FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Sensor fusion and mapping of a wheeled mobile robot using vision and Lidar

Carolina Inês Pilar Pereira Marques

Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: António Pedro Rodrigues Aguiar

August 21, 2022

© Carolina Marques, 2022

Resumo

Com o crescimento exponencial na investigação sobre robôs autónomos, variadas subpartes deste tema surgiram e estão em constante desenvolvimento. Em particular, a percepção de um robot é fundamental para poder criar sistemas verdadeiramente autónomos, que possam interpretar os diferentes estímulos do seu ambiente e agir sobre e de acordo com eles. Esta característica tem múltiplas aplicações na vida real, tais como condução autónoma, operações em armazém, ou reconhecimento e mapeamento de ambientes desconhecidos. Semelhante aos sentidos de um ser humano, um robot percepciona o seu ambiente circundante utilizando diferentes sensores, e quanto mais precisas forem estas medições, melhor o sistema pode actuar sobre o seu ambiente.

A fusão de sensores é uma área que visa aumentar a qualidade das medições, fundindo os valores obtidos de diferentes sensores, tendo em conta as suas especificações, para obter um valor único final, que é mais preciso e representativo do valor real. Esta dissertação propõe-se a aplicar métodos de fusão de sensores em situações de mapeamento, nas quais um robot móvel tem de gerar um mapa de um ambiente desconhecido.

Em específico, o principal objectivo desta dissertação passa por aumentar a precisão de um sistema com dois sensores, de visão e LiDAR, e desenvolver um algoritmo que combine as medidas de ambos os sensores para obter melhores valores, quando comparados com os obtidos de uma solução com apenas um sensor. Para tal, foi desenvolvido um algoritmo de fusão de sensores com uma estratégia de correspondência baseada nos valores de direção e distância medidos pelos dois sensores, utilizando os princípios de filtros de Kalman para juntar os dois valores. Este sistema foi então integrado num algoritmo já existente de SLAM visual, ORB-SLAM, de forma a aplicar os desenvolvimentos a um método de mapeamento.

O trabalho desenvolvido produziu resultados favoráveis, com a solução de fusão de sensores a estimar pontos com uma melhoria na precisão de mais de 30% em todas as coordenadas, quando comparados com os pontos produzidos pela solução original com apenas SLAM visual. A solução proposta foi desenvolvida em simulação e utilizou um sensor LDS-01 LiDAR e uma câmara de profundidade Intel Realsense D435. Embora o mapa produzido ainda tenha algum ruído, os resultados provam que esta solução, simples em teoria, pode produzir melhores resultados do que as soluções que recorrem a apenas um sensor. O trabalho desenvolvido no âmbito desta dissertação fornece uma abordagem diferente e uma base sólida para trabalhos futuros, uma vez que a fusão de sensores em algoritmos SLAM é ainda um tópico em desenvolvimento.

Abstract

With the exponential growth of research on autonomous robots, numerous subsections of this subject have risen and are constantly in development. In particular, a robot's perception is fundamental to creating truly autonomous systems, which can interpret the different stimuli from their environment and act upon and according to them. This characteristic has multiple real-life applications such as autonomous driving, warehouse operations, or recognition and mapping of unknown environments. Similar to human senses, a robot perceives its surroundings using different sensors, and the more accurate these measurements are, the better the system can act upon these inputs.

Sensor fusion is an area that aims to increase the quality of the measurements by fusing values obtained from different sensors, taking into account their specifications, to get a single, more accurate and representative value. This master thesis proposes to apply sensor fusion methods in mapping situations where a robot has to generate a blueprint of an unknown environment.

Specifically, the main goal of this thesis is to increase the accuracy of a dual sensor system using vision and LiDAR and to develop an algorithm that combines the measurements of both sensors to obtain better results when compared to those obtained from a single-sensor solution. To do this, a sensor fusion algorithm was developed with a matching strategy based on the range and bearing values measured by the two sensors, using the principles of the Kalman filter to join the two values. This system was then integrated into an already existing visual SLAM algorithm, ORB-SLAM, to apply the developments to a mapping method.

The developed work delivered favourable results, with the sensor fusion solution estimating 3D points with an improvement in accuracy of more than 30% across all coordinates when compared to the points produced by the original visual SLAM solution. The proposed solution was developed in simulation and used an LDS-01 LiDAR sensor and an Intel Realsense D435 depth camera. Although the output map still has some noise, the results prove that this solution, simple in theory, can produce better results than mono-sensor solutions. This thesis work provides a different approach and a solid base for future work to be developed on top of, as sensor fusion in SLAM algorithms is still a topic in development.

Acknowledgements

I would like, first of all, to thank professor António Pedro Aguiar for all the support and guidance throughout the development of this thesis, without whom none of this would be possible. I would also like to thank Dr. Rómulo Rodrigues for sparing his time and giving me assistance, which was fundamental for the development of this work.

My mom and dad, for teaching me to never give up, and to know when to take breaks.

My support system back in Coimbra, for always having my back and believing in me.

Thank you to all the friends I made during these five years of university, for all the laughter and stories, and for teaching me life lessons that I wasn't able to learn inside a classroom but are still important nonetheless.

Thank you to João Sá for being a great research partner, and to Eduardo Parracho and João Santos, for helping me solve problem after problem, even when it wasn't their duty.

And lastly, thank you to Sofia, for everything.

Carolina Marques

"Our main business is not to see what lies dimly at a distance,
but to do what lies clearly at hand."

Thomas Carlyle

Contents

List of Figures

List of Tables

Abbreviations and Symbols

2D	Two Dimensions
3D	Three Dimensions
AR	Augmented reality
BA	Bundle Adjustment
BRIEF	Binary Robust Independent Elementary Features
BRISK	Binary Robust Invariant Scalable Keypoints
CSV	Comma Separated Values
DSO	Direct Sparse Odometry
DTAM	Dense Tracking and Mapping in Real-time
EKF	Extended Kalman Filter
FAST	Features from Accelerated Segment Test
FREAK	Fast Retina Keypoint
HOG	Histogram of Oriented Gradients
IEEE	Institute of Electrical and Electronics Engineers
LDS	Laser Distance Sensor
LiDAR	Light Detection and Ranging
LIFT	Learned Invariant Feature Transform
LSD	Large-Scale Direct Monocular
MAE	Mean Absolute Error
ORB	Oriented FAST and rotated BRIEF
rBRIEF	Rotation-Aware BRIEF
RGBD	Red Green Blue Depth
RMSE	Root Mean Square Error
ROS	Robot Operating System
SIFT	Scale Invariant Feature Transform
SLAM	Simultaneous Localization And Mapping
SURF	Speeded-Up Robust Features
SVO	Semi-direct Visual-Inertial Odometry
VLOAM	Visual-Lidar Odometry and Mapping

Chapter 1

Introduction

1.1 Context and Motivation

As time passes and technology gets more advanced, different situations arise for autonomous robot applications, ranging from industrial contexts with logistics and manufacturing to more domestic contexts, as simple as autonomous vacuum cleaners or sanitation robots.

In several cases, the addition of an automatic mobile robot has become an asset, which has produced significant advances in the area in recent years. However, a major problem in autonomous robots is their perception of the surrounding environment and in what way it can resemble the behaviour of a human being.

In the same way that humans have senses to capture information from their environment, autonomous systems have sensors that provide information upon which they can act. To be fully autonomous, a robot must be able to perceive an environment, make decisions based on what it discerns and then actuate accordingly. While in humans the use of information from different senses is natural, such as the association of glass shattering with the image of a cup falling to the ground and instinctively taking a step back, in autonomous systems this association needs to be created from scratch, with multiple parameters that need to be taken into consideration.

SLAM or Simultaneous Localization and Mapping, can be used to describe a robot's capacity to comprehend an unknown environment. As the name implies, SLAM consists of the ability to create a map of the environment where the robot is inserted while estimating its position. This concept appeared for the first time in 1986 [1], and since then, numerous algorithms have been developed using various sensors to capture information from their surroundings.

Robots can have only one or multiple sensors to perceive their surroundings, and while using only one sensor is more straightforward, sometimes it doesn't provide enough information about its environment. More than one sensor can be advantageous, as it adds information that would otherwise not be captured, but it also increases the system's complexity.

The two most commonly used types of sensors in SLAM methods are image and depth sensors, whose algorithms are called Vision-SLAM and LiDAR-SLAM, the last one alluding to the most common depth sensor, respectively LiDAR. It is most frequent to see only one type of sensor being

used in a SLAM algorithm, but multi-sensor SLAM solutions also exist and are further explored in Chapter 2.

1.2 Objectives

This thesis, inserted in the area of control systems and robotics, aims to develop a navigation algorithm that uses vision and LiDAR measurements. With these two inputs and Simultaneous Localization and Mapping (SLAM) methods, a robot running this algorithm should be able to map an unknown area, taking advantage of two very different sensors, as well as improve the accuracy of that map, when compared to a mono sensor solution.

These requirements will be fulfilled by taking the measurements from a vision sensor, an RGBD camera, and matching them with measurements from a range sensor, a LiDAR sensor. Then, using sensor fusion methods, which will be explained in more detail further in this document, the two inputs will be fused and integrated with a SLAM algorithm with the specific purpose of generating a more accurate map of the environment.

1.3 Structure of the thesis

This thesis will be divided into four main sections.

- Chapter 2, Background and Literature Review, contains existing solutions for the problem this thesis covers and an explanation of some basic topics for a better understanding of this work.
- Chapter 3, Methodology, contains all the work that has been developed for this thesis and explains how and why the work has been done.
- Chapter 4, Results, contains the results of the work done, as well as comparisons to already existing solutions.
- Chapter 5, Conclusion and Future Work, contains a summary of the work done and its findings, and what more can be done to extend the investigation done on this thesis

Chapter 2

Literature Review and Background

2.1 Literature Review

As it was referred in the introduction, multiple autonomous robots already exist in the market, and not all of them use SLAM. One example is Spot, the robotic dog created by Boston Dynamics[1], presented in Figure 2.1, which uses GraphNav [2] for its autonomous navigation mode. GraphNav uses topological maps but does not map the environment. Instead, there is a need to calibrate it with artificial landmarks before using it [2]. Although not in the SLAM category, this algorithm is faster than the usual SLAM methods since there is no need to pre-process input data. However, it is less robust than, for example, FastSLAM since it does not work when the robot is stuck or kidnapped.

[1] www.bostondynamics.com/products/spot
[2] https://www.bostondynamics.com/sites/default/files/inline-files/graphnav-technical-summary.pdf

Figure 2.1: Boston Dynamics' robotic dog, Spot. Source: [3]

On the other hand, multiple market-ready robots are already being sold that use both visual and LiDAR SLAM, similar to what is proposed in this thesis. BellaBot [4] shown in Figure 2.2, is one example. A robot created by Chinese company PuduTech, the BellaBot works as a waiter, with its main intent being to carry food in restaurants from the kitchen to the customers. This robot uses a dual SLAM solution, laser and visual, to map its environment and navigate through it. It also includes 3D sensors and obstacle-avoidance technology to avoid accidents, alongside other perks, like voice commands and touch sensors.

Figure 2.2: PuduTech's BellaBot. Source: [4]

Finally, it is worth mentioning autonomous driving as a significant example of situational awareness in action. Companies like Tesla[3] already have state-of-the-art technology for self-driving cars that is both fast and accurate, to the point where their algorithms are able to detect and avoid incoming collisions, locate the car in its environment by capturing and analyzing images of the car's surroundings and detecting street signs, other cars, and people.

Even though multiple examples of autonomous navigation exist in the market, with various solutions to solve this problem, more profound research was done in this thesis to understand different SLAM methods that already exist and to know where sensor fusion between visual and LiDAR SLAM stands.

2.1.1 Sensor fusion

In the context of this thesis, the topic of sensor fusion plays a key role. Sensor fusion, also referred to as data fusion or information fusion, consists in getting the measurements provided by two or more sensors, and fusing them together, to obtain better and more precise information (when compared with the use of a single sensor) that takes into account the values of the different sensors, and their specifications, such as uncertainty and noise-derived errors. The motivation for data fusion lies in the need to compensate for sensor failures and shortcomings such as imprecision, limited temporal and spatial coverage, or uncertainty. While sensor fusion allows a more robust and reliable system, with increased confidence and reduced ambiguity and uncertainty, it

[3] https://www.tesla.com/AI

still poses challenges related with synchronism between sensors, and associating the data from different sensors.

There are multiple ways to categorize sensor fusion, but the three more relevant ones to this work are three-level categorization, categorization based on input/output, and categorization based on sensor configuration [5].

On the three-level categorization, sensor fusion is catalogued based on how the data is processed [6]. The diagram in Figure 2.3 helps to understand better how the three levels work.

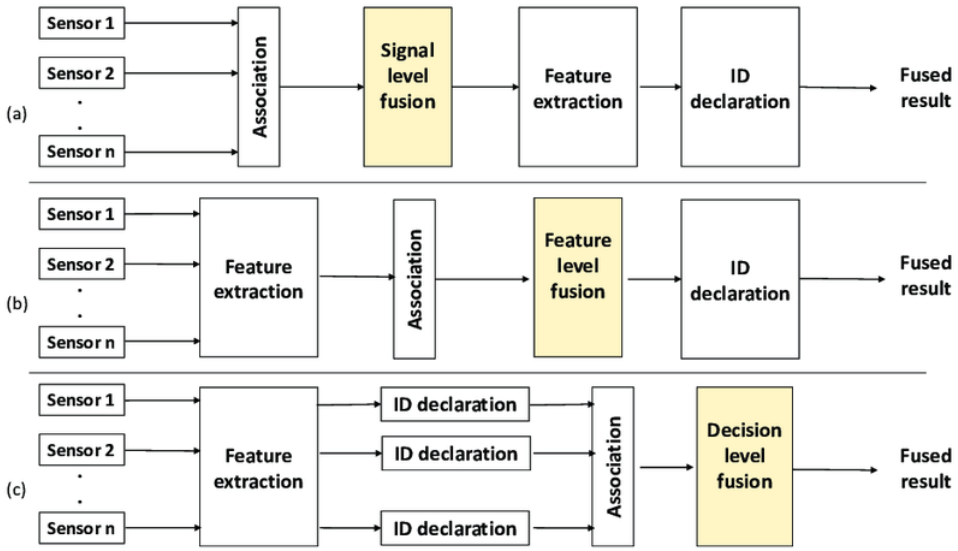

Figure 2.3: Data fusion at three levels. Source: [6]

- Low level, or raw data fusion, takes the raw information directly from multiple sensors to create new data that is more complete than the one produced by each sensor alone, which will then be analyzed and processed as if it was generated from one sensor alone.

- Intermediate level, or feature level fusion, is where features are extracted and fused, so the fusion is done not directly on the raw data, but after some processing of the information, combining all the features into one feature map, that can then be used by the rest of the system.

- High level, or decision level fusion, has each sensor processing and analyzing its data, and then uses a decision algorithm, such as Bayes filter or fuzzy logic, to finally fuse the data.

The input/output categorization is based on the three level categorization, and it was proposed by Dasarathy [7]. This arrangement derived from the need to have less ambiguous categories for sensor fusion, so Dasarathy extended from three to five categories, based on the input and output of each one. This classification is more cleary explained in Figure 2.4

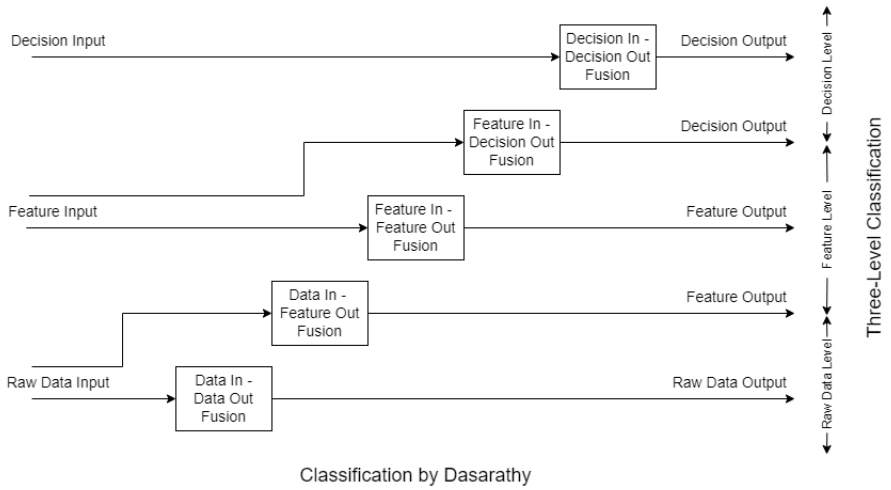

Figure 2.4: Input/Output categorization. Source: [8]

- Data In, Data Out, similar to the low-level category previously mentioned
- Data In, Feature Out, where the raw information from the sensors is taken, but the fusion method produces features based on all the data.
- Feature In, Feature Out, equivalent to the intermediate level in three-level categorization, where the features are fused into one feature map
- Feature In, Decision Out, which processes the features from each sensor and produces a decision based on them
- Decision In, Decision Out, which corresponds to the high-level categorization

The last categorization worth mentioning is based on sensor configuration and how they complement each other [9].

