoring the processes initiated by the Hector SLAM solution, it was noted that the systems starts two different processes. One process which handle mapping and one process handles trajectory estimation. As it is the goal to observe the complete usage of used CPU and memory by a single SLAM solution, the decision is made to take the sum of the CPU usage and memory usage of both Hector SLAM processes. The script written to plot these parameters is therefore adapted to take two process-ids as input. The adapted script can be retrieved in the appendices Section A.9.

When observing Figure 4.9 it becomes immediately clear that the memory usage of the Hector SLAM solutions is a flat line. This means that memory usage is consistent throughout the creation of the map and trajectory. Even when the map becomes larger, no extra memory is consumed. As Hector SLAM tries to relate each observation of the LiDAR directly onto the complete map generated so far, the solution only has to maintain one single map. The consistent memory usage therefore makes perfect sense.

Looking at the percentage of CPU used by the Hector SLAM solution within Figure 4.9, it can be noted that the CPU usage is quite consistent as well. The majority of the data lay between 13.4% and 0 with several peaks up to 20%. When looking a little deeper into the numbers, it's striking that al percentages of the used CPU are multipliers of 6.7. The reason for this is unknown and will not further be investigated.

Overall the Hector SLAM CPU and memory usage is very consistent throughout the life cycle of the algorithm. The approach used within this solution tries to reduce any overhead of big mathematical problem solutions. It can be concluded that many systems will be able to implement and use this solution even when creating large maps or using modest hardware.

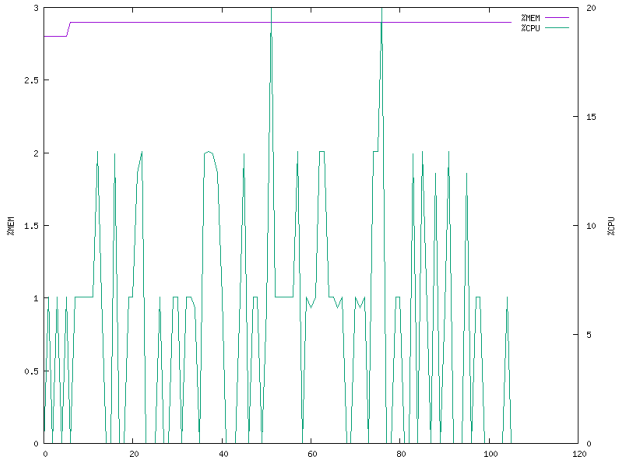
4.4.2 Cartographer

Figure 4.10 shows for that for every observation, the Cartographers memory increases over time. The larger the amount of collected LiDAR readings, the larger the memory usage becomes as well. This can be explained by the Cartographers implementation. Cartographers local SLAM component builds different sets of submaps to be compared and aligned by the global SLAM component. Each of these submaps is stored which causes the memory usage to grow.

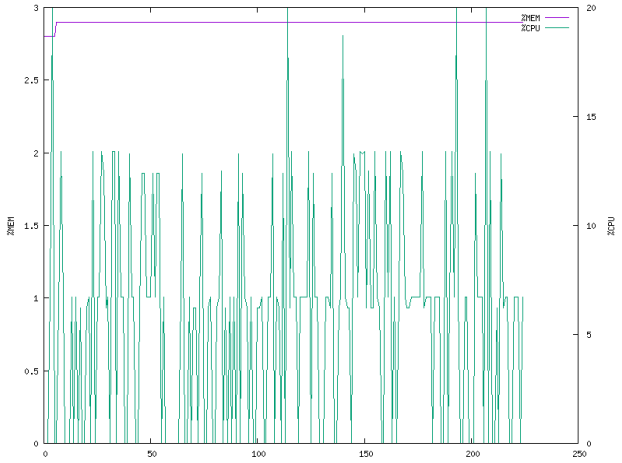
Different observations about CPU usage can be made when looking at Figure 4.10. Several peaks of CPU usage are to be found which rise to numbers even higher than 100%. These peaks can be explained by the logic of behind the global SLAM component implemented by the Cartographer solution. Aligning the generated submaps to build one big 'truth' is a computational costly operation.

