Multi-sensory ORB_SLAM2 on a synthetic dataset

submitted by

Zekui Xue

for the degree of Master of Science

of the

University of Bath

Department of Computer Science

September 2022

Abstract

The development of self-driving cars has increased people's emphasis on the simultaneous localization and mapping (SLAM) problem. When self-driving cars are driving on the road, they have to detect the real-time map of the road to decide the driving route. However, except for the self-driving car, the SLAM problem exists in various scenes, such as blind guidance in the indoor environment. Under this condition, different SLAM algorithms have been developed to solve problems. Usually, the SLAM algorithms identify the critical features of the environment to reconstruct the maps of the environment and then determine the moving trajectory. However, it is difficult to identify the features of the textureless environment because cameras cannot precisely capture the objects in the textureless environment. Due to this, this paper studies the SLAM problems in the textureless environment. After reviewing various scholars' findings related to the SLAM problems in the textureless environment, this paper makes an empirical analysis using the ORB-SLAM2 algorithm to solve a SLAM problem in an indoor textureless environment. The results of the empirical research show that the ORB-SLAM2 algorithm helps solve the SLAM problem in an indoor textureless environment. First, when using the original ORB-SLAM2 algorithm, it can generally identify the positions of the stereo keyframes and RGB-D keyframes, as well as the key points to construct the general map of the textureless environment. However, the accuracy of the original ORB-SLAM2 algorithm is not too high when it has not removed the influence of the dynamic points. But this problem can be solved by integrating the original ORB-SLAM2 algorithm with other algorithms, such as the optical flow algorithm and the multi-view geometry algorithm, to reduce the influence of the dynamic points on the identification of the positions and trajectories. Especially the accuracy of the combination of the original ORB-SLAM2 algorithm and the multi-view geometry algorithm is much higher than the accuracy of the combination of the original ORB-SLAM2 algorithm and the optical flow algorithm. Overall, the results of the above empirical analysis illustrate that the ORB-SLAM2 algorithm is available to solve the SLAM problem in the indoor textureless environment when the stereo cameras shoot only visual images and RGB-D cameras are used.

Keywords— SLAM problem, Textureless environment, ORB-SLAM2 algorithm, Stereo camera, RGB-D camera

Contents

Chapter 1

Introduction

With the development of shooting technologies, a variety of pictures are shot by people to record information. Under this condition, people have to use the information to get information. Sometimes, people are forced to get information about positions when they have several images recording the report of the surrounding scenes. Under this condition, people have developed a variety of tools to translate the position information recorded in photos, and visual simultaneous localization and mapping (SLAM) are one of the tools used by people to identify locations through analyzing information recorded in the images of surrounded scenes [1]. This dissertation aims to study the implementation of a multi-sensory SLAM algorithm, identifying its effectiveness in training data and detecting information from images to identify scenes' features. In this chapter, the background of the study is justified, and the research objectives and questions are identified. Hence, this chapter mainly contains four sections, background, research objectives and questions, justification of the values of this research, and an outline of the framework of the forward contents.

1.1 Background

The development of technologies has significantly changed people's lives, and modern technological products are gradually replacing traditional products. For example, autopilot has become popular in recent years. Many companies invest in this area to develop this new cars, replacing classic vehicles, such as Tesla and Huawei [2][3]. When these self-driving cars are driving on the road, one of the most significant events is to detect the surrounding environments and the maps for these cars to make decisions [4]. Under this condition, simultaneous localization and mapping (SLAM) are essential techniques these self-driving cars use to identify the surrounding maps [5]. According to [6], the SLAM is an algorithm used to perceive the environment and estimate the position of an object simultaneously by using sensors. It is usually used in a robot to identify the state of the robot, orienting it in different environments, because it helps the robot operators to determine the robot's current situation and then set the path for it [5]. [5] claim that robots can reduce errors with the help of the SLAM algorithm when the operators use this technique to navigate the orientations of the robot. Except for being used in navigating robots of cars by detecting maps for them, the SLAM system is also used in many other fields, for instance, detecting maps for mobile robots [7], identifying the accuracies of existing maps in indoor environments [8], navigation for airborne [9][10], and blind guidance [11]. According to [6], with the need for maps increasing, the implementation of the SLAM system will become more popular in the future because of its efficiency in identifying environments. [6] argue that the SLAM system helps deal with problems in all scenarios related to maps, whether maps exist or not. Due to this, the use of the SLAM system in the future will become more popular when more scenarios need maps. Considering this perspective, researching the development and implementation of the SLAM system is worthy.