- Complementary: In this configuration, the sensors work independently from one another, solving the problem of the incompleteness that comes from using one sensor only
- Competitive: The multiple sensors are providing measurements of the same feature, making the measurement redundant. This is particulary used in sensors with high noise or low certainty
- Cooperative: With this setup, the sensors combined provide information that wouldn't be possible to obtain with each sensor by itself.

Besides categorizing sensor fusion, different methods exist for joining the sensor information. The more well-known process is the Kalman Filter, developed by Kalman in 1960 [10], an estimation algorithm that takes into account the covariances of the noises assuming that they have a

gaussian distribution with zero mean. The Kalman filter can also be altered to fit other scenarios such as non-linear systems, where there is a need to linearize around a certain point, which generated the Extended Kalman Filter. Different alterations can also be made to improve the performance of the Kalman Filter. It is the case of the Kalman Filter with Fuzzy logic, which uses fuzzy logic to make decisions based on the output of the filter. An example of this last configuration was done by Sasiadek and Hartana in 2000 [11] who used an Extended Kalman filter and Adaptive Fuzzy Logic System, using them to adjust the Kalman Filter gain and hinder the EKF divergence problem.

Sensor fusion can also be applied to occupancy grids. Occupancy grids are collections of cells that represent the states to be estimated and can have one of two states, occupied or unoccupied. Given its simplicity, it can be estimated by applying the Bayes rule, in which the probability of an event can be described by previous observations of conditions that could be related to that event. This Bayesian approach is usually described by the Bayes filter and was done in 1989 by Moravec [12] and, on a more recent example, in 2014, by Tripathi et al [13].

In 2002, Huanwong et al [14] wrote a paper about a decision theory called Dempster-Shafer, based on the Bayesian approach. This theory allows combining information from different sources, in this case, sensors, to arrive at a value with a certain degree of belief, represented by belief functions. In this work, the theory was altered to accept values from sources with different degrees of certainty.

With the development of Artificial Neural Networks, more works have been developed in sensor fusion using them. In 2021, Barreto-cubero et al [15] presented a solution for a tri-sensor configuration that used an artificial neural network to combine all the inputs into one reading that's more accurate to generate a 2D Occupancy Grid Map that considers the data from the three sensors.

2.1.2 SLAM

History

The first talks about Simultaneous Localization and Mapping, or SLAM, started in 1986 at the IEEE Robotics and Automation Conference held in San Francisco, California [16], where probabilistic approaches to the localization problem were first introduced. Many attempts were made to understand how a solution to this problem could be calculated, and researchers had already been investigating applying estimation methods to mapping and localization algorithms. The conclusion was that probabilistic mapping is a fundamental problem in robotics, and its computational and conceptual issues needed to be addressed.

In 1986 [**?**] Smith and Cheeseman released a paper about the relationship and estimated error between coordinate frames, representing relative locations of objects. This is a fundamental part of the SLAM problem since it tries to know the location of the robot on a map that is being created, and made it possible to know in advance whether an uncertain position was known accurately enough for some task, and how a certain sensor could improve that accuracy.

Following these works, Durrant-Whyte [17] proposed that this uncertainty could be represented as a fundamental part of geometric descriptions, and Smith, Self and Cheeseman [18] developed the stochastic map. This map is made of the estimates of the relationships among objects in the map, and their respective uncertainties.

Finally, in 1995, at the International Symposium on Robotics Research, the term SLAM was used for the first time, and valuable approaches were made. Durrant-Whyte, Rye and Nebot suggested that localization can be reduced to an estimation problem, based on observations of the local environment and proposed using the Kalman Filter for this process [19].

SLAM Methods

LiDAR SLAM is, as the name implies, a SLAM technique that uses LiDAR, Light Detection and Ranging, as its primary method of capturing information from its surroundings. Using a laser, LiDAR measures the time it takes for the reflected light to return to the receiver after targeting an object or surface.

EKF SLAM is a technique that uses an Extended Kalman Filter to estimate the pose of the robot, and it was one of the first methods to be discussed and proposed as a SLAM method, by Smith and Cheeseman in 1987 [20].

In 2002, a new SLAM algorithm called FastSLAM appeared [21]. This algorithm poses an alternative to Extended Kalman Filter-based methods and solves the SLAM problem by using particle filters, where each particle represents a possible location of the robot, and has N Kalman Filters, one for every landmark. Even though the accuracy of the method raises with the number of particles, too many particles make the algorithm too computationally expensive. So Grisetti, Stachniss et al.[22] proposed an algorithm to reduce the number of particles in the particle filters, and introduced what is known as Gmapping, which takes into account not only the movement of the robot but also its most recent observation.

An alternative later emerged, which used polar coordinates [23] to counteract the need to convert the laser scan coordinates to the Cartesian plane to be able to analyze them. This brought an advantage in processing time since the conversion step was skipped altogether, and the scan match became less expensive.

Finally, a loop-closure algorithm proposed in 2013 by [24] is worth mentioning. This algorithm introduced two different categories for scan features, geometric (related to the form) and numerical (pertaining to properties of measurements such as volume or curvature). This allowed a much more versatile scan match, and these two categories make comparing features easier, which means the loop-detection is more reliable.

Vision SLAM

Features-based SLAM

Features-based SLAM consists of estimating the state through matching images and tracking landmarks, extracting features from each frame and using those features to map the environment and

locate the robot.

In 2003, [25] Davidson, Reid et. al proposed the first monocular approach, MonoSLAM, which creates a map of natural landmarks using Kalman filters to estimate the features and the pose, and a motion model to smooth the camera movement. Later in 2007, [26] Klein and Murray proposed dividing tracking and mapping into two parallel tasks, one that tracks the erratic motion of the camera, and the other produces a 3D map of feature points observed in previous video frames. This approach, while better for smaller AR applications, showed clear difficulties when it came to loop closure.

Currently, ORB SLAM [27] is one of the most commonly used vision-based SLAM. It uses ORB features to identify landmarks, and it is a reliable and complete solution for monocular SLAM since it gathers all the best features of the previous solutions.

Direct SLAM

Direct SLAM is a method that skips the features extraction and instead focuses on aspects of each pixel in the frame and tracks those pixels from frame to frame. The most relevant algorithms are the DTAM [28], LSD [29], SVO [30], and DSO [31] direct SLAM methods. Although sometimes providing more accurate maps given the amount of information they can extract, they have a heavier computational load and can sometimes underperform in bad atmospheric conditions, introducing photometric errors as these methods rely a lot on aspects like the colour or brightness of a pixel.

LiDAR and Visual sensor fusion SLAM

After analyzing both visual and LiDAR SLAM on their own, and sensor fusion, it is possible to review the work that has already been done towards the fusion between these two specific sensors.

An actual LiDAR-camera fusion was attempted by Sun, Zhou et al. [32] which starts by extracting corner features from the LiDAR sensor and vertical lines from the monocular vision sensor, and fusing both inputs to the same feature. This means a more accurate map and, therefore, SLAM method. The work in [33] also tries to fuse both sensors' inputs, with the difference that, if the vision sensor (in this case an RGB-D camera) fails, the process only uses LiDAR data.

Instead of trying to fuse both inputs, VLOAM [34] has both the visual and the laser aspect running separately, with the camera input coming at a high frequency to estimate motion in a more general, low fidelity way, and the LiDAR input coming at a lower frequency to correct these motion estimates. Another work proposed by Seo and Chou [35] is inspired by VLOAM, but builds two maps from both inputs to have more constraints in the odometry residuals, which results in a more accurate and robust odometry.

Another solution proposed by Jiang, Yin et al [36] seems to be the fastest and most robust process when it comes to sensor fusion. The process described in their paper is based on a cost function that takes into account both the scan and image data as constraints and uses a graph-optimization method to optimize the pose of the robot. This process differentiates itself by building a 2.5D map with both the obstacle and vision features, which helps with a fast re-localization. Nonetheless, this still uses both inputs separately, without an actual fusion of the measurements.

Sensor Calibration

A major problem posed by sensor fusion lies in the need to calibrate both sensors, to determine the relative transformation between them. This calibration is done via extrinsic parameters, that can be calibrated manually. Added to this, these values can change daily, meaning that the calibration has to be done frequently. All of this is not ideal for an autonomous robot, so research has been done towards the automation of this process.

In 2005, Unnikrishnan and Hebert [37] proposed a user interface to help with this calibration, where the user manually matches points in order to get the extrinsic parameters. However, this process can be very time-consuming, so the need for a more automatic alternative still persisted. For this, Kassir and Peynot [38] presented an automated procedure for the calibration of a perspective camera with a 2D laser range finder. It does not provide a self-calibration method, but rather a two-part calibration method, assuming the intrinsic parameters of the laser are known. Going further, in 2010, Park, Kim and Sohn [39] proposed using a convoluted neural network to do this calibration online, with a calibration network to initially align the input sensor coordinate systems, and a depth fusion network, to encode the LiDAR and stereo depth characteristics together. This method proves to be relatively fast and very robust to changes in the extrinsic parameters.

2.1.3 Feature extraction

Feature extraction is the process of analysing an image and obtaining relevant points to then be processed and produce valuable information. These features can be corners, edges, colours, shadows or depths. One of the main obstacles in feature extraction is what points are relevant and useful for correctly and completely describing an image. A good feature should be invariant to rotation, orientation, translation, scaling, and should be consistent throughout different light conditions [40]. Multiple algorithms have been developed in order to optimize and complete the feature extraction process, and this subsection will mention some of the more relevant and modern techniques.

In 2004, Lowe et al [41] described an algorithm capable of extracting distinctive features from an image that could be used to perform matching between sets of images, called Scale Invariant Feature Transform, or SIFT. This method produces a high number of distinctive features, that are fundamental for object detection and feature matching

A year later, Dalal and Triggs [42] published a paper on human detection using an algorithm that used local histograms of image gradient orientations. This method is called Histogram of Oriented Gradients, HOG, and it is based on the idea that the appearance and shape of a local object can be characterized by the distribution of the gradients of local intensities of pixels or their edge detections. It defines corners and edges as the main features of an image.

In 2006, Rusten and Drummond [43] developed a less computationally intense method to detect features to be used in real-time solutions such as SLAM on a mobile robot. This algorithm is called Features from Accelerated Segment Test, or FAST, and detects corner features by comparing each pixel to its neighbours and verifying if it is the lowest or highest intensity in its

neighbourhood, deeming it a corner feature if this condition is verified. Although faster than other methods presented until its time, this algorithm underperforms for noisy pictures.

An algorithm based on SIFT arose in 2006 called Speeded-Up Robust Features, SURF, claimed to outperform its parent method [44]. It focuses on scale and rotation invariant detectors and descriptors, aiming to get enough distinctive keypoints without the algorithm being too computationally expensive, making it suitable for real-time systems [45].

In 2011, Calonder, Lepetit et al published a paper describing a new binary descriptor called BRIEF, for Binary Robust Independent Elementary Features [46]. Although computationally fast, it is very noise sensitive since it uses a gaussian kernel to perform smoothing on the image before converting it into a binary feature vector. The smoothing makes this algorithm fast, but it can lead to skipping the detection of features.

The most commonly known keypoint descriptor is called ORB, Oriented FAST and Rotated BRIEF, and it uses both the FAST keypoint detector and BRIEF descriptor methods [47]. It measures corner orientation using the intensity centroid, which assumes that its corner intensity is off-centre, making it possible to assign an orientation. Its performance is equipable to SIFT and better than SURF, while being faster than both of them.

Binary Robust Invariant Scalable Keypoints, or BRISK, is a method for keypoint detection, description and matching proposed in 2011 by Leutenegger, Chli and Siegwart [48]. An image and scale saliency criterion is used to identify points of interest. The characteristic direction of each point is determined by processing local intensity gradients at the neighbourhood of each keypoint by using a sampling pattern consisting of points lying on adequately scaled concentric circles. At last, the oriented BRISK sampling pattern obtains pairwise brightness comparison results, that are assembled into the binary BRISK descriptor.

In 2012, Alahi, Ortiz and Vandergheynst proposed a keypoint descriptor inspired by the human visual system and retina, called Fast Retina Keypoint, FREAK [49]. It is a method robust to orientation, scale and noise, and with a low calculation intensity, and it works by computing a cascade of binary strings, efficiently comparing image intensities over a retinal sampling pattern.

MatchNet, a system developed in 2015, is a deep convolutional network that combines learning feature representations and feature comparison functions for training a patch matching system [50]. A public release of pre-trained MatchNet is provided by the creators.

At last, it is worth mentioning another deep network architecture called Learned Invariant Feature Transform, LIFT, published in 2016 [51]. It handles detection, orientation estimation and feature description by learning to do all three tasks in a unified manner, while still maintaining end-to-end differentiability.

2.1.4 Review Discussion

Sensor fusion and keypoint detection can be done in multiple ways, and these have been topics of research and discussion for a long time. The aim of this thesis is not to do novel work in these areas, but rather to use the methods that most benefit the proposed solution. Likewise, SLAM processes have been researched and have established robust mono sensor methods, namely in

visual and LiDAR SLAM. However, there is a lack of robust methods when it comes to the actual fusion of both sensors. The main purpose of this thesis is to build on top of the work that has already been done and try to design a working algorithm to fuse both visual and LiDAR readings.

2.2 Background

Given this thesis topic, it is worth giving some context to the different parts that compose the proposed solution. This section will give some theoretical background on SLAM, ORB features, and the Kalman filter, topics fundamental for better understanding the proposed solution that is described in the next chapters.

2.2.1 SLAM

As mentioned in the Introduction, Simultaneous Localization and Mapping, or SLAM, describes a solution to the problem of placing a robot at an unknown location and having it to build a map of its environment while localizing itself inside the map [16]. The vehicle should be able to estimate its trajectory and the location of the landmarks in real-time, without the need for any a priori knowledge about the area inside which it is inserted.

A SLAM system includes two main components: the front-end and the back-end. The first is responsible for getting the sensor data and computing it into information that can be used for estimation, as well as data association tasks such as feature tracking and loop closure. The second component is responsible for performing inference on the information captured by the front end [52]. A typical configuration can be seen in Figure 2.5. It is worth noting that, in this Figure, MAP estimation means Maximum a Posteriori estimation, a Bayesian method used to calculate the estimate of an unknown quantity, based on maximizing the posterior probability in Bayes' theorem [53].

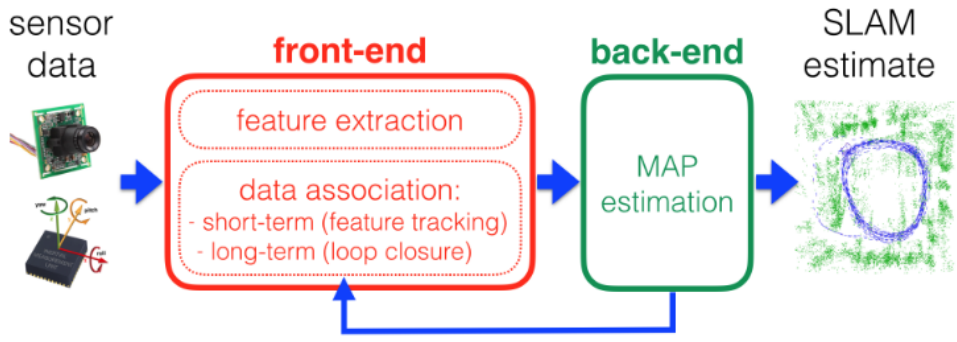

Figure 2.5: Structure of a typical SLAM System. Back-end information can sometimes be fed back into the front-end to assist in data association tasks. Source: [52].

A SLAM process can be divided into the following steps [54]:

1. Landmark Extraction
   This consists in collecting and registering features in the environment that are easily re-observed and distinguished (landmarks). Such features can include corners, blobs, or even edges, and can be represented by pointclouds, when the input sensor is a LiDAR, or images when the input sensor is a camera. In the latest case, sometimes distortion correction is also needed to be able to correctly extract these features.

2. Data Association
   This step takes the features extracted, as well as their estimated locations, and tries to match them (if it is the case) with the same features extracted in the past, or adds new features, in order to build the map. Common problems that arise in this step include as failure to observe the same landmarks in every time step, wrong identification of a landmark, or the wrong association of landmarks between time steps. To be able to minimize them, more computing power is needed, which means more time, so this process becomes a tradeoff between map accuracy and processing time.

3. Pose and Map Update
   Here, the process uses filters such as the Extended Kalman Filter (EKF) or the Particle Filter to estimate the pose of the mobile robot in relation to the map that has being built, updating the information according to what is being computed.