As the map grows, the average CPU usage between the peaks grows as well. It can be stated that the average CPU usage keeps increasing until a certain point. As the observations are not big enough to reach this point, no conclusions can be made about this statement.

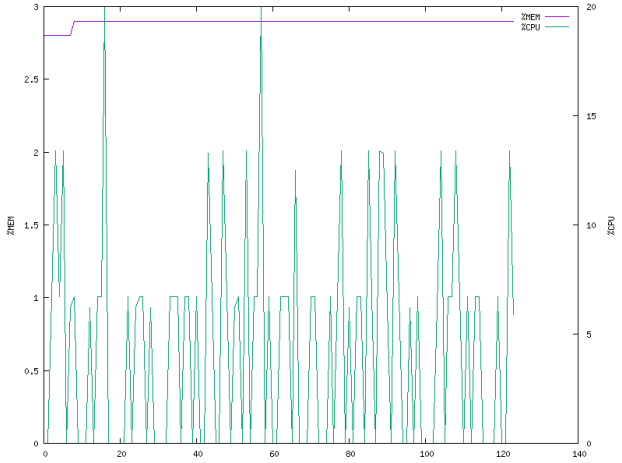
As a conclusion it can be said that the Cartographer solution is a computationally heavy algorithm. Combining two components, global and local SLAM, require a memory build up and a heavy CPU load. The system on which Cartographer was implemented did not experience the map generation to lag behind on the LiDAR movement. However this might be the case when more modest hardware is used to implement this solution.