In the past two decades, the SLAM technique has experienced fast development, and one of the main changes is its implementation in the 3D field. According to [12], two-dimensional (2D) and three-dimensional (3D) scenarios are two categories of information usually used to identify the position of an object in an environment. Compared to 2D scenarios, which have only considered the situations of an object in the plane, the 3D scenarios reflect more accurate information about the object's position in space [12][13]. According to [14], both the 2D and 3D scenarios help identify the surrounding maps to enable operators to determine the positions of the target objects. Furthermore, compared to the 2D sensors, the 3D sensors can reflect more information about things by detecting more points, which are more valuable to let the operators identify the shape of the objects, such as walls, shelves, and tables, in the target environment, and then make more accurate orders to orient the paths of robots to reduce errors [14]. In recent years, the implementation of the 2D and 3D SLAM techniques has become popular to reduce the errors in detecting the information of maps, particularly in indoor scenarios [15]. Furthermore, to get more information on the environment to construct the 3D maps, the multi-sensory approach is widely used to improve the efficiency of the SLAM technique [16]. Under this condition, the research on the multi-sensory SLAM approach is increasing when it can significantly improve the accuracy of the identification of maps and increase the implementation of the SLAM approach in more scenarios. Because of this, this study focuses on implementing the multi-sensory SLAM approach in detecting the information of positions in different scenes.

Although previous scholars have identified the effectiveness of the SLAM approach in detecting the information of maps, it still faces many problems in some aspects, for instance, when using the cameras to shoot images for textureless regions [17]. [17] claim that the troubles in detecting the information in the textureless regions significantly impact the security of micro aerial vehicles (MAVs) because the operators may face risks in seeing the information in these regions and then make wrong orders. This problem has impacted the effectiveness of MAV navigation. Besides MAVs, this problem also affects the security of other autonomous vehicles because it increases the difficulties in tracking paths [18]. Because of this, more researchers have focused on this problem in recent years, trying to solve the issues related to tracking the textureless regions using the SLAM approach. This study also tries to research how to use a multi-sensory SLAM approach to detect information in a textureless area.

1.2 Research objectives and research questions

The main objective of this dissertation is to investigate the effectiveness of a multi-sensory SLAM algorithm in detecting information in a textureless region. To achieve this goal, this study develops a multi-sensory SLAM algorithm and then implements it to train a synthetic dataset obtained from RGB images. Aligning with this research objective, the following questions are researched in this study.

First, what is the mechanism of the SLAM algorithm to detect information from images? Second, what are the difficulties in detecting data from the textureless regions? Third, how do we solve the problems related to seeing information from the textureless areas by the multi-sensory SLAM algorithm?

1.3 Justification of the values of this research

The values of this study can be considered from theoretical and practical perspectives. Believing the theoretical perspective can enrich the theories related to the SLAM algorithm. Currently, the SLAM algorithm develops quickly because more scenes need maps for navigation. However, tracking objects in the textureless regions is still tricky, and researchers have done various studies in this field to solve the problems related to detecting information from the images of the textureless areas. Previous scholars have tried many approaches to solve this problem, for example, combining the SLAM algorithm with a deep-learning method [19] and building multi-view stereo to change the scenes of the textureless regions to improve the information on the images detected by the traditional SLAM algorithm [20]. However, these approaches may increase the tasks of the SLAM algorithm to increase the difficulties of using this method in detecting maps, and they cannot be popularly used in different scenes. In this case, researching a multi-sensory SLAM algorithm can increase the solutions to the problems related to detecting the information about the textureless regions. At the same time, compared to the previous approaches, the multi-sensory SLAM algorithm does not need the operators to take more operations, for example, integrating with a deep-learning algorithm. In contrast, it focuses on getting information about the scenes from different perspectives with more sensors.

Considering the practical perspective, the findings of this study can increase the implementation of the SLAM algorithm in more scenes, at least, the textureless regions. When these problems related to tracking the textureless areas are solved, the performance of the SLAM algorithm can be expanded to a more extensive range, not only in autonomous vehicles. For example, the SLAM algorithm can be implemented in minimally invasive surgery scenes while these scenes involve a variety of textureless regions [21]. Although the sets researched in this study are not so complex, other researchers can refer to the multi-sensory SLAM algorithm developed to build their algorithms to deal with the problems related to textureless regions in different scenes.

1.4 An outline of the framework of the forward contents

The following contents contain four chapters. Chapter 2 is a literature review which reviews previous scholars' academic articles related to SLAM and discusses their findings to identify the development history of the SLAM algorithm and its implementation in different scenes. Mainly, this chapter discusses the mechanism of the traditional SLAM algorithm and the difficulties related to detecting information in textureless regions. Chapter 3 explains the research method, which constructs the multi-sensory SLAM algorithm used in this study. Chapter 4 is an empirical analysis using the multi-sensory SLAM algorithm to train a synthetic dataset to detect the information contained in these scenes. At the same time, the effectiveness of the multi-sensory SLAM algorithm is justified in the empirical analysis. Finally, chapter 5 is the study's conclusion, summarising the above research findings and discussing the future research orientation.