4. Loop closure
   As the system continues to get new information on its environment, it might recognize that it is at a point that has been previously visited. This is where loop closure comes into action, completing the map and correcting accumulated errors. A graphical explanation of how loop closure can be useful is shown in Figure 2.6.

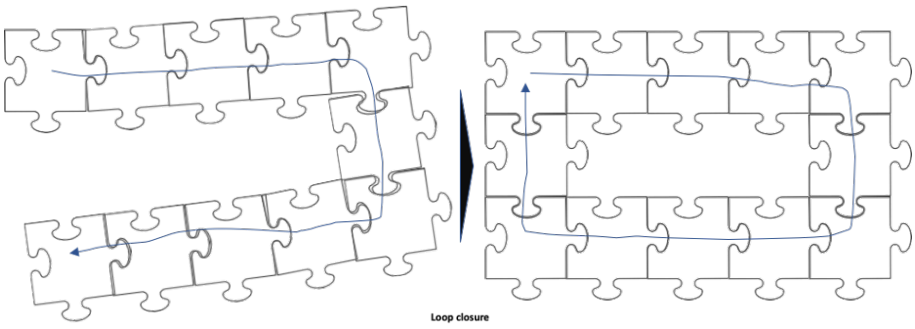

Figure 2.6: Graphic example of the "loop closure". Source: [55]

For this step, there are two methods to accomplish loop closure detection [56].

- *Geometry-based methods* use the known movement information of the vehicle to detect when the estimated position has been passed through before, verifying that there has a loop. This can be done using, for example, a front-end odometer. This solution

is simple and intuitive but can become unusable when the accumulated error becomes too large.

- *Features-based methods* consist in detecting features in the captured image and comparing it to previous frames to detect if the robot has already been in that position. This method, even though it eliminates the accumulated error that comes with geometry-based methods, can become very heavy, since there is a need to compare each new frame with every previous one captured.

2.2.2 ORB features

Even though many different feature extraction methods exist, ORB features are the ones used in this work as they are a fast and efficient way to detect and describe unique and relevant keypoints in any image. ORB means Oriented FAST and Rotated BRIEF, and as described in the previous section, are based on the FAST keypoint detector and BRIEF descriptor [47].

The algorithm starts by detecting FAST keypoints in the image. The FAST algorithm is a corner detection method developed in 2006 by Rosten and Drummond [43] that is very computationally efficient and suitable for real-time systems. The algorithm starts by selecting a pixel $\mathbf{p}$ in an image, to be evaluated as a keypoint or not, with intensity $\mathbf{I_p}$. Declaring a threshold value t, the algorithm stores a circle of 16 pixels around the chosen keypoint. This keypoint is considered a corner if n adjacent pixels in the circle are all brighter than $\mathbf{I_{p+t}}$ or darker than $\mathbf{I_{p-t}}$. Figure 2.7 presents this schematic to a real corner pixel. In the original version of the fast algorithm, n has the value of 12. In order to speed up this detection, the FAST algorithm starts by comparing 4 equidistant pixels in the circle. If at least 3 of these pixels satisfy the condition, the algorithm continues on to verify the remaining pixels, and if not, the keypoint is discarded. This procedure is then repeated for all the possible keypoints, until all of them are verified.

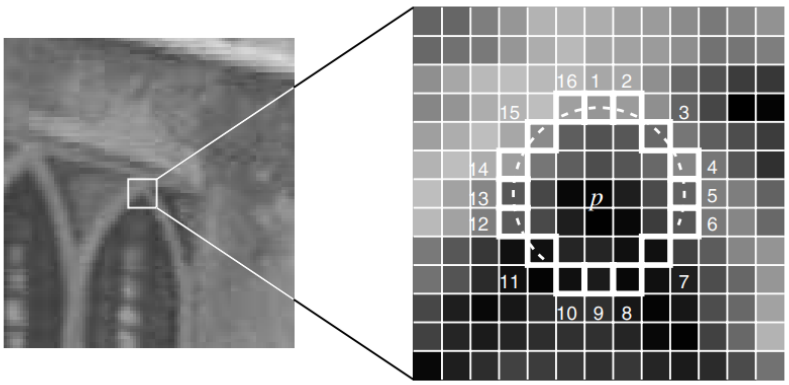

Figure 2.7: Corner detection in an image patch, with highlighted circle pixels.
Source: [43]

A few problems arise with this methodology. First, for n smaller than 12, the algorithm detects a number of keypoints that are too small. Besides that, the order by which the pixels are tested is of

importance for the speed of the algorithm. A way to solve this problem passes by using machine learning to optimize this process. Finally, the FAST algorithm isn't good at detecting corners in computer-generated images where the corner edges align perfectly with the x and y axes, or with sharp images, where the corner doesn't have a ring of darker/lighter pixels around it. Adding a gaussian blur to the image helps to reduce the sharpness of the edges, making the corners easier to detect by the algorithm

Notice also that FAST features do not take into account orientation or multiscaling. So, the ORB algorithm uses two methods to overcome these issues. For the differences in scaling, the ORB algorithm uses a multiscale image pyramid, which is a sequence of the same image at different resolutions, to run the FAST detector through in order to detect keypoints and guarantee scale invariation. With the purpose of assigning an orientation to each keypoint, the algorithm uses an intensity centroid, to detect how the intensity levels change around the keypoint. The intensity centroid assumes that a corner's intensity is offset from its centre, and is able to assign an orientation to the keypoint. The centroid of a patch is calculated with the moments of a patch, given by

$$m_{pq} = \sum_{x,y} x^p y^q I(x,y) \tag{2.1}$$

where $I(\cdot)$ is the intensity of the pixel at coordinates (x,y), and $p,q \geq 0$ are index order moments. The coordinates of the centroid, or centre of mass, of the patch, are calculated by

$$C = (\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}) \tag{2.2}$$

Finally, the orientation of the corner, calculated from the corner's centre to the centroid, is given by

$$\theta = atan2(m_{01}, m_{10}) \tag{2.3}$$

Having established the orientation, the keypoint can now be rotated to a standard direction and the descriptor is computed, making the algorithm rotation-invariant.

Afterwards, the BRIEF algorithm [46] computes the descriptors for each keypoint, starting by smoothing the image using a gaussian filter, to erase any high-frequency noise. To compute each descriptor, BRIEF selects random pairs of pixels in a pre-determined neighbourhood of the keypoint. Each pixel is selected from two normal distributions centred around the keypoint, one with a standard deviation of σ and another with a standard deviation of $\sigma/2$. Then, it runs a binary test τ, where p and q are the pairs of random points, and I the intensity of each point.

$$\tau(I; p_i, q_i) \begin{cases} 1: & I(p_i) < I(q_i) \\ 0: & I(p_i) \geq I(q_i) \end{cases} \tag{2.4}$$

For an n-bit vector, the algorithm repeats this process n times for each keypoint.

Since BRIEF falls short for in-plane rotation, ORB uses an altered version of the algorithm,

rBRIEF (rotation-aware BRIEF), which steers BRIEF according to the orientation of the keypoints. Considering S, a matrix of all the coordinates of the pairs of random points used to construct the descriptor vector for each keypoint, it is possible to obtain the steered version of S, S_θ, by multiplying the rotation matrix corresponding to the path orientation θ to the original S matrix, that is,

$$S_\theta = R_\theta S \tag{2.5}$$

For a regular BRIEF descriptor, the feature is defined as a vector of binary tests, f(n), with n being the size of the keypoint vector descriptor, and the points p and q belonging to S.

$$f(n) = \sum_{i=1}^{n} 2^{i-1} \tau(I; p_i, q_i) \tag{2.6}$$

The steered operator is then defined by

$$g_n(I, \theta) = f_n(I) | (p_i, q_i) \in S_\theta$$

Then, the algorithm constructs a lookup table of precomputed BRIEF patterns. As long as the orientation is consistent, the correct set of points from S_θ will be used to compute the keypoint's descriptor, making rBRIEF able to calculate keypoint descriptors invariant to rotation.

2.2.3 Kalman Filter

Random Variables

A random variable is, in essence, a function that maps points in a sample space to real numbers, for example, a sequence of positions over time. In continuous random variables, there's only a possibility to evaluate the probability of an event A within a certain interval. This is called the cumulative distribution function, and its derivative, more commonly used, is the probability density function.

Random variables have a mean and a variance associated with them. The mean also referred to as the expected value, is the weighted average of the occurrences of all the finite events of the random variable (x_i), taking into account each event's probability (p_i).

$$\text{Expected value of random variable } X = E(X) = \sum_{i=1}^{n} x_i p_i \tag{2.7}$$

This equation can be applied to functions of random variable X

$$E(g(X)) = \sum_{i=1}^{n} p_i g(x_i) \tag{2.8}$$

and with this equation, it is possible to obtain the variance of random variable X ($g(x_i)$)

$$\text{Variance } X = E[(X - E(X))^2] = E(X^2) - E(X)^2 \tag{2.9}$$

The variance is the expected value of the square of the difference between X, and its expected value, and it is very useful since it provides insights into how much a random variable varies around its mean value.

The normal/Gaussian distribution is a particularly well-known probability distribution that's commonly used to model random systems. A continuous random variable X that is normally distributed, with mean μ and variance σ^2, has its density function described by

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}} \tag{2.10}$$

State-Space Model

A dynamic process, such as the control of an autonomous vehicle, can be described by a state-space model. State-space models are convenient since they allow a more abstract look into dynamic systems, taking into account only their inputs, outputs and states, relating them with linear equations (for linear systems) as follows:

$$\mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k) \tag{2.11}$$

$$\mathbf{y}(k) = \mathbf{C}\mathbf{x}(k) + \mathbf{D}\mathbf{u}(k) \tag{2.12}$$

In discrete time-invariant systems, the equations 2.11 and 2.12 represent the state-space generalized model where

- **x** is the state vector
- **y** is the output vector
- **u** is the input/control vector
- **A** is the state matrix
- **B** is the input matrix
- **C** is the output matrix
- **D** is the feedforward matrix, and may not exist in certain cases.

Equation 2.11, which can also be referred to as the process model, represents a new state x_{k+1} modelled as a linear combination of the previous state x_k and the input of the system u_k. Equation 2.12, referred to as the measurement model, characterizes how the process measurements/observations y_k are related to the internal state x_k.

Almost all linear estimation methods rely on these two equations, such as the Kalman Filter, discussed in the following section.

Kalman Filter

The Kalman Filter appeared in 1960, in a work published by Rudolf E. Kalman [10] , as a recursive solution that accomplishes the prediction, separation, or detection of a random signal.

A filter plays the role of calculating the estimate of the system's state given its inputs and calculated outputs and some criteria to optimize. This is called the general filtering problem, a very discussed issue in the area of stochastic processes.

The Kalman filter is designed for estimating the state x of a discrete-time controlled process, which is represented by the following equations, portraying the process and the measurement model, respectively

$$x_k = Ax_{k-1} + Bu_k + w_{k-1} \tag{2.13}$$

$$z_k = Hx_k + v_k \tag{2.14}$$

It is worth noting the differences between equations 2.11 and 2.13 and equations 2.12 and 2.14. The Kalman filter equations add a new component to each equation, the process noise w_k and the measurement noise v_k. The signals w_k and v_k are random variables, considered independent from each other, and are assumed to have gaussian probability distributions, with zero mean, and covariances of Q and R, respectively

$$p(w) \sim N(0, Q) \tag{2.15}$$

$$p(v) \sim N(0, R) \tag{2.16}$$

Besides this, the output variable, symbolized in equation 2.12 as y_k, is now represented as z_k

The Kalman Filter estimates the system's state in two steps, which are run cyclically every time step, usually at different rates. The first one is called the prediction phase and includes the time update equations, and is normally ran at a higher frequency. This step is responsible for taking the previous state and error covariance estimated values to calculate a priori estimates $\hat{x}_k^-$ for the next step, the correction phase, that can run at a lower frequency. This includes the measurement update equations, and takes the new measurements to merge them with the a priori estimates, to obtain the final and improved a posteriori estimate $\hat{x}_k$. A priori and a posteriori are terms used to refer to the instantiations before and after the moment where the correction step happens.

The Kalman Filter equations are presented below, and their theoretical explanations can be found in [57]

Time update equations (Prediction step)

$$\hat{x}_k^- = A\hat{x}_{k-1}^- + Bu_k \tag{2.17}$$

$$\hat{P}_k^- = AP_{K-1}A^T + Q \tag{2.18}$$

Measurement update equations (Correction step)

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1} \tag{2.19}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(x_k - H\hat{x}_k^-) \tag{2.20}$$

$$P_k = (I - K_k H)P_k^- \tag{2.21}$$

In the prediction step, the estimated state of the system is calculated using the state-space equation 2.11 and its previous state, while the error a priori covariance is calculated taking into account the process noise covariance. The correction step starts by calculating the K matrix, or the Kalman gain. Then, it updates the state estimate, taking into account the observations taken. The H matrix is what relates the state of the system to its observations, and allows one to obtain the measurement prediction from the state estimate. Finally, this step ends with the update of the error covariance.

The Kalman Filter is a computationally simple filter to implement since it is a recursive cycle [10]. It still considers all the previous measurements and estimated values and allows them to propagate, without the need to save all of the values in memory, which is ideal for real-time systems.

Chapter 3

Methodology

This chapter is divided into two sections. The first explains the setup used during the work for this thesis and the second describes the proposed solution, as well as the work done during its development.

3.1 Setup

The purpose of this thesis is to develop sensor fusion between a vision and a depth sensor, namely an RGB-D camera and a LiDAR sensor, to work on a mobile robot in real-time. The proposed setup is set in simulation and includes a Turtlebot robot, with an LDS-01 LiDAR and an Intel Realsense D435 Depth Camera. Each component of the setup will be explained in more detail in the following sections, and how the components interact with one another will be explained in the section after.

3.1.1 ROS

Robot Operating System, or ROS, is a group of libraries and tools to develop robot applications[1]. It is based on a subscriber-publisher architecture, where each service, or group of services, in a project is represented by a node, which publishes topics for other nodes to subscribe to, and receives messages from other nodes by subscribing to their published topics. It is also possible to publish multiple topics simultaneously, synchronizing them, which is an important feature of a sensor fusion solution, since synchronism is very important to be able to match observations from different sources. An example of a ROS system can be found in Figure 3.1, where the circles represent the ROS nodes, and the arrows illustrate the topics communicated between nodes. The ROS version used in this thesis was ROS Melodic, running on Ubuntu 18.04.

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

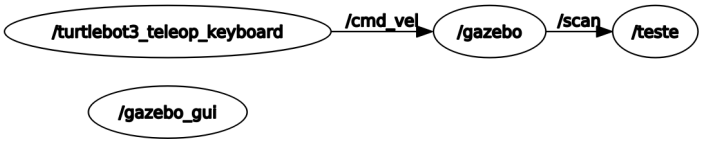

Figure 3.1: Visual example of ROS topics and nodes

3.1.2 Turtlebot3

The Tutlebot3, seen in Figure 3.2 is a ROS-based mobile robot created for development purposes[2]. As a modular system, it is possible to add other sensors besides the ones already included in the robot. This is convenient since the proposed setup requires an RGB-D sensor, which doesn't come included in the initial setup of the Turtlebot. Besides this, the Turtlebot has an open-source simulation, with different environments and setups, which makes it very easy to test algorithms without the need to use a real robot. The robot model used in this work was the Turtlebot burger, which includes an LDS-01 LiDAR sensor, that provides the depth component in sensor fusion.

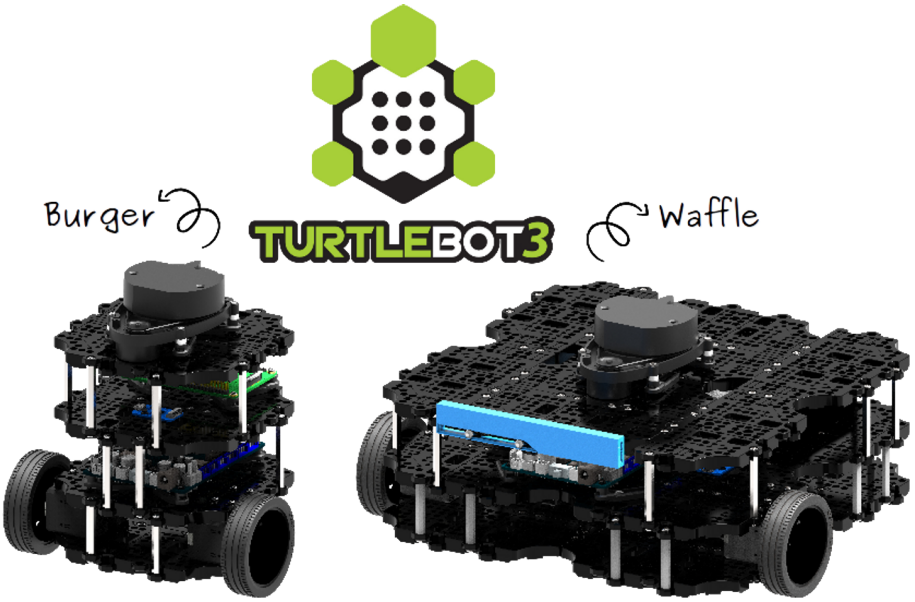

Figure 3.2: Turtlebot3 robots. Source: https://www.robotis.us/turtlebot-3-burger-us/ consulted 06/08/2022

[2]https://www.robotis.us/turtlebot-3-burger-us/

3.1.3 Gazebo and Rviz

Gazebo is an open-source simulator, that supports actuator control and sensor simulation[3]. An example of an environment provided with the turtlebot simulation that was used in this project, a simulated house, can be seen in Figure 3.3. This system is responsible for providing the simulated sensor values, according to the environment chosen, as well as providing a visual representation of the robot and how it is interacting with its surroundings.