(a) Observation 1

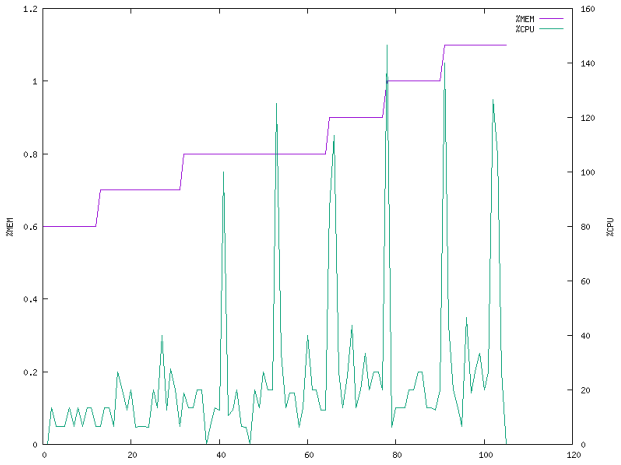


(b) Observation 2

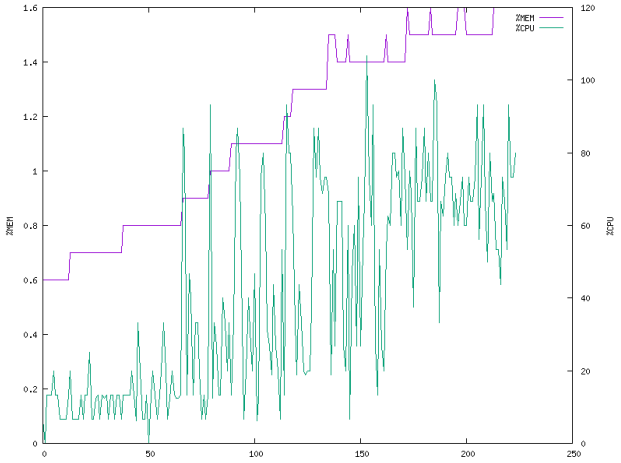


(c) Observation 3

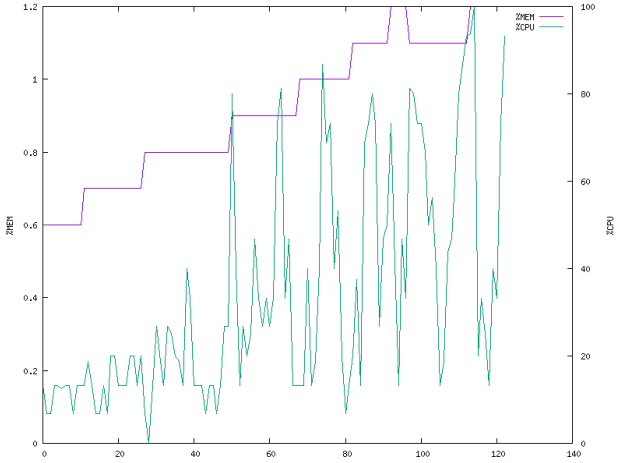
Figure 4.9: Graph of CPU percentage and memory used by Hector SLAM while mapping the observations



(a) Observation 1



(b) Observation 2



(c) Observation 3

Figure 4.10: Graph of CPU percentage and memory used by Cartographer while mapping the observations

5

Conclusion and Future Work

This thesis compared different SLAM solutions and their performance within a given building floor using exclusively LiDAR observational data as an input. The solutions investigated in this work are Hector SLAM and Google Cartographer. Both of these solutions can be implemented by using only LiDAR readings as an input. Section 2.2 explains feature-based SLAM algorithms which rely on odometry data and grid-based SLAM solutions which use a grid to represent maps. As only grid-based solutions are able to generate maps based on only observational data, both of the implemented SLAM solutions implement this type of approach.

Defining an overall best solution is not possible. When implementing, one should consider a best-fit to the needs of the project. As both approaches of Hector SLAM and Google Cartographer have their advantages and their trade-offs.

Hector SLAM tries to map every LiDAR observation onto the most fit location within one general map. Using this approach, no optimisation of the map needs to be done afterwards. Section 4.4 teaches that this approach uses a low amount of computational resources. Section 4.2 shows that Hector SLAM is not consistent in the way it generates a map. This becomes more obvious when being provided with a lower amount of LiDAR readings. The more observational data available, the more consistent the maps become. Sections 4.2 and 4.3 show that Hector SLAM performs best when the area

is scanned multiple times and swift rotations around the vertical axis of the LiDAR are avoided or at very low speed. Using this information, one may consider the implementation of Hector SLAM over Google Cartographer when being provided with modest hardware, a scenario in which the surface will be scanned multiple times and rotations around the vertical axis are (forced to be) limited.

Google Cartographer is built with two separate components. The first component, local SLAM, builds submaps by using a predefined amount of LiDAR observations. These submaps are then combined to create one general map by the second component, global SLAM. Section 4.4 clarifies that this implementation uses a fair amount of computational power and memory usage increases over time. By looking at Section 4.2, it is clear that Google Cartographer creates a consistent map when being provided with the same data. This makes the Cartographer solution a reliable partner for doing SLAM. The same section shows as well that Google Cartographer is, thanks to its two-component structure, able to cope with observational data of which the LiDAR makes swift rotations around its vertical axis. Both Sections 4.2 and 4.3 indicate that the mapping improves when the area is scanned multiple times. When being provided with decent hardware, a scenario in which reliability is of the essence or when the LiDAR will rotate around its vertical axis, one may consider to implement the Google Cartographer solution.

Based upon Section 4.3, both Google Cartographer and Hector SLAM have similar aspect ratio differences compared to the reality for 'observation 2', which is the most ideal observation for both algorithms. One may question if this is due to the implementations of the SLAM solutions, poor measurement skills while manually measuring the floor, or bad LiDAR calibration.

As it is important to make the best choice when selecting a SLAM solution, one would like to extend this investigation. Before making concrete decisions it is advised to use multiple scanning environments to detect possible differences in behaviour. This thesis provides an approach on how to compare different SLAM solutions which can be applied to other available solutions. As more of these packages exist, it is advised to investigate different options than the ones proposed within this thesis.