Chapter 2

Literature review

This chapter reviews previous scholars' studies related to SLAM, justifying their findings on implementing this method in localization and mapping. This chapter focuses on the following aspects, including the mechanism of the SLAM algorithm, its development history, and the current findings related to it in detecting map information about different scenes. Notably, the development of the multi-sensory SLAM algorithm is emphasized in the process of discussing the development history of the SLAM algorithm. Meanwhile, this chapter also discusses the difficulties of detecting information about the textureless regions, justifying the possible solutions to these problems with the multi-sensory SLAM algorithm.

2.1 The concept of the SLAM problem

The SLAM problem is relatively new and always relevant to the use of mobile robots [22][23][24]. When a mobile robot goes to a new and unknown environment, it has to quickly determine its map to ensure its function as an autonomous tool [22]. At the same time, to get used to the new environment, the elements related to locations should be added to the map of the mobile robot to ensure its functions because these environmental elements determine the paths of the mobile robot [25]. In the process of identifying the environmental factors, such as landmarks, obstacles and walls, related to the locations, cameras, sensors, and other visual tools are applied to collect information, and the SLAM systems use the information collected by these visual tools to interpret the unknown environment to improve the map [25][26]. However, because the situations in unfamiliar environments are complex, and the information collected by sensors and cameras is diverse from different perspectives, researchers are required to develop SLAM algorithms to analyze the various data to solve the SLAM problems [27]. Based on the above explanation, the SLAM problem aims to solve the problem related to the detection of the maps in an unknown environment to help robots to design moving routes. However, because the situations of the unfamiliar territory are diverse and complex, the SLAM problems are also various, and people have to develop different algorithms according to the actual cases of the SLAM problems. Therefore, the requirements for solving SLAM problems have promoted the development of SLAM algorithms.

2.2 The development history of the SLAM algorithm

Although the research on the SLAM algorithm has been hot in recent years, it has not been developed for many years. According to [28], the significant development of the SLAM algorithm happened in the past 25 years, and various scholars flow into this field to solve the problems related to map building. Meannwhile, [28] argue that different scholars develop different SLAM algorithms since the scenarios related to environments and maps are permanently changed, and scholars have to create unique algorithms to solve particular cases related to map building. For example, [28] point out that the SLAM algorithms have been used in various scenarios, such as outdoor, indoor, air, underwater, and so on. Therefore, the findings of [28] have illustrated that the SLAM algorithms have been used in identifying the maps of different environments for an extended period.

Except for [28], a variety of scholars have researched the development history of the SLAM algorithms in different fields to solve various problems, not only environmental map identification. For example, [29] have reviewed the development history of the SLAM algorithms for the scenarios using a single robot (labelled as a single-robot SLAM algorithm). According to [29], the single-robot SLAM algorithms can be divided into four categories when considering the perspective of the ways to represent maps and three categories when considering the view of the ways to process data. In detail, the four map presentation single-robot SLAM algorithms are polygon-based SLAM, appearance-based SLAM, view-based SLAM, and feature-based SLAM. In contrast, the three data-process single-robot SLAM algorithms are AI (artificial intelligence) SLAM, smoothing SLAM, and filtering SLAM [29]. Although these single-robot SLAM algorithms are divided into different categories according to various criteria, it does not mean that the features of these SLAM algorithms are strictly separated. For instance, the EKF-SLAM algorithm is a filtering SLAM that can apply different map representation methods to control data [29][30]. For example, [30] used the feature-based map representation in the filtering data processing procedure, [31] used the view-based map representation method in the filtering data processing procedure, and [32] used the appearance-based map representation method in filtering data processing procedure. [33] have used the polygon-based map representation in the filtering data processing procedure. The above scholars' findings reveal that the SLAM algorithms are a mix of map representation and data processing methods. Overall, these SLAM algorithms use different features of the pictures of these maps to identify the environment. They also depend on the cameras' settings to detect real-time information in unknown scenes. In addition, these single-robot SLAM algorithms were developed and applied by different scholars in different stages of the development of the history of the SLAM algorithm.

2.2.1 The feature-based SLAM algorithms

The feature-based SLAM is the easiest one to solve the SLAM problem. In the 1990s, various scholars, such as [30], developed this SLAM algorithm. First, [30] developed the primary mechanism of the feature-based SLAM algorithm, justifying how to identify the relationship between different features in the space. Then, [34][35] optimized the feature-based SLAM algorithm by optimizing the calculation steps, particularly the data filtering procedures. They also used the optimized feature-based SLAM algorithm to test the map of an outdoor environment. Following [30] and [34][35], other scholars have also solved the limitations of this type of SLAM algorithm because its application depends on the identification of the features in the target environment. For example, [36] has proposed the idea of using LIDAR technology to collect information about the outdoor environments to identify more features, [37] have offered the idea of adding sensors to collect data from the environment to identify more features, [38] have proposed the idea of using synthetic aperture sonar to collect information of critical components when dealing with the underwater scenarios, [39] has also justified the effectiveness of using sonar to identify the features of environments in the process of testing aquatic systems. [40] have researched the feasibility of using sonar to identify the components of under-constrained landmarks. The above scholars tried to solve the limitation of the feature-based SLAM algorithm by adding more tools to collect image information to identify feature information about the target environment.