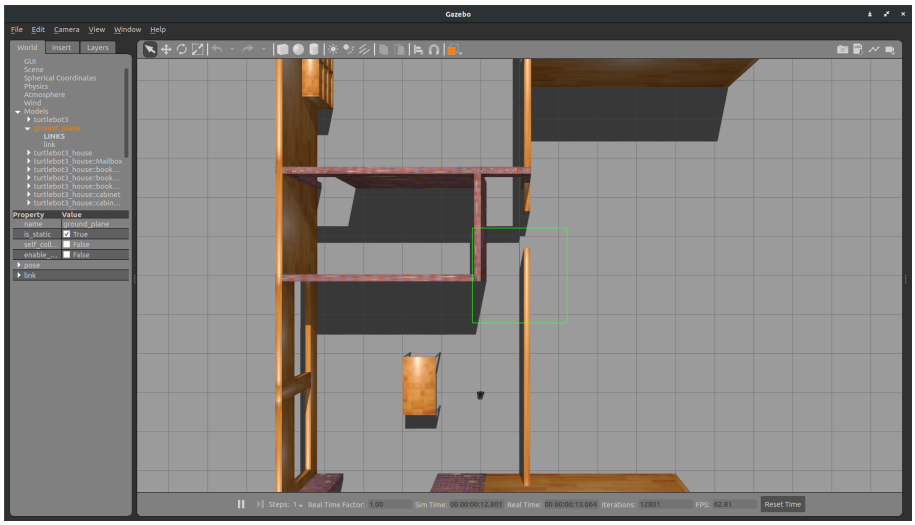

Figure 3.3: Turtlebot3 simulation viewed in Gazebo

Rviz is a 3D renderer for the aforementioned ROS framework [4]. It is a simpler tool, used to see different ROS topics published and subscribed to, and very useful to understand the information captured by the sensors in real-time, which helps during the development process. An example of RVIZ showing the LiDAR and Depth camera measurements, that are being published to their respective ROS topics, can be seen in Figure 3.4. The white points represent the depth points captured by the D435 and the red dots represent the LiDAR measurements.

[3] https://gazebosim.org/
[4] http://wiki.ros.org/rviz

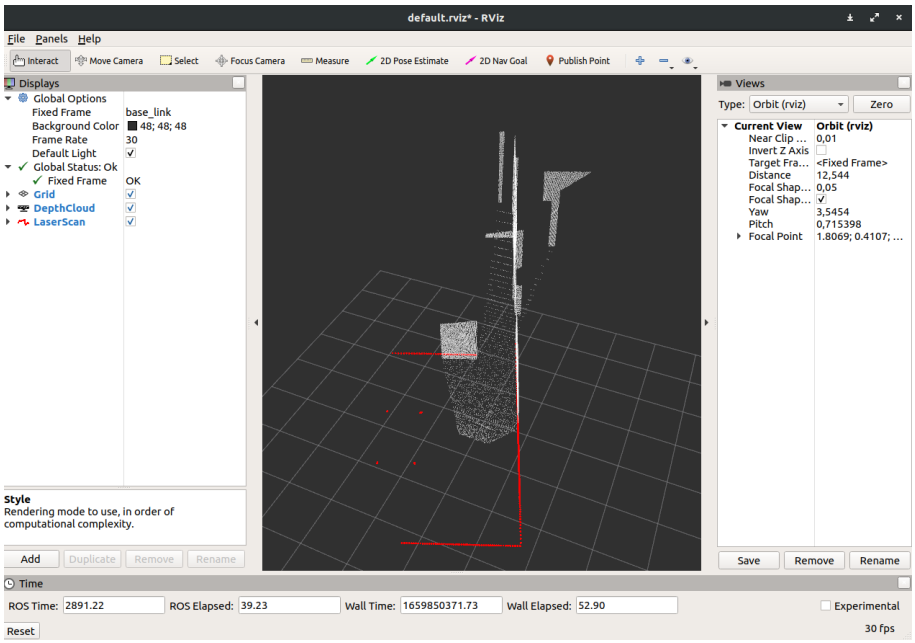

Figure 3.4: Turtlebot3 sensors measurements viewed in RVIZ.

3.1.4 Intel Realsense Sensor

The sensor responsible for vision in the proposed setup is the Intel RealSense Depth camera D435[5], shown in Figure 3.5. This sensor includes a stereo camera, an IR projector and an RGB module. The RGB module is responsible for creating the colour image, from where the keypoints are extracted, and the stereo camera is able to create a depth map. Through the camera's intrinsic parameters, it is possible to reproject the keypoints detected on the colour image into the depth map, and therefore extract their depth and real-world coordinates. This sensor has an ideal range of 0.3 meters up to 3 meters, with an accuracy of around 2% until 2 meters of distance, according to the sensor's datasheet [58].

Figure 3.5: RGB-D sensor used in the setup. Source: https://www.intelrealsense.com/depth-camera-d435/ consulted 06/08/2022

[5]https://www.intelrealsense.com/depth-camera-d435/

3.1.5 LiDAR

LiDAR is a technique used for determining depth values by pointing a laser at an object or surface and measuring the time needed for the reflected light to return to the receiver. The LiDAR sensor used in the proposed solution is shown in Figure 3.6, and it is the 360 Laser Distance Sensor LDS-01 by Robotis[6], which comes included with the Turtlebot3 burger setup. This sensor has a range of around 120 mm until 3500 mm, with an accuracy of 15mm for distances until 499mm, and 5.0% until 3500mm.

Figure 3.6: LiDAR sensor used in the setup. Source: https://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_lds_01/ consulted 06/08/2022

3.2 Implementation

The proposed solution is represented in Figure 3.7, which contains the different components and plugins used as rounded rectangles, and the ROS topics they communicate with, portrayed by arrows.

[6]https://www.robotis.us/360-laser-distance-sensor-lds-01-lidar/

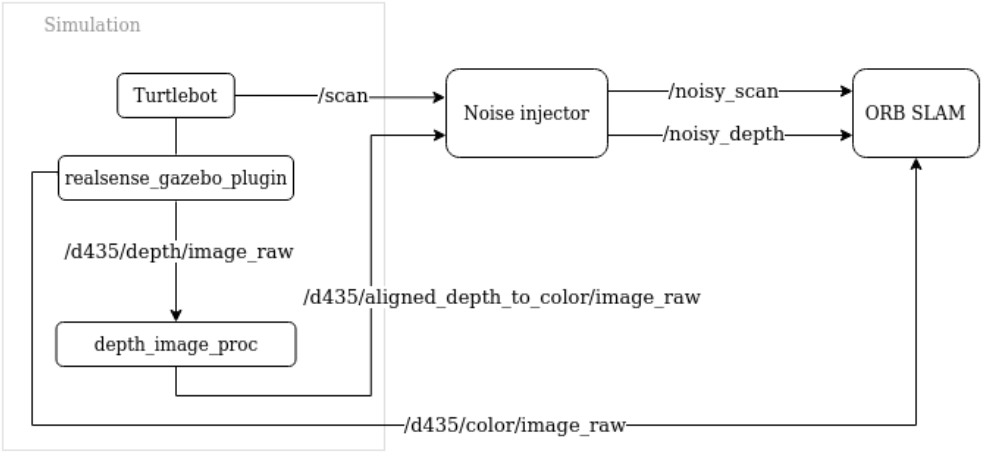

Figure 3.7: Proposed solution

The solution is composed of three main parts, the simulation, the noise injector, and the ORB-SLAM algorithm. Firstly, the simulation, composed of the Turtlebot and the Realsense sensor, acquires the points from the simulated environment and publishes them in their designated topics, */scan* for the LiDAR measurements, */d435/depth/image_raw* and */d435/color/image_raw* for the D435 outputs. The Turtlebot component is responsible for processing and publishing the measurements captured by the simulated LiDAR sensor. Since this element doesn't include the D435 sensor, it was necessary to add its simulation to the original package. This was done using the plugin *realsense_gazebo_plugin*, developed by Pal-robotics[7], which simulates the ROS topics published by the sensor, with the sensor attached to the robot inside Gazebo. This plugin uses Intel's simulation files[8], including the sensor description to add to the Gazebo simulation, in order to have a graphic rendering of the sensor.

With this setup, a working and controllable simulation is obtained, publishing all three topics needed for sensor fusion: the output from the LiDAR sensor, and both the RGB and Depth values captured by the Depth Camera. Despite this, two problems persisted, 1) the depth image and the colour image were being published in different frames, and 2) all the sensor outputs being published had no noise associated with them since they were capturing information inside a simulated world. As explained in Chapter 2's Kalman Filter section, this method uses the noise of both the process and the measurement model to estimate a value, making this component needed for the sensor fusion algorithm. In an ideal model, sensors would be able to capture inputs without any noise, meaning any system would have an accurate representation of the real world at all times, making sensor fusion redundant. Since this is not the case, and this simulation aims to represent a physical environment in a relatively faithful way, noise needs to be added to the simulated sensors' outputs

[7] https://github.com/pal-robotics/realsense_gazebo_plugin
[8] https://github.com/IntelRealSense/realsense-ros

To solve the first problem, a package called *depth_image_proc* was used. This package, created by Patrick Mihelich[9] contains nodelets, which are simpler and lighter versions of ROS nodes. Nodelets, instead of sending the information through ROS topics like in regular ROS nodes, only send the pointers to the information, which makes them significantly lighter and faster than regular nodes, which is appropriate to real-life systems. The nodelet used from this package is called *register*. It takes the camera's intrinsic parameters, as well as both the colour and depth camera topics, projecting the depth information to the colour frame, and republishing the reframed depth values through the topic *d435/aligned_depth_to_color*.

Then, with the information in the correct frames, noise was added to each of the measurements. This was done via an intermediate node, represented as the Noise Injector in Figure 3.7, which received the LiDAR and Depth camera measurements, added random gaussian noise to them according to each sensor's specification, and published the noisy measurements in two new ROS topics, *noisy_scan* and *noisy_depth*.

Afterwards, the ORB-SLAM algorithm receives the three processed topics, and runs the SLAM and Sensor Fusion algorithm, explained further in the following section.

3.2.1 ORB-SLAM

The algorithm used for the mapping of the robot is ORB-SLAM2, a version of the original ORB-SLAM algorithm, with support for RGB-D and stereo cameras. This version was developed by Raul Mur-Artal et al. [59] and the code is available on a public repository, which was used for this thesis development[10].

In opposition to the previous version, ORB-SLAM, which uses monocular vision and strictly feature-based techniques to apply SLAM, ORB-SLAM2 uses depth information to obtain the 3D coordinates for extracted features on the image, using stereo coordinates. For stereo cameras, the stereo keypoints are extracted from both cameras and then computed to generate a 3D point. For RGB-D cameras, which only have one camera, there's a need to generate a virtual right coordinate, u_R, based on the point's associated depth value, and the camera's estimated baseline. In this case, the baseline is the distance between the camera and the infrared transmitter. This virtual right coordinate is given by equation 3.1, where u_L is the keypoint coordinate in the x-axis in the RGB image, f_x is the focal length of the camera, b is the estimated baseline, and d is the depth value captured by the infrared sensor. With this, the algorithm is able to process the inputs provided by both types of cameras in the same way, taking into account the measured depth value in the case of RGB-D cameras.

$$u_R = u_L - \frac{f_x b}{d} \tag{3.1}$$

The ORB-SLAM2 algorithm runs essentially three main parallel threads, tracking, local mapping and loop closing. The tracking thread is responsible for the localization of the robot, detecting

[9] http://wiki.ros.org/depth_image_proc
[10] https://github.com/raulmur/ORB_SLAM2

keypoints and calculating matches between frames captured by the camera sensor. This thread is also responsible for minimizing errors using motion-only Bundle Adjustment, BA. Bundle Adjustment refers to a technique used to estimate the keypoints coordinates and camera positions, by minimizing a cost function [60]. Motion-only bundle adjustment only optimizes the camera position, while maintaining the coordinates of the points fixed.

The local mapping thread is in charge of the local map and performs local Bundle Adjustment, where Bundle Adjustment is applied only to a set of consecutive keyframes, and all the keypoints detected by them. The loop closing thread detects large loops and corrects all of the accumulated drift by performing a pose-graph optimization, estimating the camera's pose from all the relative pose measurements. This thread launches a fourth thread to perform full bundle adjustment, similar to local bundle adjustment but applied to all of the captured keyframes, after the pose-graph optimization, to compute the optimal trajectory and mapping. Figure 3.8 gives a more in-depth graphic representation of how the threads inside ORB-SLAM2 work.

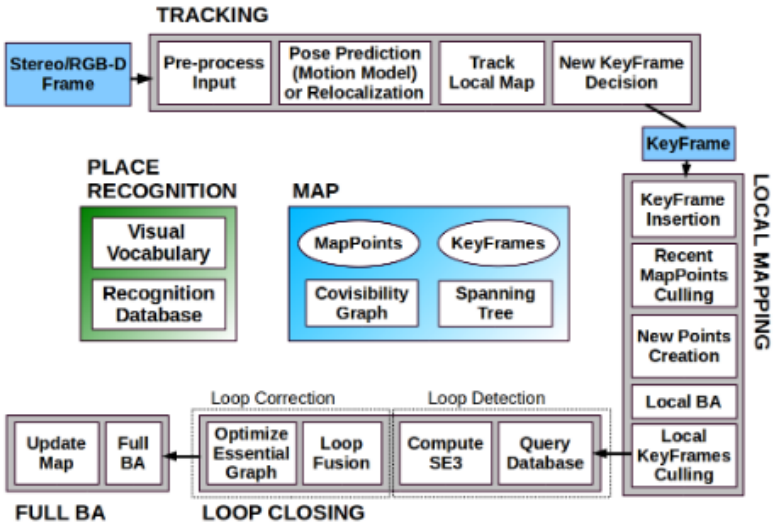

Figure 3.8: Threads in ORB-SLAM, Tracking, Local Mapping, Loop Closing and Full Bundle Adjustment, represented by the gray boxes, where each set of white boxes represents each thread's main functions. Source: [60]

The thread where the sensor fusion needs to be inserted is the tracking thread, in which each keypoint detected in a frame is converted to a 3D point in the map. In order to insert sensor fusion into the ORB-SLAM2 algorithm, the algorithm needs to subscribe to the altered LiDAR scan topic, and save it alongside the rest of the information provided by the RGB-D camera. This was done by associating a LiDAR scan measurements array with every new RGB frame received on the subscribed ROS topic.

3.2.2 Sensor fusion algorithm

After detecting the keypoints, and converting them to 3D coordinates using stereo projection, these points were fused with the LiDAR points using a Kalman Filter. The fusion algorithm has two steps, 1) matching a keypoint feature to a point in the laser scan range array, and 2) generating the final fused value of that point. For the matching step, it is necessary to convert the keypoint's 3D coordinates to polar coordinates, disregarding the height of the point. First, the keypoints' coordinates from the RGB-D frame are converted to the LiDAR scan frame. These frames are represented in Figure 3.9, where the blue refers to the camera referential, and the red to the LiDAR sensor referential, with dx being the displacement between the x-axis on the LiDAR and the z-axis on the camera, and dy the displacement between the y-axis on the LiDAR and the x-axis on the camera. Since the height is disregarded in this conversion, the top view is the fundamental representation to understand this change in frames, where x_L and z_C point to the front of the robot. Equations 3.2 and 3.3 demonstrate how the keypoint's x and y coordinates were converted to be represented in the LiDAR frame, considering the referential frames. Equation 3.4 indicates how the change to polar coordinates was calculated, with d representing the range of the observation to the origin of the frame, and θ representing the angle the observation makes with the x-axis.