A

Appendices

A.1 Hector SLAM - my_launch.launch

```
<?xml version="1.0"?>

<launch>

  <arg name="geotiff_map_file_path" default="$(find
     hector_geotiff)/maps"/>

  <param name="/use_sim_time" value="true"/>

  <node pkg="rviz" type="rviz" name="rviz"
    args="-d $(find hector_slam_launch)/rviz_cfg/mapping_demo.
       rviz"/>

  <include file="$(find hector_mapping)/launch/my_mapping.
     launch"/>

  <include file="$(find hector_geotiff)/launch/geotiff_mapper.
     launch">
```

```
    <arg name="trajectory_source_frame_name" value="
        scanmatcher_frame"/>
    <arg name="map_file_path" value="$(arg
        geotiff_map_file_path)"/>
  </include>

</launch>
```

A.2 Hector SLAM - my_mapping.launch

```
<?xml version="1.0"?>

<launch>
  <arg name="tf_map_scanmatch_transform_frame_name" default="
     scanmatcher_frame"/>
  <arg name="base_frame" default="base_link"/>
  <arg name="odom_frame" default="base_link"/>
  <arg name="pub_map_odom_transform" default="true"/>
  <arg name="scan_subscriber_queue_size" default="5"/>
  <arg name="scan_topic" default="scan"/>
  <arg name="map_size" default="2048"/>

  <node pkg="hector_mapping" type="hector_mapping" name="
     hector_mapping" output="screen">

    <!-- Frame names -->
    <param name="map_frame" value="map" />
    <param name="base_frame" value="$(arg base_frame)" />
    <param name="odom_frame" value="$(arg odom_frame)" />

    <!-- Tf use -->
    <param name="use_tf_scan_transformation" value="true"/>
    <param name="use_tf_pose_start_estimate" value="false"/>
    <param name="pub_map_odom_transform" value="$(arg
       pub_map_odom_transform)"/>

    <!-- Map size / start point -->
    <param name="map_resolution" value="0.050"/>
    <param name="map_size" value="$(arg map_size)"/>
    <param name="map_start_x" value="0.5"/>
    <param name="map_start_y" value="0.5" />
    <param name="map_multi_res_levels" value="2" />

    <!-- Map update parameters -->
    <param name="update_factor_free" value="0.4"/>
    <param name="update_factor_occupied" value="0.9" />
    <param name="map_update_distance_thresh" value="0.4"/>
    <param name="map_update_angle_thresh" value="0.06" />
```

```
    <param name="laser_z_min_value" value = "-1.0" />
    <param name="laser_z_max_value" value = "1.0" />

    <!-- Advertising config -->
    <param name="advertise_map_service" value="true"/>

    <param name="scan_subscriber_queue_size" value="$(arg
       scan_subscriber_queue_size)"/>
    <param name="scan_topic" value="$(arg scan_topic)"/>

    <!-- Debug parameters -->
    <param name="tf_map_scanmatch_transform_frame_name" value="
       $(arg tf_map_scanmatch_transform_frame_name)" />
  </node>

<node pkg="tf" type="static_transform_publisher" name="
   base_link_laser_broadcaster" args="0 0 0 0 0 0 /base_link /
   laser 100"/>
</launch>
```

A.3 Google Cartographer Installation Guide

On Ubuntu 18.04 with ROS Melodic Morenia use these commands to install the above tools:

```
sudo apt-get update
sudo apt-get install -y python-wstool python-rosdep ninja-build stow
```

After the tools are installed, create a new cartographer_ros workspace in 'catkin_ws'.

```
mkdir catkin_ws
cd catkin_ws
wstool init src
wstool merge -t src https://raw.githubusercontent.com/cartographer-
project/cartographer_ros/master/cartographer_ros.rosinstall
wstool update -t src
```