Except for this, some scholars have also researched other ways to improve the application of feature-based SLAM. One way is to improve the procedures of the algorithm. For example, [41] have proposed new ways to optimize the information filtering procedures, [42] have improved the algorithm to identify the feature extraction to increase the accuracy of determining the positions of robots in the indoor environment, [43] have proposed the methods to divide a large map into small sub-maps, using the feature-based SLAM to deal with these sub-maps and then optimize the processes to join different sub-maps, [44] have studied algorithms to distinguish features to reduce the influence of outliers, increasing the accuracy of the feature-based SLAM algorithm, and [45] have developed a method to identify the features in a dynamic way, rather than treating these features as static ones. These scholars have tested their SLAM algorithms, illustrating their efficiency in identifying features in different scenarios and promoting the development of the feature-based SLAM algorithm.

2.2.2 The view-based SLAM algorithms

The view-based SLAM algorithm is also known as the location-based SLAM algorithm, which is the second type of SLAM algorithm after the development of the feature-based SLAM, developed in the middle and late 1990s [46][47]. However, unlike the feature-based SLAM algorithms that identify the environments by distinguishing the key features, the view-based SLAM algorithms do not determine the critical components of the map representation [29][48]. In contrast, it scans the whole map and then uses the matching algorithms to identify the locations of the robot. Therefore, the view-based SLAM algorithm is from the view perspective of the robot rather than from the environmental context. Therefore, it does not require calculation algorithms to extract the information on critical features. As a result, it also does not face the shortcomings of losing data on the map faced by the feature-based SLAM algorithms [29]. Because of this, a variety of scholars, such as [49], [50], [51], and [52] argue that the view-based SLAM algorithms are better than feature-based SLAM algorithms in identifying map information, particularly in the process of dealing with the dynamic environmental contexts.

Although the view-based SLAM algorithms are thought to be better than the feature-based algorithms, their application also faces various constraints. First, to scan information, the view-based SLAM algorithms need a variety of scanners, such as scan laser rangers, to review the environmental data and then match the scanned objects with a known map [51]. Second, suppose the operators of the view-based SLAM algorithms want to check all the collected environmental information with the known maps. In that case, the calculation will become very heavy, wasting time and increasing the burden of analysis [53]. In this case, some scholars have proposed methods to improve the efficiency of the view-based SLAM algorithms. For example, [54] have proposed a strategy to enhance the view-based SLAM algorithms by integrating them with the feature-based SLAM algorithms, focusing on matching the information with key features rather than the whole known maps, [55] propose a method of using omnidirectional images and reliable prior inputs to reduce the heavy matching calculation.

Unlike the development of feature-based SLAM algorithms, scholars are still exploring new strategies to increase the efficiency of using the view-based SLAM algorithms in solving SLAM problems in different environmental scenarios.

2.2.3 The appearance-based SLAM algorithms

The appearance-based SLAM algorithms are specially developed to deal with loop closure problems [56][57][58][59]. Unlike the feature-based and view-based SLAM algorithms, the appearance-based SLAM was developed after 2000. It was based on the feature-based and view-based SLAM algorithms because appearance contained the concept of view and feature [56]. In other words, the appearance-based SLAM algorithms treat critical features and the view of the whole map as objects in the calculation. The selection of features and views depends on the actual environment. Such operations can be used in appearance-based algorithms because it is applied to deal with loop closure problems. The algorithm can use the previously collected dataset as existing maps for previous scenes, comparing them with the newly organized locations in the same environment to judge the new situations of the domain [56].

Compared to the feature-based SLAM algorithms and the view-based SLAM algorithms, the application of the appearance-based SLAM algorithms faces various problems. First of all, it is primarily developed for solving loop closure problems. Hence it is difficult to use this type of SLAM algorithm in open environments. Second, it has inherited its issues because it has integrated feature-based and view-based SLAM algorithms. Third, because the appearance-based SLAM algorithms can compare the current views with previous views, the algorithms need many memories to store the pictures mentioned above, which can directly increase the burden on the computing facilities [29]. Because of this, recent scholars have focused on improving the search algorithms for the views and features, and [60] have proposed a method of using a graph-based nearest neighbour search to solve this problem. In addition, [61] have offered a technique called IBuILD to reduce the searching jobs of the new scenes from the previous stages, which have already been stored in the data pool of the algorithm. Therefore, the research on applying appearance-based SLAM algorithms has been increasing in recent years.