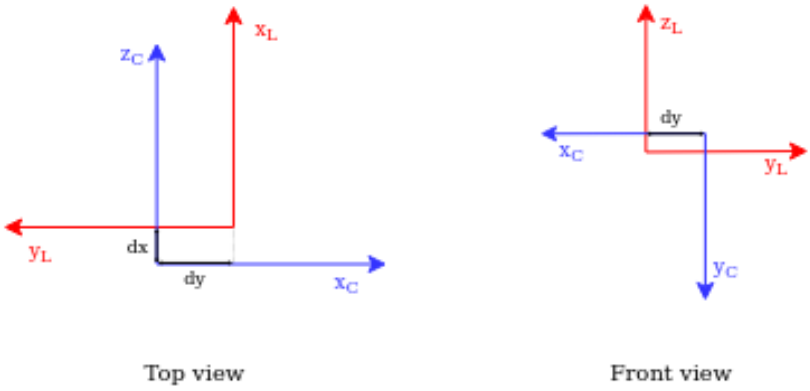

Figure 3.9: LiDAR sensor (red) and camera (blue) frames, represented in a top and front view

$$x_C^L \text{ (x-coordinate of the keypoint in the LiDAR frame)} = z_C - dx \tag{3.2}$$

$$y_C^L \text{ (y-coordinate of the keypoint in the LiDAR frame)} = dy - x_C \tag{3.3}$$

$$\begin{cases} \theta = atan2(y_C^L, x_C^L) \\ d = \sqrt{{x_C^L}^2 + {y_C^L}^2} \end{cases} \tag{3.4}$$

As previously mentioned, in the camera frame, the y coordinate, corresponding to the real-world height of a point, is disregarded in this conversion, since the LiDAR sensor captures points in 2D.

The LiDAR sensor publishes its measurements in an ordered vector, in ascending order of the bearing. This was taken advantage of in the final step of the input matching. The algorithm searches the scan values for a point with the closest distance to the calculated d, in equation 3.4, inside an interval with a predefined length n, and centred on the calculated θ of the keypoint. This way, the algorithm doesn't have to go through all of the points obtained by the LiDAR sensor to calculate a match, and instead only has to search inside a predefined length section. The scan point with the lowest difference in distance to the keypoint's distance is selected as the match to apply the Kalman Filter, as long as the difference is under a certain tolerance t. Both the range n and tolerance t are parameters that can be altered by the user, and its influences on the output are explained in more detail in Chapter 4. A flowchart representing this feature matching method can be seen in Figure 3.10.

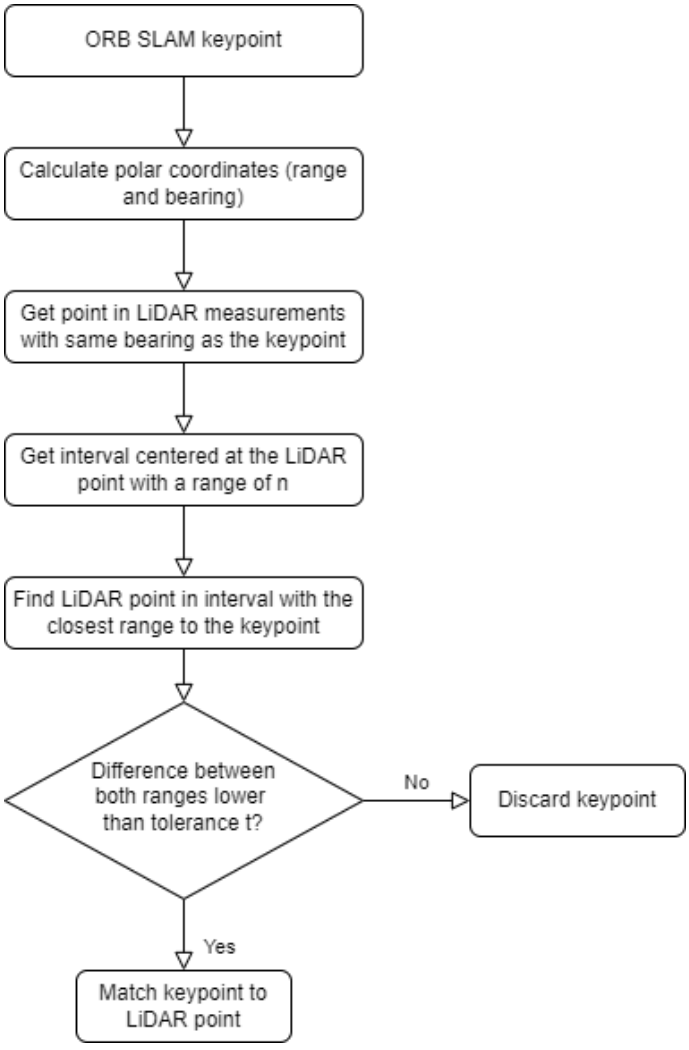

Figure 3.10: Feature matching algorithm flow chart

This approach ensures that the algorithm only applies sensor fusion to keypoints that are detected at the LiDAR sensor level, or keypoints that are on obstacles or surfaces that are perpendicular to and can reach the floor so that the laser sensor can measure its distance. This is useful to filter keypoints that aren't relevant for the map of the environment, such as shadows, and helps reduce the computational load of the system, while still maintaining an appropriate amount of points to build a meaningful map. Figure 3.11 helps understand which keypoints and scan points are chosen as matches, and which keypoints get filtered out.

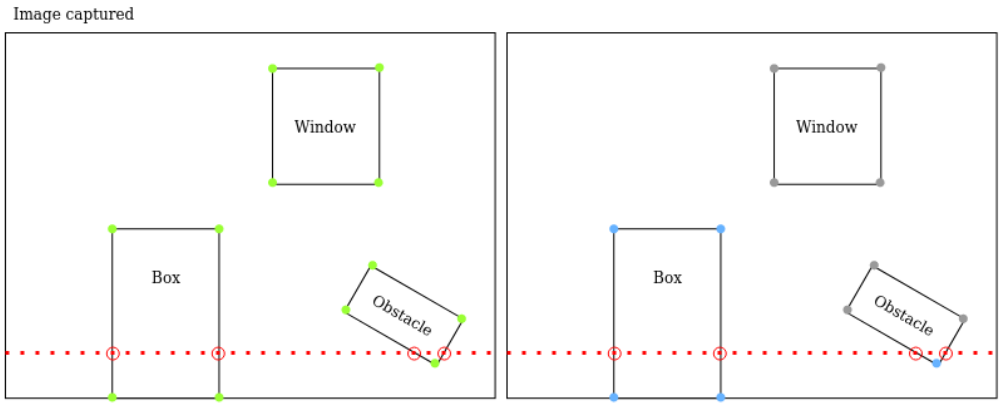

Figure 3.11: Image input of the environment with detected keypoints, before (left) and after (right) sensor fusion. In this scenario, all the objects are on a different plane than the background, closer to the robot. The red dots represent the laser scan values, and the red circles the laser scan values corresponding to an object. The green circles represent the ORB keypoints, before sensor fusion, whereas the grey circles represent the keypoints that were disregarded, and the blue circles are the ones that were effectively fused. It is worth noting that the window keypoints were discarded since the algorithm could not find any point in the laser scan values with a similar enough distance to the robot to perform sensor fusion.

With the coordinates from both sources, the keypoint's fused coordinates are calculated by applying the Kalman Filter formula, taking into consideration both sensors' variance values noise. It's possible to approximate the LiDAR sensor noise as a Gaussian distribution with zero mean μ_L and variance σ_L^2. This value can be extracted from the sensor's datasheet [61], since it provides the user with the information about the device's precision and accuracy, presented in Table 3.1. While the accuracy of the sensor refers to how much the measurement oscillates around the true value, the precision refers to how much the measurements related to the same object are close together. Given this, σ_L can be approximated based on the sensor's precision values.

Distance Accuracy (120mm - 499mm)	±15mm
Distance Accuracy (500mm - 3,500mm)	±5.0%
Distance Precision (120mm - 499mm)	±10mm
Distance Precision (500mm - 3,500mm)	±3.5%

Table 3.1: LDS-01 LiDAR sensor specifications. Source: https://www.robotis.us/360-laser-distance-sensor-lds-01-lidar/ consulted 10/08/2022

Applying the empirical rule, which states that 99,7% of any gaussian observations are contained inside the range of $[\mu - 3\sigma; \mu + 3\sigma]$ it is possible to estimate the noise standard deviation as shown in equation 3.5, where r represents the distance measured by the sensor in milimeters.

$$\begin{cases} \sigma_L = \frac{10}{3}\text{mm}, & \text{if distance between 120mm and 499mm} \\ \sigma_L = \frac{0.035}{3} r \text{ mm}, & \text{if distance between 500mm and 3500mm} \end{cases} \tag{3.5}$$

This value, however, refers to the standard deviation of the noise associated with the range, d, captured by the LiDAR sensor. For this reason, the value needs to be propagated in order to reflect accurately its impact in the x and y coordinates. These coordinates can be calculated from the range and bearing values captured by the sensor, using trigonometric functions. The LiDAR sensor produces an ordered array of distances measured in a 360 degree range, in intervals of 1 degree, captured and transmitted to the ROS topic. Acoording to the sensor's datasheet [61] this interval, called the angular resolution, has no interval of tolerance, so it can be assumed that the θ value has no noise associated. Equations 3.6 and 3.7 indicates how to get the coordinates in cartesian form.

$$x_L = dcos(\theta) \tag{3.6}$$

$$y_L = dsin(\theta) \tag{3.7}$$

With these equations, it's possible to propagate the variance of the range in order to get the variance of x_L and y_L. Since θ is noiseless, the variance of the cartesian coordinates, σ_{xL} and σ_{yL}, is calculated as demonstrated in equations 3.8 and 3.9.

$$\sigma_{xL}^2 = \left(\frac{\partial x_L}{\partial d}\right)^2 \sigma_L^2 \iff \sigma_{xL}^2 = cos(\theta)^2 {\sigma_L}^2 \tag{3.8}$$

$$\sigma_{yL}^2 = \left(\frac{\partial y_L}{\partial d}\right)^2 \sigma_L^2 \iff \sigma_{yL}^2 = sin(\theta)^2 {\sigma_L}^2 \tag{3.9}$$

The same needs to be done for the RGB-D camera coordinates. According to the camera's datasheet, for up to 2 meters, the sensor's spatial noise standard deviation is less than 0.02r, with r

representing the distance measured in meters [58]. For ease of computation, it is assumed that the sensor's noise has a standard deviation σ_C of 0.02r for all values of depth.

Similar to the LiDAR sensor, the camera sensor's variance σ_C^2refers to the depth measurements z, which corresponds to the z coordinate in the original camera frame. For this reason, the x and y camera coordinates in the LiDAR frame, x_C^L and y_C^L respectively, need to be written in relation to this value, to understand how the variance propagates to these variables. This can be seen in equations 3.10 and 3.11, where u represents the horizontal position in the image of the pixel corresponding to the keypoint, c_x is the camera's centre of projection, f_x is the camera's focal length, and dx and dy are the displacements of the frames.

$$x_C^L = z - dx \tag{3.10}$$

$$y_C^L = dy - \frac{u - c_x}{f_x} z \tag{3.11}$$

The variances for each coordinate can then be calculated by propagating σ_C with equations 3.12 and 3.13

$$\sigma_{xC}^2 = \left(\frac{\partial x_C^L}{\partial z}\right)^2 \sigma_C^2 \Longleftrightarrow \sigma_{xC}^2 = \sigma_z^2 \tag{3.12}$$

$$\sigma_{yC}{}^2 = \left(\frac{\partial y_C^L}{\partial z}\right)^2 \sigma_C^2 \Longleftrightarrow \sigma_{yC}^2 = \left(-\frac{u - c_x}{f_x}\right)^2 \sigma_C^2 \tag{3.13}$$

Finally, the fused coordinates are calculated with the Kalman Filter formula, assuming that the camera measurements are the previous measurements, (σ_{prev}^2 and x_{prev} for the variance and measured value, respectively), and the LiDAR values are the current observations (σ_{curr}^2 and x_{curr}), which is translated in equation 3.14, and is applied directly to the x coordinate in equation 3.15.

$$a_{fused} = \frac{\sigma_{prev}^2 a_{curr} + \sigma_{curr}^2 a_{prev}}{\sigma_{prev}^2 + \sigma_{curr}^2} \tag{3.14}$$

$$x_{fused} = \frac{\sigma_{xC}^2 x_L + \sigma_{xL}^2 x_C^L}{\sigma_{xC}^2 + \sigma_{xL}^2} \tag{3.15}$$

With x corresponding to the depth of the camera, this value can be used to deduce the remaining coordinate of the point. Transposing this value back to its original camera frame, by reverting equation 3.2, the fused depth coordinate z_C is calculated and use to compute the remaining x_C and y_C coordinates, now in the camera frame, using Equations 3.16 and 3.17.

$$x_C = \frac{(u - c_x)}{f_x} z_C \tag{3.16}$$

$$y_C = \frac{(v - c_y)}{f_y} z_C \tag{3.17}$$

Where u and v is the keypoint coordinate in the x and y-axis in the RGB image, respectively, f_x and f_y have the same value and represent the focal length of the camera, and c_x and c_y are the camera's centre in pixel coordinates.

Following these calculations, the keypoint is then added to the map, after translating its coordinates back to the world frame, which is already implemented in the ORB-SLAM algorithm.

Although the y coordinate could have been calculated in a similar way as shown in Equation 3.15 using sensor fusion, it was chosen to use only the x_{fused} coordinate and to calculate the remaining two values based on this. This way, the x_C value is not influenced by the propagation of the variance, and both x_C and y_C are calculated in the same way, producing better results.

To better understand the algorithm's flow, Figure 3.12 represents the full algorithm developed, including the logic that was inserted in the ORB SLAM algorithm.

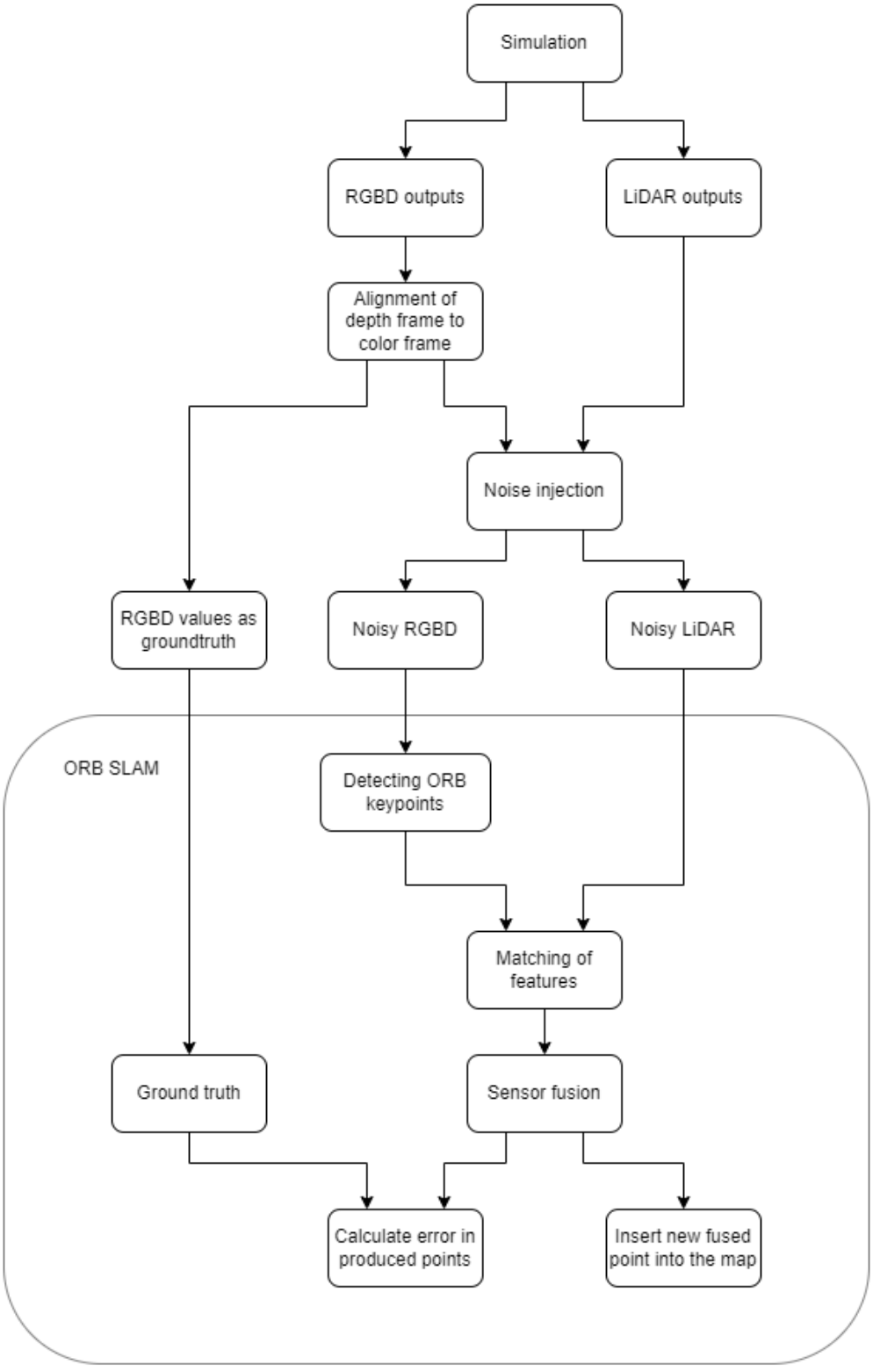

Figure 3.12: Developed solution diagram

Chapter 4

Results

Since the main focus of this thesis is applying sensor fusion to the mapping of an environment, the results aim to portray the accuracy of the generated map when compared to a ground truth map.