Now you need to install cartographer_ros' dependencies. First, we use rosdep to install the required packages. The command 'sudo rosdep init' will print an error if you have already executed it since installing ROS. This error can be ignored.

```
sudo rosdep init
rosdep update
rosdep install --from-paths src --ignore-src --rosdistro=${ROS_DISTRO} -y
```

Cartographer uses the abseil-cpp library that needs to be manually installed using this script:

```
src/cartographer/scripts/install_abseil.sh
```

Due to conflicting versions you might need to uninstall the ROS abseil-cpp using

```
sudo apt-get remove ros-${ROS_DISTRO}-abseil-cpp
```

Build and install.

```
catkin_make_isolated --install --use-ninja
```

A.4 Google Cartographer - my_carto.launch

```
<?xml version="1.0" ?>
<launch>
  <node name="cartographer_occupancy_grid_node" pkg="
     cartographer_ros" type="cartographer_occupancy_grid_node"
     args="-resolution 0.05"/>

  <node name="cartographer_node" pkg="cartographer_ros" type="
     cartographer_node" args="-configuration_directory $(find
     cartographer_ros)/configuration_files -
     configuration_basename my_carto.lua" output="screen" />

  <node name="rviz" pkg="rviz" type="rviz" required="true" args
     ="-d $(find cartographer_ros)/configuration_files/demo_2d.
     rviz" />
</launch>
```

A.5 Google Cartographer - my_carto.lua

```
include "map_builder.lua"
include "trajectory_builder.lua"

options = {
  map_builder = MAP_BUILDER,
  trajectory_builder = TRAJECTORY_BUILDER,
  map_frame = "map",
  tracking_frame = "base_link",
  published_frame = "base_link",
  odom_frame = "odom",
  provide_odom_frame = false,
  publish_frame_projected_to_2d = false,
  use_odometry = false,
  use_nav_sat = false,
  use_landmarks = false,
  num_laser_scans = 1,
  num_multi_echo_laser_scans = 0,
  num_subdivisions_per_laser_scan = 1,
  num_point_clouds = 0,
  lookup_transform_timeout_sec = 0.2,
  submap_publish_period_sec = 0.3,
  pose_publish_period_sec = 5e-3,
  trajectory_publish_period_sec = 30e-3,
  rangefinder_sampling_ratio = 1.,
  odometry_sampling_ratio = 1.,
  fixed_frame_pose_sampling_ratio = 1.,
  imu_sampling_ratio = 1.,
  landmarks_sampling_ratio = 1.,
}

TRAJECTORY_BUILDER_2D.min_range = 0.12
TRAJECTORY_BUILDER_2D.max_range = 10
TRAJECTORY_BUILDER_2D.missing_data_ray_length = 10

TRAJECTORY_BUILDER_2D.num_accumulated_range_data = 1

TRAJECTORY_BUILDER_2D.use_imu_data = false
```

```
TRAJECTORY_BUILDER_2D.use_online_correlative_scan_matching =
    true

return options
```

A.6 Installation of YDLidar ROS-Driver

Clone ydlidar_ros_driver package for github :

```
git clone https://github.com/YDLIDAR/ydlidar_ros_driver.git
ydlidar_ws/src/ydlidar_ros_driver
```

Build ydlidar_ros_driver package :

```
cd ydlidar_ws
catkin_make
```

Package environment setup :

```
source ./devel/setup.sh
```

Add permanent workspace environment variables. It's convenient if the ROS environment variables are automatically added to your bash session every time a new shell is launched:

```
$echo "source ~/ydlidar_ws/devel/setup.bash" >> ~/.bashrc
$source ~/.bashrc
```

Create serial port Alias

```
$chmod 0777 src/ydlidar_ros_driver/startup/*
$sudo sh src/ydlidar_ros_driver/startup/initenv.sh
```