2.2.4 The polygon-based SLAM algorithms

The above three map representation algorithms are mainly for solving 2D map problems, but the polygon-based SLAM algorithms are used primarily to solve 3D map problems [29][62][63]. It is also developed in the early 2000s, and in recent years, scholars have also developed polygon-based SLAM algorithms for 2D maps [64]. To build 3D maps, the polygon-based SLAM algorithms may integrate the feature-based and view-based algorithms in creating maps for different planes [33][65]. Compared to other map representation SLAM algorithms, the development of the polygon-based SLAM algorithms is still at the start stage. A variety of problems related to it have not been solved, for example, how to deal with the features of different planes, how to combine the information of other aircraft to construct the 3D images of the environment, and how to deal with the problems caused by a large amount of data in the computing procedures [29][64]. Because of this, the polygon-based SLAM has attracted various scholars in recent years.

2.2.5 The filtering SLAM algorithms

The filtering SLAM algorithms are the earliest SLAM algorithms used by scholars to solve SLAM problems [28]. For example, when [30] processed the feature-based SLAM algorithms, they used the filtering data processing method to deal with the environmental information. According to [28], the filtering SLAM algorithms have been popularly used by scholars in the past. Many well-known SLAM algorithms are based on this type of data processing methods, such as particle filter SLAM (PF-SLAM), extended information filter SLAM (EIF-SLAM), and an extended Kalman filter SLAM (EKF-SLAM). Although these scholars use the filtering SLAM algorithms in different ways, they are based on the Bayes filter method [66][67]. However, various scholars use the filtering SLAM algorithms to improve their efficiency in processing data. For example, when scholars use the particle filter SLAM, they have taken different ways to deal with the information about each particle to increase data processing efficiency. [68] have proposed the distributed particle (DP) filtering method to filter data related to particles, which has optimized the ways to store data in the calculating process, improving the memory store ways to improve the efficiency of data processing. [69] have implemented the FastSLAM method to deal with the information for each particle. [70] have implemented the Rao–Blackwellized particle filters to improve the data filtering efficiency by enhancing the maps grid. [71] have proposed improving the particle network to reduce the filtering operations in the data processing. [72] have proposed using Hilbert maps to improve the grid of maps and the efficiency of data processing for each particle. Integrating the above examples of methods used by these scholars, although they have used the same idea for filtering data, they have tried to use different ways to control the data filtering procedures to increase efficiency.

Besides using different algorithms to increase the efficiency of filtering the information about each particle, the filtering SLAM algorithms can also be divided into two categories, vector-based filtering processing methods and set-based filtering processing methods [28]. The FastSLAM, as mentioned above, EIF-SLAM, and EKF-SLAM are all vector-based filtering processing methods because they treat the position of the robots in the map in a random vector. Different from the vector-based filtering method, the set-based filtering method has adopted the concept of the random finite set (RFS) [73][74][75]. Compared to vector-based filtering methods, set-based filtering methods are particularly efficient in dealing with the following problems: missed detections, false detections, and inaccurate measurements [28]. Similar to the vector-based filtering methods, the set-based filtering methods also contain a variety of algorithms when scholars use the RFS concept in different ways, such as the probability hypothesis density (PhD) and the Rao–Blackwellized (RB) PHD-SLAM [28][76]. Many scholars, such as [77], [78], and [75], have tested the set-based filtering methods, pointing out that they perform more effectively than the vector-based filtering methods, improving the accuracy of the calculation. Because of this, the set-based filtering methods have attracted more scholars' interest in recent years.

2.2.6 The smoothing SLAM algorithms

The smoothing SLAM algorithms are different from the filtering methods, focusing more on controlling the constraints related to nodes in the map or graph [28]. The most popular smoothing SLAM algorithms are the sub-map matching SLAM algorithm and the GraphSLAM algorithm [28]. The GraphSLAM algorithm was proposed by [47] and then developed by a variety of scholars, such as [79], [80], [81], [82], [83], and [84]. [79] have adopted the local registration and global correlation technique to connect local data with global maps. [80] have proposed a mesh-based method, using a distributed sensor network to finish the multi-robot mapping tasks. [81] have referred to the multigrid techniques to develop a multilevel relaxation algorithm. [82] have proposed the fast iterative alignment method to justify the robots' trajectory. [83] have proposed hierarchical optimization methods to deal with the information on maps, and [84] have proposed a generic linear constraint (GLCs) to remove the nodes on the maps to optimize the computing cost in the data processing procedures. Overall, the above scholars' findings reveal that the GraphSLAM algorithms focus on improving the efficiency of dealing with map nodes to improve the data processing procedures.