To do this, the ORB-SLAM algorithm was altered to subscribe to both the noisy sensor output topics and to a noiseless depth topic, published originally by the D435 sensor, which acted as ground truth to compare the generated values. Since the development was done in a simulated environment, the original topics published by the simulated sensor could be considered as ground truth as there was no noise parameter added to them. After this, the ORB-SLAM algorithm was run and the coordinates of each map point were saved, both with the sensor fusion algorithm and vision-only algorithm, as well as the original map points, which were extracted from the noiseless ground-truth depth map. Then, at the end of each run, whenever the SLAM system was shut down, all the generated points, both the vison-only and sensor fusion outputs, as well as the ground truth values, were saved into a file for further analysis, which will be presented in the following sections of this chapter.

4.1 Obstacles

Before analysing the results, it is worth noting that ORB-SLAM is a complex algorithm by itself, and injecting the sensor-fusion system was troublesome. The place in the code where sensor fusion needed to be added, i.e. before the map points were added to the map, was not clear. There were many details that needed to be optimized for sensor fusion, such as how the SLAM system matches each new observation to build the map and how it calculated the robot's estimated position. The system would often lose track of the map and position, given the reduction in map points generated by the sensor-fusion algorithm. This meant that, in order to fully test the sensor-fusion part, the algorithm could only be tested in small portions of the map, as can be seen in the analysis of the results below. As the main goal was related to the sensor fusion algorithm itself, the focus was shifted to that topic, so there still is room for improvement when it comes to integrating this algorithm with the ORB-SLAM system.

Another obstacle was the characteristics of the sensors used. With the constraint of having to use the LiDAR included in the Turtlebot robot, the vision sensor used had higher accuracy and precision values than the LiDAR sensor. This makes sensor fusion in this system redundant since the goal was to add a more precise sensor to the setup, in order to improve the vision sensor outputs and obtain more accurate map points. As a way to overcome this problem, and since each sensor's noise could be manipulated, the vision sensor's noise variance was altered to be higher than the LiDAR sensor, changing its precision from 2% to 10%.

4.2 Parameter tuning

Some parameters could be tuned in order to have a more reliable system, such as the sensitivity of the ORB detector, the tolerance in the match between the LiDAR and the vision points, and the interval in which the system searched for matches between inputs.

The sensitivity of the ORB detector refers to how easily the algorithm can detect ORB features. If sensitivity is too low, few points are detected, which makes it harder for the algorithm to build a map. On the contrary, a sensitivity that's too high leads to memory and performance issues. In order to have a reliable and meaningful map, the sensitivity was brought up to have more ORB features to calculate matches.

The tolerance and the interval of the matches influence the algorithm's performance in similar ways. The tolerance refers to the difference in the distance from the point to the camera's position between the vision point and the LiDAR point. The interval of the matches relates to the interval in LiDAR points in which the algorithm searches for a match, taking advantage of the ordered points produced by the LiDAR sensor.

If these values are too high, false matches can occur and noise gets inserted into the map. If, however, these values are too low, there is no allowance for the value's noise, and the algorithm may not be able to find a match. The tolerance selected was 100mm and for the interval, a value of 11 was chosen. This means that a match for a determined point was searched for in an interval centred at the point in the LiDAR range array that had the same bearing as the desired vision point, with a full range of 11 measurements. In this way, the closest match between the two inputs is found in an interval that goes five measurements to both sides of the central point. A diagram with a visual representation of the measurements taken by the LiDAR and RGBD sensor in order to better understand how this method works can be seen in Figure 4.1.

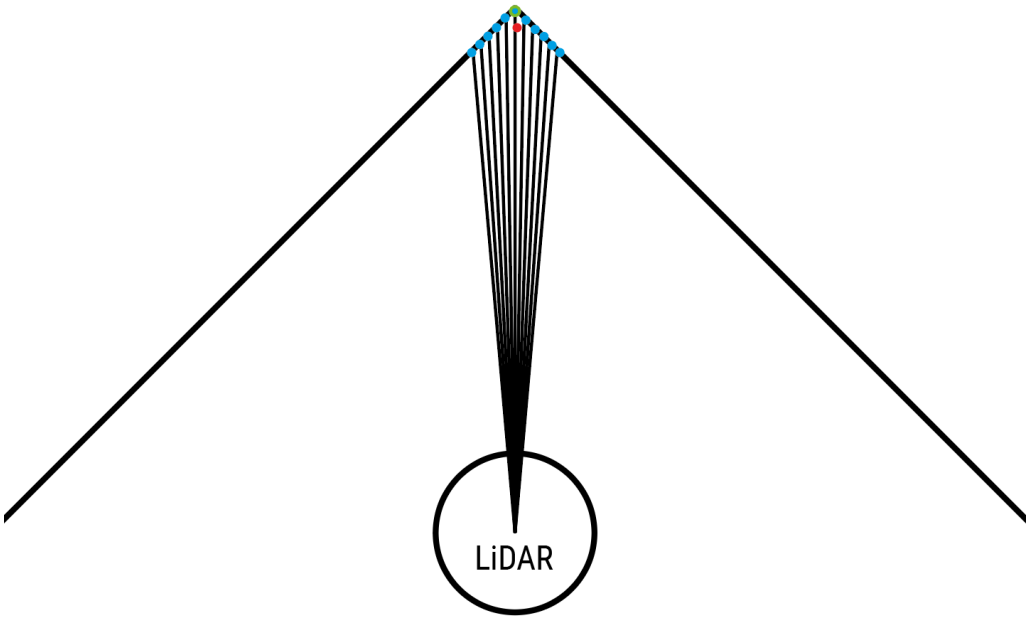

Figure 4.1: Top view of the measurements taken by the LiDAR (blue points) and the feature detected by the camera (red point). In this case, the feature to be detected is the corner, and only 11 of the LiDAR measurements are shown. The red dot represents the RGB-D sensor's estimated point to be matched, and the blue dots represent the 11 measurements taken by the LiDAR sensor, candidates for the feature matching. The dot with the green border is at the centre of this interval, as it is the point with the closest angle to the camera's calculated point.

4.3 Analysis of results

The results of this work can be divided into two sections: the generated map, and the accuracy of the map points.

The generated maps can be seen in Figure 4.2 and in the appendix A. The top right window in each image presents the simulated environment in Gazebo, and the top left is the window opened by the ORB-SLAM, with the generated map illustrated by the red points and the blue representing the path travelled by the robot. The view of the camera at the time each of these images was captured can be seen at the bottom of each image, with the green dots portraying the detected ORB features where a LiDAR match was detected. These ORB features are also represented as the black points in the map, on the top left image.

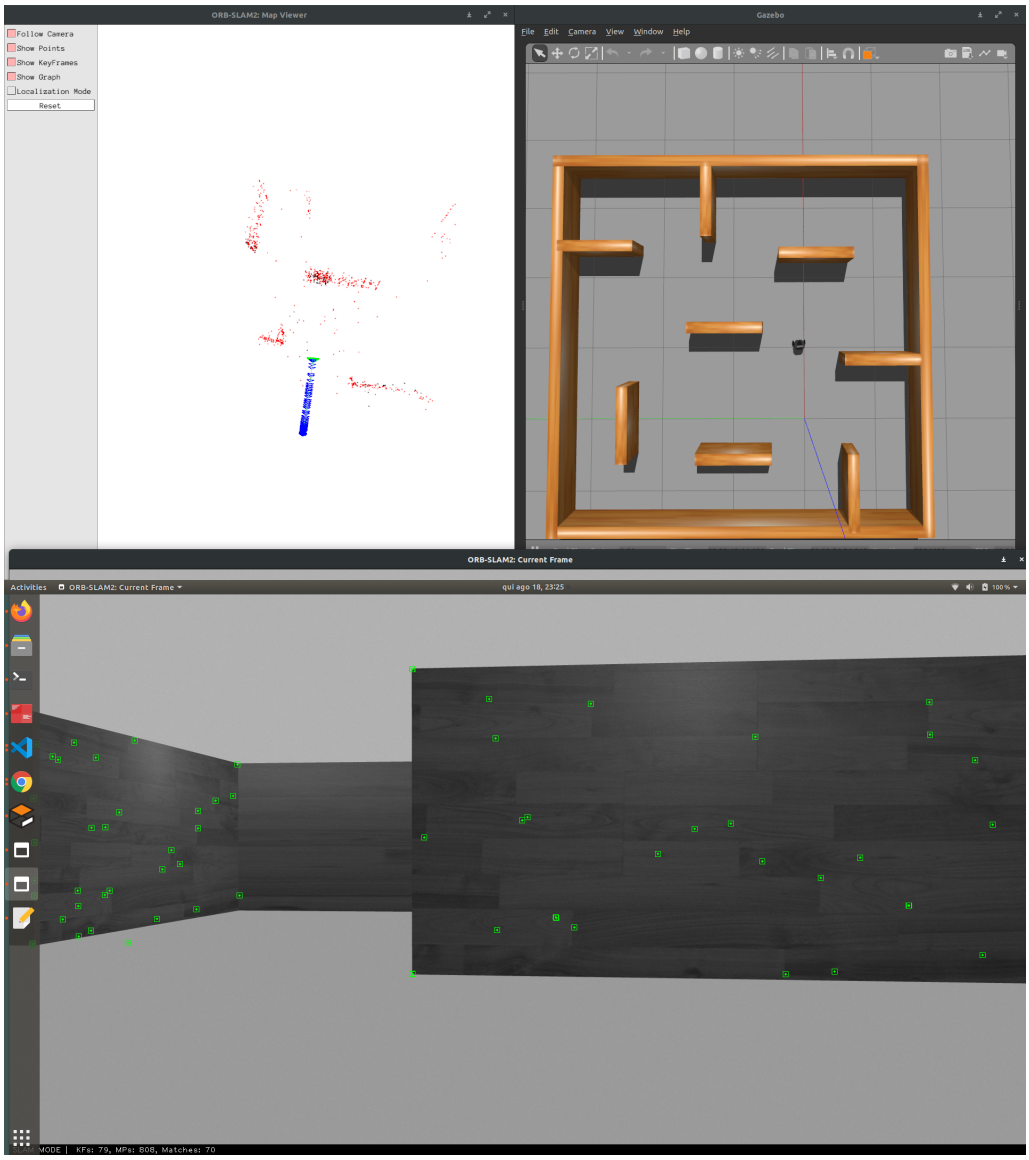

Figure 4.2: Output of the ORB-SLAM algorithm integrated with sensor fusion (top left), simulated environment in real-time (top right) and robot view and detected ORB features (bottom). In the top left image, the red represents the generated map by the ORB-SLAM algorithm, and the blue represents the path travelled by the robot.

Every time the system lost track of its position, it performed a reset of all variables, and the map produced until that point was cleared. This means that the generated map could only be captured in small portions, which is why the images in Figure 4.2 and in the appendix A only show a partial map, with a small trajectory. Even though the created map has some noise, the walls of the environment are visually recognizable, and the estimated path is similar to the actual path followed by the robot, not represented in the image.

As previously mentioned, in order to evaluate the performance of the sensor fusion algorithm, a Comma Separated Values (CSV) file was generated with all the points that were calculated during

each run of the simulation, divided into three groups: sensor-fusion generated points, vision-only points, and the ground-truth. Then, through the use of a spreadsheet program, two measures of performance were calculated, the Mean Absolute Error, MAE, and the Root Mean Square Error, RMSE, for both the original vision-only generated points and the sensor-fusion values, in order to compare both solutions.

The MAE measures the average value of the errors in a set of predictions. It's the average of the absolute differences between the calculated values and the real values, where all the differences have the same weight. The formula for the MAE can be found in Equation 4.1, where n represents the number of measurements taken.

$$MAE = \frac{1}{n}\sum_{i=1}^{N} |(x_{calc} - x_{true})| \tag{4.1}$$

The RMSE also measures the average value of the error, but instead calculates the square root of the average of the squared errors. Its formula can be found in Equation 4.2.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{N} (x_{calc} - x_{true})^2} \tag{4.2}$$

Since both metrics use unsigned values of the errors, whether in absolute or squared form, both are indifferent to the direction of errors. However, as the errors are squared before they are averaged, the RMSE gives a larger weight to larger errors, which is useful when these errors are particularly undesirable.

Out of 3525 calculated points, the values calculated for both metrics can be found in Tables 4.1 and 4.2.

	Sensor-fusion error	Vision only
x-coordinate MAE	0,0078890	0,0115306
y-coordinate MAE	0,0031036	0,0047996
z-coordinate MAE	0,0193770	0,0311657

Table 4.1: Calculated MAE in meters

	Sensor-fusion	Vision-only
x-coordinate RMSE	0,05101	0,05250
y-coordinate RMSE	0,01331	0,01382
z-coordinate RMSE	0,09244	0,09660

Table 4.2: Calculated RMSE in meters

From these values, it is seen that the RMSE value has little improvement, a reduction in between 2 and 4% of this metric, from the vision-only solution to the sensor-fusion proposal, and the MAE has an improvement of over 30% for all coordinates. As this last metric only considers the error itself, it is possible to conclude that the sensor fusion algorithm is successful in minimizing the error in the accuracy of the estimated points of the map. The little change in RMSE values between the two solutions can be explained by how big the errors can get in both versions of the algorithm. As this metric gives more weight to larger errors by squaring them in its equation, it can be concluded that the sensor fusion solution has lower maximum errors than the vision-only algorithm, but performs overall better, by minimizing the average of the errors.

Even though the algorithm loses track of the map frequently, the developed work reaches most of its goals, with the refinement of the accuracy of the map points being accomplished. However, it still leaves space for improvement, particularly when it comes to the integration of the sensor fusion algorithm in the SLAM system. Suggestions on how to improve even further this work can be found in the next section.

Chapter 5

Conclusions and Future Work

5.1 Work Discussion

This thesis aimed to employ multiple sensors to improve the accuracy of maps of unknown environments using mobile robots. This was accomplished using an already existing vision-SLAM algorithm, ORB-SLAM, and Kalman filter to fuse the inputs from two different sensors. The measurements from both sensors to be fused were matched based on the range and bearing of each point, inside a predetermined interval, taking advantage of the order in which the LiDAR sensor measurements are published in its ROS topic. After analyzing the results, it can be concluded that this requirement was satisfied.

The accuracy of the generated map points was improved, but the full map of an unknown environment could not be generated. This is due to a number of factors.

First, ORB-SLAM is a large and complex algorithm, with many different aspects which made it difficult to understand in such a short time. Improvements and optimizations can be made, namely where the sensor fusion algorithm is being applied, and how it can be better integrated with the original algorithm. As the mapping component itself was not the main objective of this thesis, the focus was given to improving the sensor fusion results by themselves, since this algorithm could then be injected into virtually any SLAM algorithm at the map creation step.

The second problem encountered was the specifications of the LiDAR sensor used, which had the capability to only capture points in two dimensions, and had lower accuracy than the vision sensor. This problem was, however, easily overcome as the development was done in a simulated environment

This thesis was able to bring a new and different approach to this problem, by matching ORB features to LiDAR measurements using their ranges and bearings. This solution seems relatively simple, but is still able to provide sound and relevant results, and proves as a relevant contribution to the sensor-fusion SLAM discussion.

5.2 Future work

Based on the produced results, it would be interesting to work on a solution that used a 3D LiDAR sensor instead of a 2D system, since the matching between both inputs would be more accurate, and the fusion between the three coordinates would be direct. However, this solution introduces the problem of matching between the two sensor's outputs, as the range and bearing of the LiDAR sensor would no longer be directly available.

From another standpoint, using the same 2D-3D configuration, the LiDAR sensor used could be one with a higher range, so that further away points could also be detected by this sensor. This would improve the matching between the two detectors, as there would be more points captured by the LiDAR sensor to match with the vision points, therefore producing a more complete map in less amount of time.

Besides this, future work can also include analyzing the ORB-SLAM algorithm and further adapting it to integrate the sensor fusion more deeply into the code, optimizing the SLAM algorithm for it, and testing this system on a physical setup, as opposed to simulation.