A.7 YDLidar X4 Launch-File

```
<?xml version="1.0"?>
<launch>
  <node name="ydlidar_node" pkg="ydlidar_ros" type="
     ydlidar_node" output="screen" respawn="false" >
    <param name="port" type="string" value="/dev/ydlidar"/>
    <param name="baudrate" type="int" value="128000"/>
    <param name="frame_id" type="string" value="laser"/>
    <param name="resolution_fixed" type="bool" value="true"/>
    <param name="auto_reconnect" type="bool" value="true"/>
    <param name="reversion" type="bool" value="false"/>
    <param name="angle_min" type="double" value="-180" />
    <param name="angle_max" type="double" value="180" />
    <param name="range_min" type="double" value="0.1" />
    <param name="range_max" type="double" value="12.0" />
    <param name="ignore_array" type="string" value="" />
    <param name="frequency" type="double" value="8"/>
    <param name="samp_rate" type="int" value="5"/>
  </node>
</launch>
```

A.8 CPU and Memory Usage Plotting Code

```
trap ctrl_c INT

LOG=$(mktemp)
SCRIPT=$(mktemp)
IMAGE=$(mktemp)

echo "Output to LOG=$LOG and SCRIPT=$SCRIPT and IMAGE=$IMAGE"

cat >$SCRIPT <<EOL
set term png small size 800,600
set output "$IMAGE"
set ylabel "%MEM"
set y2label "%CPU"
set ytics nomirror
set y2tics nomirror in
set yrange [0:*]
set y2range [0:*]
plot "$LOG" using 3 with lines axes x1y1 title "%MEM", "$LOG"
   using 2 with lines axes x1y2 title "%CPU"
EOL

function ctrl_c() {
 gnuplot $SCRIPT
 xdg-open $IMAGE
 exit 0;
}

while true; do
top -b -n 1 -p $1 | tail -1 | head -4 | awk '{gsub(",",".",$9);
    gsub(",",".",$10); print $1, $9, $10}' | tee -a $LOG
sleep 1
done
```

A.9 CPU and Memory Usage Plotting Code for Hector SLAM

```
trap ctrl_c INT

LOG=$(mktemp)
SCRIPT=$(mktemp)
IMAGE=$(mktemp)

echo "Output to LOG=$LOG and SCRIPT=$SCRIPT and IMAGE=$IMAGE"

cat >$SCRIPT <<EOL
set term png small size 800,600
set output "$IMAGE"
set ylabel "%MEM"
set y2label "%CPU"
set ytics nomirror
set y2tics nomirror in
set yrange [0:*]
set y2range [0:*]
plot "$LOG" using 3 with lines axes x1y1 title "%MEM", "$LOG"
   using 2 with lines axes x1y2 title "%CPU"
EOL

function ctrl_c() {
 gnuplot $SCRIPT
 xdg-open $IMAGE
 exit 0;
}

while true; do
top -b -n 1 -p $1, $2 | tail -2 | head -4 | awk '{gsub(",",".",
   $9); gsub(",",".",$10); sumCPU += $9; sumMEM += $10; print
   $1, sumCPU, sumMEM}' | tail -1 | tee -a $LOG
sleep 1

done
```

Bibliography

[1] Ramsey Faragher and Rob Harle. Smartslam - an efficient smartphone indoor positioning system exploiting machine learning and opportunistic sensing. In *ION GNSS*, volume 13, pages 1–14, 2013.

[2] Tim Bailey and Hugh Durrant-Whyte. Simultaneous localization and mapping (slam): Part ii. *IEEE robotics & automation magazine*, 13(3): 108–117, 2006.

[3] Sungnam Lee, Yohan Chon, and Hojung Cha. Smartphone-based indoor pedestrian tracking using geo-magnetic observations. *Mobile Information Systems*, 9(2):123–137, 2013.

[4] Dissanayake, MWM Gamini and Newman, Paul and Clark, Steve and Durrant-Whyte, Hugh F and Csorba, Michael. A Solution to the Simultaneous Localization and Map Building (SLAM) Problem. *IEEE Transactions on Robotics and Automation*, 17(3):229–241, 2001. doi: 10.1109/70.938381.