Different from the GraphSLAM algorithms, the sub-map matching SLAM algorithms focus on the methods of connecting different sub-maps, and it is based on the GraphSLAM algorithms. In detail, when the sub-map matching SLAM algorithm is used to connect sub-maps, it allows the operators to use the GraphSLAM algorithms to deal with the information in these sub-maps [29]. Because of this, when the sub-map matching algorithms are used, the operators can deal with the scenes with a large scale of maps, particularly the outdoor environments, and a variety of scholars' jobs have proved the efficiency of the sub-map matching SLAM algorithms in dealing with large scale environmental problems, such as [85], [86], [87], [88], [89], [90], and [91]. According to these scholars' findings, the sub-map matching SLAM algorithms are more likely to be map merging operations done by multiple-robot SLAM algorithms. The core of this type of SLAM algorithm is how to combine different sub-maps to construct the overall maps of the environment. To achieve this goal, various scholars have proposed other methods. For example, [85] and [89] used the hierarchical framework, [87] used the Atlas framework, and [91] used the lidar-based multi-robot SLAM system. According to [29], the promise of using the sub-map matching SLAM algorithms is the feasibility of the initial estimation of the transformation between different sub-maps, which allows the algorithms to make an exhaustive search among all sub-maps and match them. In this case, the sub-map matching SLAM algorithms have increased the requirement of memory stored in the calculating procedures.

2.2.7 Artificial Intelligence SLAM algorithms

Artificial intelligence (AI) SLAM algorithms have recently been relatively new and developed. This SLAM algorithm is not entirely independent of previous data processing because it depends on filtering and smoothing. Compared to the two types mentioned above of data processing SLAM algorithms, the primary value of the AI SLAM algorithms is to use different AI algorithms to improve the data processing procedures of the filtering and smoothing SLAM algorithms [29]. In recent years, many scholars have increased the investigation ion the AI SLAM algorithms. For example, [92] and [93] have tested the feasibility of using the neural network techniques in the data processing procedures of the SLAM algorithms, [94] have tried the combination of the genetic algorithm and the scan-matching SLAM algorithm. [95] have summarised the ways to combine the SLAM algorithms and different deep learning approaches. According to [29], integrating these AI algorithms with the traditional SLAM algorithms has reduced the requirements of building unique mathematical models in data professing for solving the SLAM problems. Still, the combination between them has also increased the complexity of the AI SLAM algorithms because this method requires a training procedure, which may need more data inputs and increase the time for computing.

2.2.8 Discussion of these SLAM algorithms

Overall, these SLAM algorithms have their advantages but also limitations in application. The feature-based SLAM algorithm helps detect the information of these maps when their geographic features are straightforward to identify. However, if the geometry features of the images for maps are not apparent, the feature-based SLAM algorithm cannot be effectively used. At the same time, the dependence on features limits the implementation of the SLAM algorithm because not all environments have typical characteristics. Compared to the feature-based SLAM algorithm, the other algorithms have reduced their dependence on features to increase their application in solving the SLAM problems, but they also have their limitations. The view-based SLAM algorithm depends on cameras' settings to collect views from different angles. If not enough opinions are available, this method may fail to identify the map information. The appearance-based SLAM algorithm is primarily designed to solve the loop closure problem, but not all scenes. In this case, its application is limited when it is intended. The polygon-based SLAM algorithm faces a similar situation when it is mainly designed to solve the 3D modelling problem related to map detection. Although the appearance-based SLAM algorithm and the polygon-based SLAM algorithm are to solve particular SLAM problems, they are still similar to the three algorithms mentioned above because they all interpret the information contained in images. Different from them, the filtering and smoothing SLAM algorithms are not so. They emphasize the data processing procedures to increase the efficiency of the SLAM algorithms. The AI SLAM algorithm applies the AI computing method in solving the SLAM problem, integrating the advantages of the above two categories of SLAM algorithms.

Integrating the above analysis of the development of different types of SLAM algorithms, its development trend can be summarised as follows. First, considering the perspective of several sensors or other shooting facilities, such as cameras, the SLAM algorithms are developed from one sensor or one camera to multiple sensors or cameras to solve more complex environments. Second, the SLAM algorithms are created from 2D to 3D scenes considering the map representation aspect. Third, considering the data processing perspective, the SLAM algorithms are developed from traditional data processing methods to AI and machine learning approaches. Fourth, considering the number of participants of robots, the SLAM algorithms are developed by solving environmental problems related to one robot to multiple robots. It should be pointed out that the above SLAM algorithms are mainly for single robot scenarios rather than multi-robots. Compared to the single-robot SLAM algorithms, the development of the multi-robot SLAM algorithms is still at the start stage. Its action also depends on the development of the single-robot SLAM algorithms. Because of this, the research on the single-robot SLAM algorithms is still mainstream in this field. Because of this, this essay focuses on the single-robot scenes and the multi-sensory SLAM algorithms.