Appendix A

ORB-SLAM Outputs

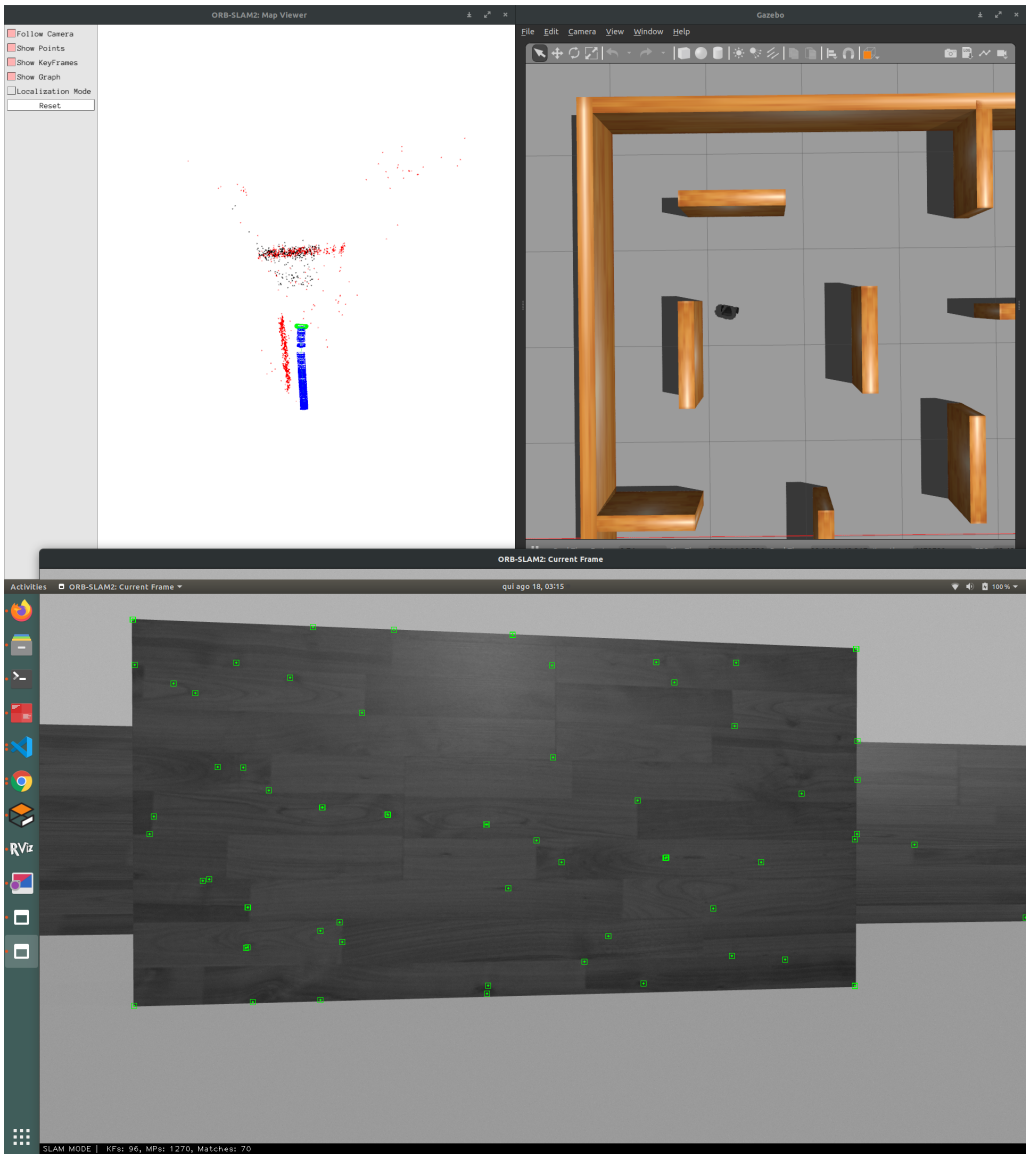

Figure A.1: Output of the ORB-SLAM algorithm integrated with sensor fusion (top left), simulated environment in real-time (top right) and robot view and detected ORB features (bottom). In the top left image, the red represents the generated map by the ORB-SLAM algorithm, and the blue represents the path travelled by the robot.

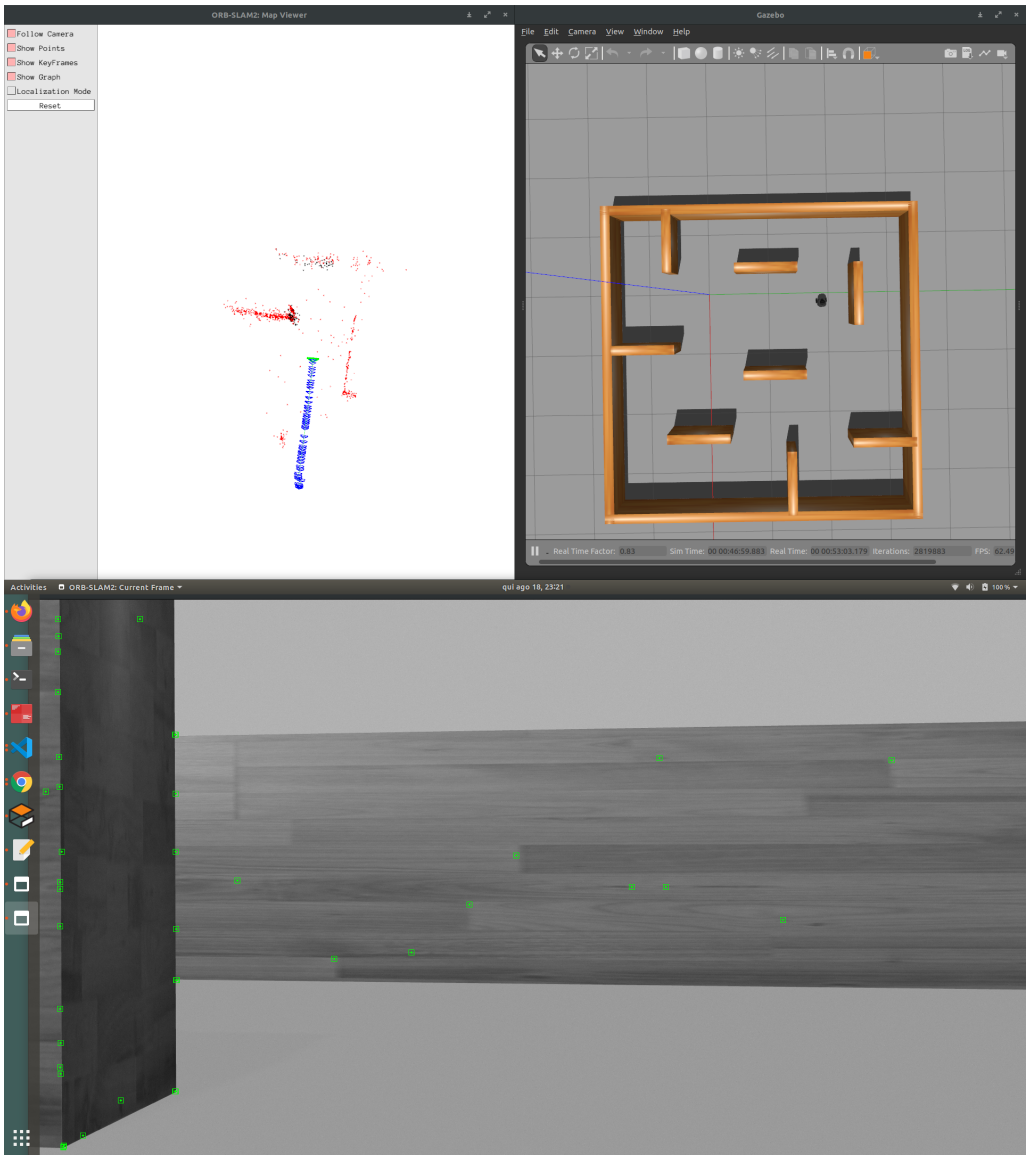

Figure A.2: Output of the ORB-SLAM algorithm integrated with sensor fusion (top left), simulated environment in real-time (top right) and robot view and detected ORB features (bottom). In the top left image, the red represents the generated map by the ORB-SLAM algorithm, and the blue represents the path travelled by the robot.

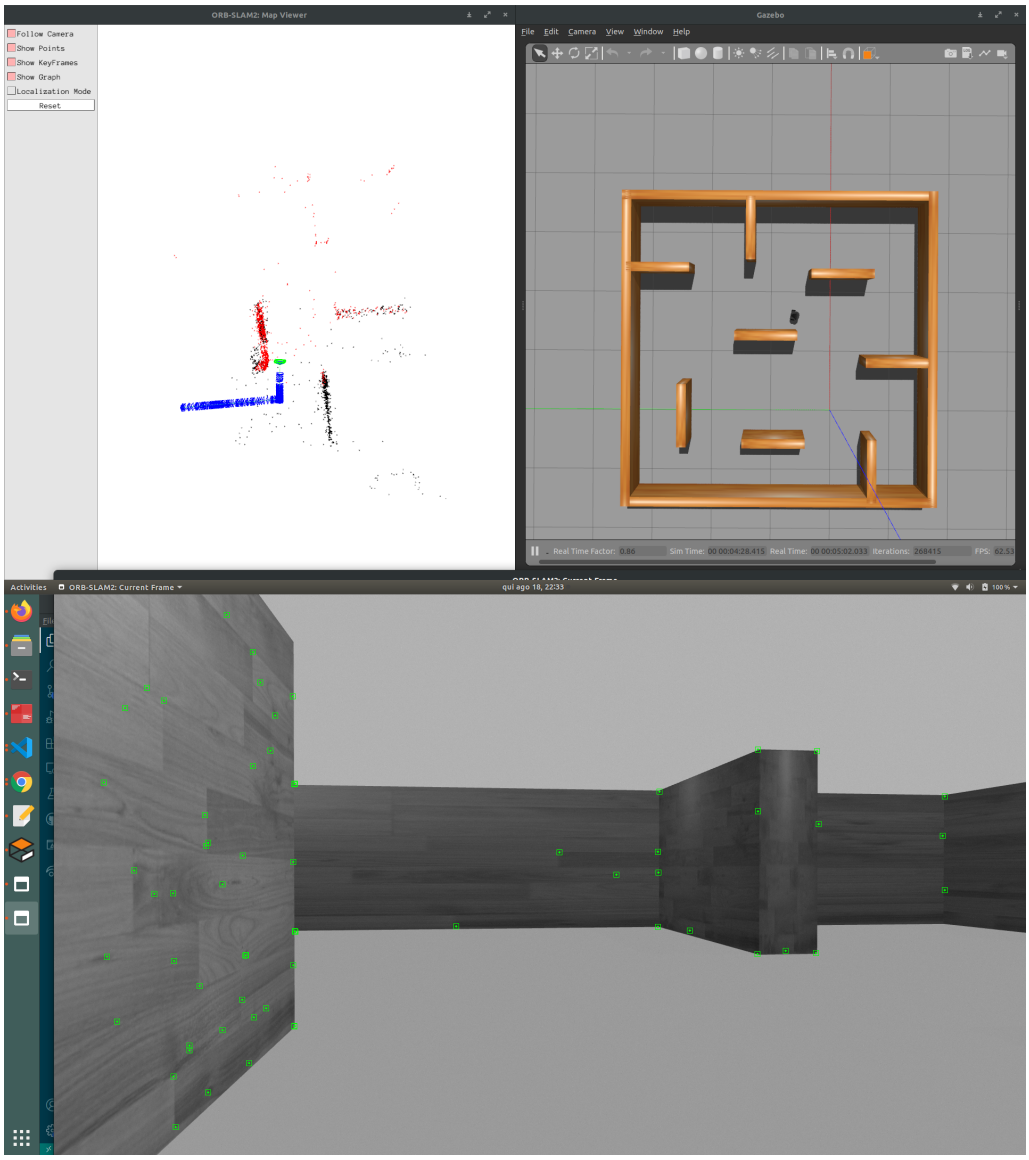

Figure A.3: Output of the ORB-SLAM algorithm integrated with sensor fusion (top left), simulated environment in real-time (top right) and robot view and detected ORB features (bottom). In the top left image, the red represents the generated map by the ORB-SLAM algorithm, and the blue represents the path travelled by the robot.

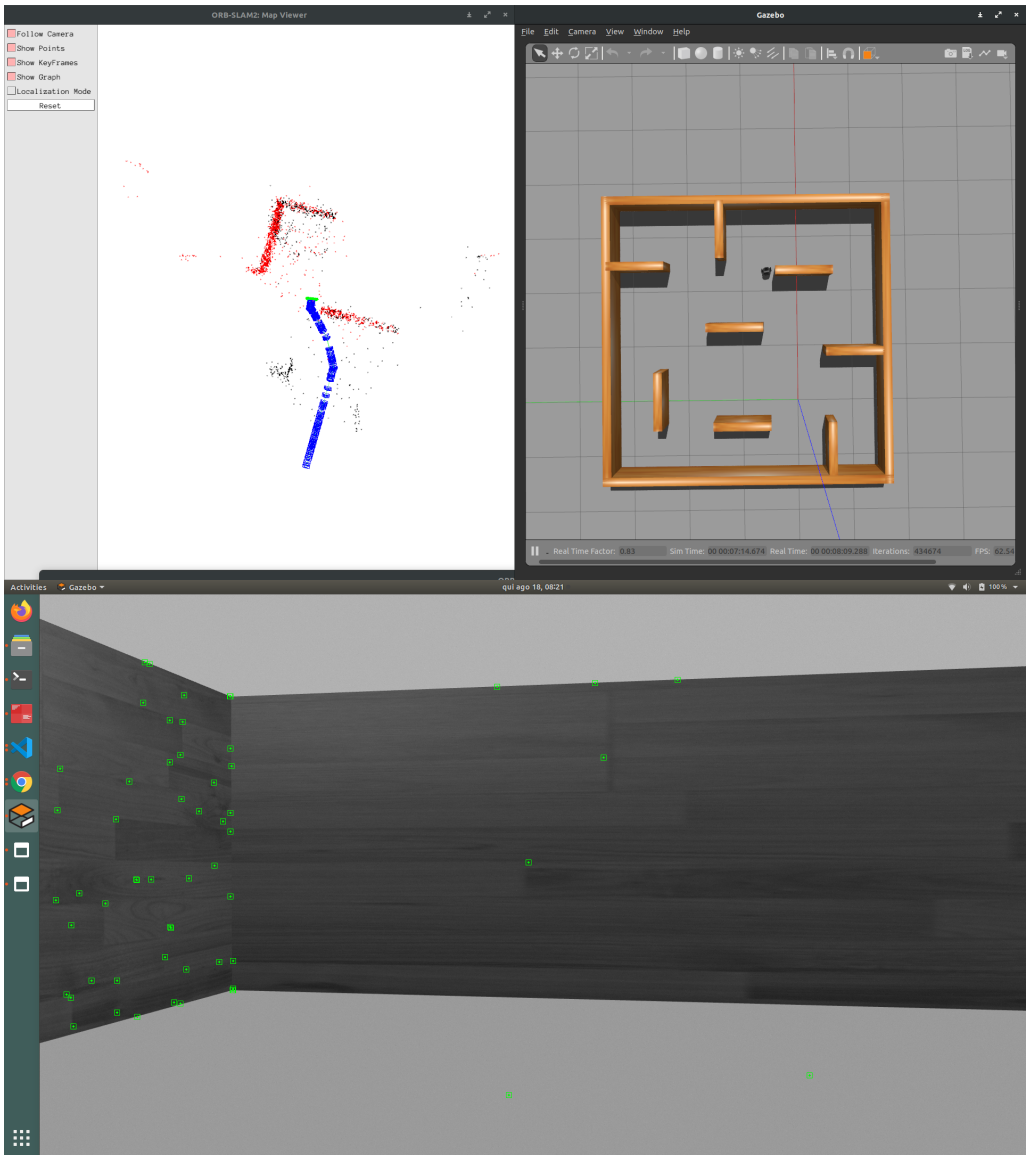

Figure A.4: Output of the ORB-SLAM algorithm integrated with sensor fusion (top left), simulated environment in real-time (top right) and robot view and detected ORB features (bottom). In the top left image, the red represents the generated map by the ORB-SLAM algorithm, and the blue represents the path travelled by the robot.