[5] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. *Aaai/iaai*, 593598, 2002.

[6] Kevin Lim, Paul Treitz, Michael Wulder, Benoît St-Onge, and Martin Flood. Lidar remote sensing of forest structure. *Progress in physical geography*, 27(1):88–106, 2003.

[7] Yiming Ji, Saâd Biaz, Santosh Pandey, and Prathima Agrawal. Ariadne: A dynamic indoor signal map construction and localization system. In *Proceedings of the 4th international conference on Mobile systems, applications and services*, pages 151–164, 2006.

[8] Frank Pedeflous. What does ultrasonic mean? `https://www.omegasonics.com/ultrasonic-cleaner-solutions/what-does-ultrasonic-mean/`, 2017. [Online; accessed March 3, 2021].

[9] Michal Kelemen, Ivan Virgala, Tatiana Kelemenová, Lubica Mikova, Peter Frankovskỳ, Tomáš Lipták, and Milan Lörinc. Distance measurement via using of ultrasonic sensor. *Journal of Automation and Control*, 3(3):71–74, 2015.

[10] Tarek Mohammad. Using ultrasonic and infrared sensors for distance measurement. *World Academy of Science, Engineering and Technology*, 51:293–299, 2009.

[11] Yongtae Do and Jongman Kim. Infrared range sensor array for 3d sensing in robotic applications. *International Journal of Advanced Robotic Systems*, 10(4):193, 2013.

[12] Jessica Maloney. Choosing the right distance sensor for your application. `https://www.terabee.com/choosing-right-distance-sensor-your-application/`, 2019. [Online; accessed March 11, 2021].

[13] Mikko Kytö, Mikko Nuutinen, and Pirkko Oittinen. Method for measuring stereo camera depth accuracy based on stereoscopic vision. In *Three-Dimensional Imaging, Interaction, and Measurement*, volume 7864, page 78640I. International Society for Optics and Photonics, 2011.

[14] Dhaval K Patel, Pankaj A Bachani, and Nirav R Shah. Distance measurement system using binocular stereo vision approach. *Int J Eng Res Technol*, 2(12):2461–2464, 2013.

[15] FRAMOS. Stereo applications need a dedicated lens choosing. `https://www.framos.com/en/news/stereo-applications-need-a-dedicated-lens-choosing`, 2018. [Online; accessed March 11, 2021].

[16] Xiao Ai, Richard Nock, John G Rarity, and Naim Dahnoun. High-resolution random-modulation cw lidar. *Applied optics*, 50(22):4478–4488, 2011.

[17] Refael Whyte, Lee Streeter, Michael J Cree, and Adrian A Dorrington. Application of lidar techniques to time-of-flight range imaging. *Applied optics*, 54(33):9654–9664, 2015.

[18] R Agishev, B Gross, F Moshary, A Gilerson, and S Ahmed. Range-resolved pulsed and cwfm lidars: Potential capabilities comparison. *Applied Physics B*, 85(1):149–162, 2006.

[19] Chao Zhang, Sze Yun Set, and Shinji Yamashita. Enhancement in dynamic range of amplitude-modulated continuous-wave laser scanner having a coaxial configuration. *IEEE Transactions on Instrumentation and Measurement*, 70:1–10, 2020. doi: 10.1109/TIM.2020.3011768.

[20] Yifei Jiang, Yun Xiang, Xin Pan, Kun Li, Qin Lv, Robert P Dick, Li Shang, and Michael Hannigan. Hallway based automatic indoor floorplan construction using room fingerprints. In *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*, pages 315–324, 2013.

[21] Georgios Pipelidis, Omid Reza Moslehi Rad, Dorota Iwaszczuk, Christian Prehofer, and Urs Hugentobler. A novel approach for dynamic vertical indoor mapping through crowd-sourced smartphone sensor data. In *2017 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 1–8. IEEE, 2017.