2.3 The implementation of the SLAM algorithms in different scenes

Currently, the SLAM algorithms have been implemented in a variety of scenes. According to different criteria, the stages for the SLAM algorithms can be divided into different categories. For example, when considering the positions of the scenes, they can be divided into indoor and outdoor locations. Both the indoor and outdoor environments have attracted scholars to do more profound research [96][97]. Second, when considering the movement of the environments, the scenes can be divided into static and dynamic [98]. Scholars have already investigated the research of constructing static maps for many years, and more scholars are attracted by the application of the SLAM algorithms in solving problems related to dynamic scenes [99][100][101][102]. In the process of dealing with the problems related to dynamic scenes, the experience of constructing static maps is also adopted by scholars, and some scholars, for example, [101], have even tried to divide the dynamic scenes into different segments and then construct the static part and the moving part separately. Third, according to the structures of objects, the locations for various SLAM problems can be divided into 2D and 3D [103]. Fourth, according to the lighting situation of the environment, the scenes can be divided into textureless environment and lighting environment [104][105]. According to [105], various types of textureless environments exist. Still, their general features are dark and challenging to detect accurate information about the objects on the maps. Because of this, constructing maps for the textureless environment is still difficult, although various scholars have flowed into this field to solve this SLAM problem [104]. In this case, this essay tries to conduct more profound research to develop proper SLAM algorithms to solve the textureless environment problem.

2.4 The difficulties in detecting information about the textureless regions

Detecting information about the textureless regions faces many difficulties. First, the textureless areas are always dark, increasing the challenges in distinguishing the objects on the figures [106][107]. Because of this, [106] point out that the dark scenes have increased the difficulties of extracting information from the textureless regions, such as the edge of different objects. Then, it is difficult to identify the critical features of the textureless areas. In this case, the traditional visual SLAM algorithms cannot effectively identify the essential elements of the environments [108][20]. Mainly, [109] argue that it is difficult to identify the features of the surfaces of objects in textureless regions. The second difficulty facing the textureless environment is that the RGB-D sensors cannot effectively process the data of the environment, particularly in dynamic scenes [110]. The limitations of the RGB-D sensors cause this challenge because their data sizes are relatively small to justify the features of the objects in the textureless regions. Mainly, the RGB-D sensors are time-consuming, which has also increased the difficulties of identifying the key points from different frames shot by these RGB-D sensors [110]. According to the above scholars' analysis, identifying the features of objects in the textureless region is the most critical problem that must be considered in using the SLAM algorithms to solve this type of SLAM problem. Besides, increasing the data size of the textureless regions in the dynamic scenes and the 3D scenes is another significant problem that should be considered in developing SLAM algorithms to solve this type of problem.

Overall, compared to these texture scenes, how to detect the features of the textureless is one of the most important problems which must be solved. Under this condition, the SLAM mentioned above algorithms, such as the feature-based ones, cannot be directly used to solve the SLAM problem in the textureless regions. But the ideas of the other algorithms can be referred to. For example, the idea of using different information from images shot from different views and the loop closing treatment can be adopted in solving the textureless regions because they enable the operators to get more information about the environment. Hence, the advantages of SLAM, as mentioned above, should be referred to solve the SLAM problem of textureless areas.

2.5 The possible solutions to the problems of detecting information about the textureless regions

The textureless problems have attracted many scholars in recent years, and they have tested various methods to increase the accuracy of detecting the information in the textureless regions. One approach is to control the sensors and cameras on the robots to improve the quality of images for these textureless regions. For example, [111] and [112] have tested a method of using a polarisation camera to improve the images of the textureless regions. With the polarization camera, the operators can get the polarisation characteristics of the target objects in the textureless areas and then use the average vector and phase angle to reconstruct the objects [111]. [26] and [113] claim that the Microsoft Kinect sensors can also solve the problem of being unable to track apparent features of objects in textureless regions. These scholars' method is to improve the quality of information collected by sensors and cameras and then reduce the difficulties for the SLAM algorithms to extract data from these images.

Another method is to develop new algorithms to increase the ability of the traditional SLAM algorithms to extract information from images of the textureless regions. [114] have proposed the idea of using superpixels to modify the conventional monocular SLAM algorithms to increase the SLAM algorithms' abilities to extract more accurate information from the images of the textureless regions. According to [114][115], using superpixels can solve the problem of being unable to detect low-level features by these SLAM algorithms designed for the highly-textured regions. This method can also reduce the dependency on highly-qualified cameras and sensors. [116] have proposed using both feature plane patches and points to identify the pose of objects in the maps, rather than just considering the robustness of these featured plane patches and issues. In this case, [116] argue that this SLAM algorithm can perform better in identifying these objects with smooth edges and surfaces, such as the objects in the textureless regions, and they have also identified the stability of their SLAM algorithm in different scenarios.

Overall, both two orientations are researched by scholars to solve the mapping problems related to textureless regions. But compared to the first method which highly depends on the quality of cameras and sensors, the second method has attracted more scholars because this method just requires scholars to improve the mechanism of their SLAM algorithms, rather than to improve the quality of their robot facilities, which can be used in more scenarios related to the SLAM problems. Because of this, this essay wants to follow the second method to develop new SLAM algorithms to detect information in textureless regions.