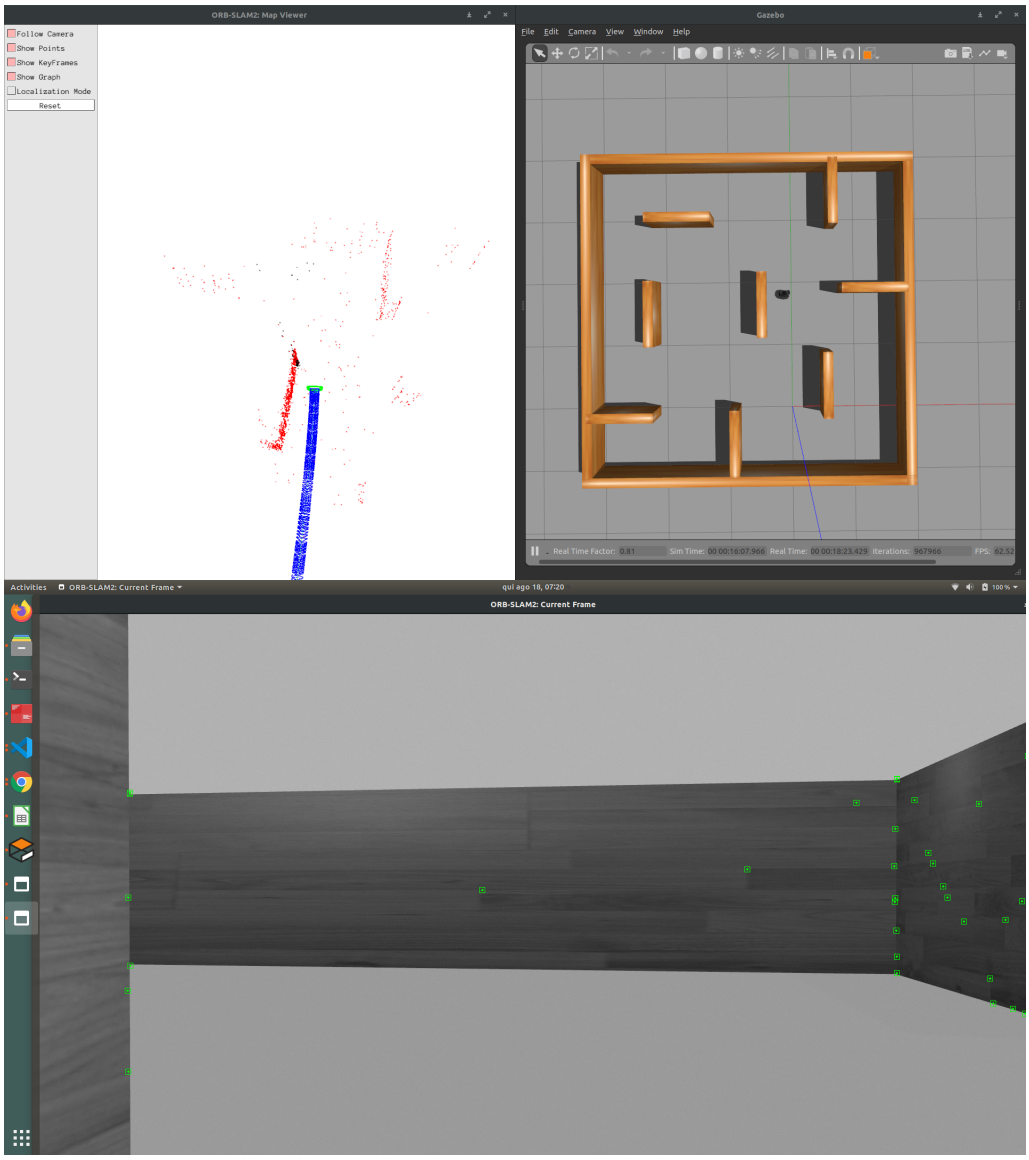

Figure A.5: Output of the ORB-SLAM algorithm integrated with sensor fusion (top left), simulated environment in real-time (top right) and robot view and detected ORB features (bottom). In the top left image, the red represents the generated map by the ORB-SLAM algorithm, and the blue represents the path travelled by the robot.

References

[1] Randall C. Smith and Peter Cheeseman. On the representation and estimation of spatial uncertainty:. 5:56–68, 7 2016. URL: https://journals.sagepub.com/doi/abs/10.1177/027836498600500404, doi:10.1177/027836498600500404.

[2] Kevin Chen, Juan Pablo De Vicente, Gabriel Sepúlveda, Fei Xia, Alvaro Soto, Marynel Vázquez, and Silvio Savarese. A behavioral approach to visual navigation with graph localization networks. https://graphnav.stanford.edu (accessed: Feb. 18, 2022).

[3] Amanda Kooser. "Boston dynamics robot dog spot finally goes on sale for $74,500 - cnet". https://www.cnet.com/news/boston-dynamics-robot-dog-spot-finally-goes-on-sale-for-74500/ (accessed Feb. 18, 2022).

[4] PuduRobotics. "Smart delivery robot-pudu robotics". https://www.pudurobotics.com/product/detail/bellabot (accessed Feb. 18, 2022).

[5] Wilfried Elmenreich. An introduction to sensor fusion flexible real-time control and diagnostic system for application related stress testing. 2002. URL: https://www.researchgate.net/publication/267771481.

[6] Md Nazmuzzaman Khan and Sohel Anwar. Paradox elimination in dempster–shafer combination rule with novel entropy function: Application in decision-level multi-sensor fusion. *Sensors*, 19(21), 2019. URL: https://www.mdpi.com/1424-8220/19/21/4810, doi:10.3390/s19214810.

[7] Belur V. Dasarathy. Sensor fusion potential exploitation-innovative architectures and illustrative applications. *Proceedings of the IEEE*, 85:24–38, 1997. doi:10.1109/5.554206.

[8] Wilfried Elmenreich. *Sensor Fusion in Time-Triggered Systems*. PhD thesis, 01 2002.

[9] Hugh F. Durrant-Whyte. Sensor models and multisensor integration:. *http://dx.doi.org/10.1177/027836498800700608*, 7:97–113, 7 2016. URL: https://journals.sagepub.com/doi/abs/10.1177/027836498800700608, doi:10.1177/027836498800700608.

[10] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82:35–45, 3 1960. URL: https://asmedigitalcollection.asme.org/fluidsengineering/article/82/1/35/397706/A-New-Approach-to-Linear-Filtering-and-Prediction, doi:10.1115/1.3662552.

[11] J.Z. Sasiadek and P. Hartana. Sensor data fusion using kalman filter. In *Proceedings of the Third International Conference on Information Fusion*, volume 2, pages WED5/19–WED5/25 vol.2, 2000. doi:10.1109/IFIC.2000.859866.

[12] H. P. Moravec. Sensor fusion in certainty grids for mobile robots. *Sensor Devices and Systems for Robotics*, 52:253–276, 1989. URL: https://link.springer.com/chapter/10.1007/978-3-642-74567-6_19, doi:10.1007/978-3-642-74567-6_19.

[13] Priyanshu Tripathi, K.S. Nagla, Harvir Singh, and Sudhir Mahajan. Occupancy grid mapping for mobile robot using sensor fusion. In *2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)*, pages 47–51, 2014. doi:10.1109/ICICICT.2014.6781251.

[14] D. Rabijns, W. Van Moer, G. Vandersteen, and J. Schoukens. Using multisines to measure state-of-the-art analog to digital converters. *Conference Record - IEEE Instrumentation and Measurement Technology Conference*, 1:7–12, 2005. doi:10.1109/IMTC.2002.1006807.

[15] Sensor data fusion for a mobile robot using neural networks. *Sensors 2022, Vol. 22, Page 305*, 22:305, 12 2021. URL: https://www.mdpi.com/1424-8220/22/1/305/htmhttps://www.mdpi.com/1424-8220/22/1/305, doi:10.3390/S22010305.

[16] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, 13(2):99–110, 2006. doi:10.1109/MRA.2006.1638022.

[17] Hugh F. Durrant-Whyte. Uncertain geometry in robotics. *IEEE Journal on Robotics and Automation*, 4(1):23–31, 1988. doi:10.1109/56.768.

[18] Randall Smith, Matthew Self, and Peter Cheeseman. *Estimating Uncertain Spatial Relationships in Robotics*, pages 167–193. Springer New York, New York, NY, 1990. URL: https://doi.org/10.1007/978-1-4613-8997-2_14, doi:10.1007/978-1-4613-8997-2_14.

[19] Hugh Durrant-Whyte, David Rye, and Eduardo Nebot. Localization of autonomous guided vehicles. In Georges Giralt and Gerhard Hirzinger, editors, *Robotics Research*, pages 613–625, London, 1996. Springer London.

[20] Randall C. Smith and Peter Cheeseman. On the representation and estimation of spatial uncertainty. *http://dx.doi.org/10.1177/027836498600500404*, 5:56–68, 7 2016. URL: https://journals.sagepub.com/doi/abs/10.1177/027836498600500404, doi:10.1177/027836498600500404.

[21] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. Fastslam: A factored solution to the simultaneous localization and mapping problem. 11 2002.

[22] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23:34–46, 2 2007. doi:10.1109/TRO.2006.889486.

[23] Albert Diosi and Lindsay Kleeman. Fast laser scan matching using polar coordinates:. *http://dx.doi.org/10.1177/0278364907082042*, 26:1125–1153, 7 2016. doi:10.1177/0278364907082042.

[24] Fernando Martin, Rudolph Triebel, Luis Moreno, and Roland Siegwart. Two different tools for three-dimensional mapping: De-based scan matching and feature-based loop detection. *Robotica*, 32:19–41, 1 2014. doi:10.1017/S026357471300060X.

[25] Andrew J. Davison, Ian D. Reid, Nicholas D. Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29:1052–1067, 6 2007. doi:10.1109/TPAMI.2007.1049.

[26] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 225–234, 2007. doi:10.1109/ISMAR.2007.4538852.

[27] Raul Mur-Artal, J. M.M. Montiel, and Juan D. Tardos. Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31:1147–1163, 10 2015. doi:10.1109/TRO.2015.2463671.

[28] Richard A. Newcombe, Steven J. Lovegrove, and Andrew J. Davison. Dtam: Dense tracking and mapping in real-time. In *2011 International Conference on Computer Vision*, pages 2320–2327, 2011. doi:10.1109/ICCV.2011.6126513.

[29] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8690 LNCS:834–849, 2014. URL: https://link.springer.com/chapter/10.1007/978-3-319-10605-2_54, doi:10.1007/978-3-319-10605-2_54.

[30] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. Svo: Fast semi-direct monocular visual odometry. pages 15–22. Institute of Electrical and Electronics Engineers Inc., 9 2014. doi:10.1109/ICRA.2014.6906584.

[31] Rui Wang, Martin Schwörer, and Daniel Cremers. Stereo dso: Large-scale direct sparse visual odometry with stereo cameras, 2017. URL: https://arxiv.org/abs/1708.07878, doi:10.48550/ARXIV.1708.07878.

[32] Fengchi Sun, Yuan Zhou, Chao Li, and Yalou Huang. Research on active slam with fusion of monocular vision and laser range data. pages 6550–6554, 2010. doi:10.1109/WCICA.2010.5554412.

[33] Yinglei Xu, Yongsheng Ou, and Tiantian Xu. Slam of robot based on the fusion of vision and lidar. *2018 IEEE International Conference on Cyborg and Bionic Systems, CBS 2018*, pages 121–126, 1 2019. doi:10.1109/CBS.2018.8612212.

[34] Ji Zhang and Sanjiv Singh. Visual-lidar odometry and mapping: Low-drift, robust, and fast. *Proceedings - IEEE International Conference on Robotics and Automation*, 2015-June:2174–2181, 6 2015. doi:10.1109/ICRA.2015.7139486.

[35] Youngwoo Seo and Chih Chung Chou. A tight coupling of vision-lidar measurements for an effective odometry. *IEEE Intelligent Vehicles Symposium, Proceedings*, 2019-June:1118–1123, 6 2019. doi:10.1109/IVS.2019.8814164.

[36] Guolai Jiang, Lei Yin, Shaokun Jin, Chaoran Tian, Xinbo Ma, and Yongsheng Ou. A simultaneous localization and mapping (slam) framework for 2.5d map building based on low-cost lidar and vision fusion. *Applied Sciences 2019, Vol. 9, Page 2105*, 9:2105, 5 2019. URL: https://www.mdpi.com/2076-3417/9/10/2105/htmhttps://www.mdpi.com/2076-3417/9/10/2105, doi:10.3390/APP9102105.

[37] Ranjith Unnikrishnan and Martial Hebert. Fast extrinsic calibration of a laser rangefinder to a camera. Technical Report CMU-RI-TR-05-09, Carnegie Mellon University, Pittsburgh, PA, July 2005.

[38] Abdallah Kassir and Thierry Peynot. Reliable automatic camera-laser calibration. In G Wyeth and B Upcroft, editors, *Proceedings of the 2010 Australasian Conference on Robotics & Automation*, pages 1–10. Australian Robotics and Automation Association (ARAA), Australia, 2010. URL: https://eprints.qut.edu.au/67618/.

[39] Kihong Park, Seungryong Kim, and Kwanghoon Sohn. High-precision depth estimation using uncalibrated lidar and stereo fusion. *IEEE Transactions on Intelligent Transportation Systems*, 21:321–335, 1 2020. doi:10.1109/TITS.2019.2891788.

[40] Ayodeji Olalekan Salau and Shruti Jain. Feature extraction: A survey of the types, techniques, applications. In *2019 International Conference on Signal Processing and Communication (ICSC)*, pages 158–164, 2019. doi:10.1109/ICSC45622.2019.8938371.

[41] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision 2004 60:2*, 60:91–110, 11 2004. URL: https://link.springer.com/article/10.1023/B:VISI.0000029664.99615.94, doi:10.1023/B:VISI.0000029664.99615.94.

[42] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. *Proceedings - 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005*, I:886–893, 2005. doi:10.1109/CVPR.2005.177.

[43] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3951 LNCS:430–443, 2006. URL: https://link.springer.com/chapter/10.1007/11744023_34, doi:10.1007/11744023_34/COVER.

[44] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3951 LNCS:404–417, 2006. URL: https://link.springer.com/chapter/10.1007/11744023_32, doi:10.1007/11744023_32/COVER.

[45] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 110:346–359, 6 2008. doi:10.1016/J.CVIU.2007.09.014.

[46] Michael Calonder, Vincent Lepetit, Mustafa Özuysal, Tomasz Trzcinski, Christoph Strecha, and Pascal Fua. Brief: Computing a local binary descriptor very fast. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34:1281–1298, 2012. doi:10.1109/TPAMI.2011.222.

[47] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571, 2011. doi:10.1109/ICCV.2011.6126544.

[48] Stefan Leutenegger, Margarita Chli, and Roland Y. Siegwart. Brisk: Binary robust invariant scalable keypoints. In *2011 International Conference on Computer Vision*, pages 2548–2555, 2011. doi:10.1109/ICCV.2011.6126542.

[49] Alexandre Alahi, Raphael Ortiz, and Pierre Vandergheynst. Freak: Fast retina keypoint. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 510–517, 2012. doi:10.1109/CVPR.2012.6247715.

[50] Xufeng Han, Thomas Leung, Yangqing Jia, Rahul Sukthankar, and Alexander C. Berg. Matchnet: Unifying feature and metric learning for patch-based matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[51] Kwang Moo Yi, Eduard Trulls, Vincent Lepetit, and Pascal Fua. Lift: Learned invariant feature transform. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9910 LNCS:467–483, 2016. URL: https://link.springer.com/chapter/10.1007/978-3-319-46466-4_28, doi:10.1007/978-3-319-46466-4_28/TABLES/3.

[52] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, Jose Neira, Ian Reid, and John J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32:1309–1332, 12 2016. URL: http://arxiv.org/abs/1606.05830http://dx.doi.org/10.1109/TRO.2016.2624754, doi:10.1109/TRO.2016.2624754.

[53] Robert Bassett and Julio Deride. Maximum a posteriori estimators as a limit of bayes estimators. *Mathematical Programming*, 174:129–144, 3 2019. URL: https://link.springer.com/article/10.1007/s10107-018-1241-0, doi:10.1007/S10107-018-1241-0/FIGURES/1.

[54] Søren Riisgaard and Morten Rufus Blas. Slam for dummies a tutorial approach to simultaneous localization and mapping by the 'dummies', 2005.

[55] Visual slam: The basics | kudan global. https://www.kudan.io/archives/433 (cccessed on: Feb. 19, 2022).

[56] Shoubin Chen, Baoding Zhou, Changhui Jiang, Weixing Xue, and Qingquan Li. A lidar/visual slam backend with loop closure detection and graph optimization. *Remote Sensing 2021, Vol. 13, Page 2720*, 13:2720, 7 2021. URL: https://www.mdpi.com/2072-4292/13/14/2720/htmhttps://www.mdpi.com/2072-4292/13/14/2720, doi:10.3390/RS13142720.

[57] Maria Isabel Ribeiro. Kalman and extended kalman filters: Concept, derivation and properties, 2004.

[58] Intel®. *Intel® RealSense™ Product Family D400 Series Datasheet*, April 2022. Rev. 013.

[59] Raúl Mur-Artal and Juan D. Tardós. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017. doi:10.1109/TRO.2017.2705103.

[60] Bill Triggs, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. Bundle adjustment – a modern synthesis. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*,

1883:298–372, 2000. URL: https://link.springer.com/chapter/10.1007/3-540-44480-7_21, doi:10.1007/3-540-44480-7_21/COVER.

[61] HL Data Storage. *LDS1.5 SPECIFICATIONS*, February 2014. URL: https://emanual.robotis.com/assets/docs/LDS_Basic_Specification.pdf.