[22] Swadhin Pradhan, Ghufran Baig, Wenguang Mao, Lili Qiu, Guohai Chen, and Bo Yang. Smartphone-based acoustic indoor space mapping. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(2):1–26, 2018.

[23] Moustafa Alzantot and Moustafa Youssef. Crowdinside: Automatic construction of indoor floorplans. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, pages 99–108. Association for Computing Machinery, New York, NY, United States, 2012. ISBN 978-1-4503-1691-0.

[24] mathworks.com. What is slam? `https://nl.mathworks.com/discovery/slam.html/`, 2021. [Online; accessed June 13, 2021].

[25] Lukas Luft, Federico Boniardi, Alexander Schaefer, Daniel Büscher, and Wolfram Burgard. On the bayes filter for shared autonomy. *IEEE Robotics and Automation Letters*, 4(4):3286–3293, 2019.

[26] Howie M Choset, Seth Hutchinson, Kevin M Lynch, George Kantor, Wolfram Burgard, Lydia E Kavraki, and Sebastian Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementation*, chapter 9, pages 337–345. MIT press, 2005. ISBN 978-0262033275.

[27] Cyrill Stachniss. *Robotic Mapping and Exploration*, volume 55. Springer, 01 2009. ISBN 978-3-642-01096-5. doi: 10.1007/978-3-642-01097-2.

[28] Howie M Choset, Seth Hutchinson, Kevin M Lynch, George Kantor, Wolfram Burgard, Lydia E Kavraki, and Sebastian Thrun. *Kalman Filtering*, pages 269–300. MIT press, 2005. ISBN 978-0262033275.

[29] Cyrill Stachniss, John J Leonard, and Sebastian Thrun. Simultaneous localization and mapping. In *Springer Handbook of Robotics*, pages 1153–1176. Springer, 2016.

[30] Zhe Chen et al. Bayesian filtering: From kalman filters to particle filters, and beyond. *Statistics*, 182(1):1–69, 2003.

[31] Cyrill Stachniss and Wolfram Burgard. Particle filters for robot navigation. *Foundations and Trends® in Robotics*, 3(4):211–282, 2014. ISSN 1935-8253. doi: 10.1561/2300000013. URL `http://dx.doi.org/10.1561/2300000013`.

[32] semanticscholar.org. Introduction to particle filters. `https://www.semanticscholar.org/paper/2.1-Introduction-to-Particle-Filters/5898f7a39309b48a6e46fcd661ec455ef4c446e2`, . [Online; accessed June 13, 2021].

[33] Simo Särkkä, Aki Vehtari, and Jouko Lampinen. Rao-blackwellized particle filter for multiple target tracking. *Information Fusion*, 8(1):2–15, 2007.

[34] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *IJCAI*, volume 3, pages 1151–1156, 2003.

[35] Adam Milstein. Occupancy grid maps for localization and mapping. *Motion planning*, pages 381–408, 2008.

[36] G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46, 2007. doi: 10.1109/TRO.2006.889486.

[37] Alberto Elfes. Occupancy grids: A probabilistic framework for robot perception and navigation. *Computer*, 22:46 – 57, 1991. doi: 10.1109/2.30720.

[38] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-time loop closure in 2d lidar slam. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1271–1278. IEEE, 2016.

[39] The Cartographer Authors. Algorithm walkthrough for tuning. `https://google-cartographer-ros.readthedocs.io/en/latest/algo_walkthrough.html`, . Accessed: 2021-04-30.

[40] Stefan Kohlbrecher, Oskar Von Stryk, Johannes Meyer, and Uwe Klingauf. A flexible and scalable slam system with full 3d motion estimation. In *2011 IEEE international symposium on safety, security, and rescue robotics*, pages 155–160. IEEE, 2011.

[41] Joao Machado Santos, David Portugal, and Rui 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)*, pages 1–6. IEEE, 2013.

[42] Nicolas Azrak. graph.sh. `https://gist.github.com/nicolasazrak/32d68ed6c845a095f75f037ecc2f0436`, .