2.6 Research gaps

Overall, previous scholars have done various jobs in developing the SLAM algorithms to solve the SLAM problems and have already obtained outstanding achievements. However, detecting the information of objects in the textureless regions is still tricky, and this SLAM problem has not been solved. Although some scholars have done some research in this field, proposing some methods to increase the ability of the SLAM algorithms to detect the features of objects in the textureless regions, the limitations of these algorithms are apparent. They either depend on using high-qualified cameras and sensors or make significant changes in the traditional SLAM algorithms, requiring the operators to control more professional skills in the data processing. In this case, these methods cannot be widely used in different scenes. Due to this, this essay tries to develop a SLAM algorithm depending on analyzing the data collected by multiple sensors, reducing the dependence on special sensors and cameras, on increasing its application in more scenarios.

Chapter 3

Methodology

In recent years, the rapid development of deep learning technology has provided new ideas for solving the above problems. Considering the rapidity and real-time, the SSD target detection framework is proposed to improve the speed while ensuring detection accuracy. To improve the running speed of neural networks, miniaturized networks represented by mobile are proposed. These networks reduce the amount of computation by ingeniously designing the network structure and simplifying the convolution kernel. ORB-SLAM2 algorithm is an excellent framework of visual SLAM algorithm based on feature point method in recent years. The image information stream input to the visual SLAM algorithm often contains various objects. Combined with the semantic information extraction advantages of the target detection network and the accurate geometric information obtained by the SLAM algorithm, the robot can get more structured, semantic and hierarchical map information from the surrounding environment. In recent years, some researchers have eliminated the dynamic points in the camera field according to the prior report of the target detection results and the measurement information of the active point detection algorithm to improve the positioning accuracy of the algorithm.

To solve the SLAM problem, a variety of SLAM algorithms have been developed by researchers, and one of the widely used SLAM algorithms is the ORB-SLAM2 algorithm. This SLAM algorithm is selected in this study because it is one of the practical algorithms used to solve the SLAM problem in the textureless environment, such as the indoor textureless environment [117][118]. In general, the ORB-SLAM2 algorithm is developed from the ORB-SLAM algorithm. The ORB-SLAM algorithm was created in 2015 to solve the visual SLAM problem for monocular cameras with an indirect method [119]. In 2017, the ORB-SLAM algorithm was extended to RGB and stereo cameras, developing into the ORB-SLAM2 algorithm [120]. The Stereo SLAM method does not depend on the features of images, making it a possible approach to solve the SLAM problems related to textureless environments. In contrast, the RGB-D SLAM method is a feature-based algorithm. Still, its accuracy is high, enabling operators to precisely reconstruct the map of a region, particularly in a room with a limited scale [44]. The ORB-SLAM2 has integrated the advantages of the primary ORB-SLAM algorithm, the Stereo SLAM method, and the RGB-D SLAM algorithm, which effectively solves the SLAM problems in the textureless region. Hence, it is applied in this study. In the following contents of this chapter, the details of the ORB-SLAM2 algorithm are explained. And then, in the next chapter, this algorithm is tested to show how it performs in tracing the route in a textureless environment.

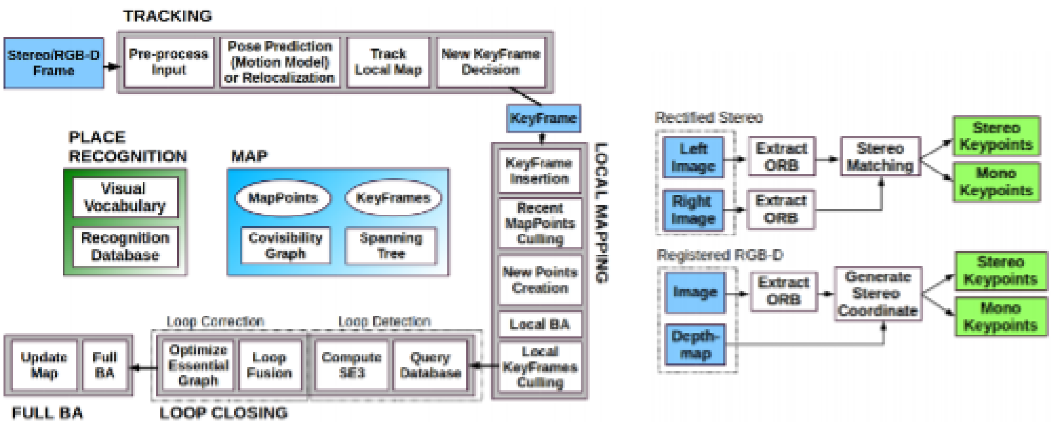
3.1 The general framework of the ORB-SLAM2 algorithm

The overall framework of the ORB-SLAM2 algorithm is shown in Figure 3-1. According to Figure 3-1(a), the calculating procedures of the ORB-SLAM2 algorithm can be divided into three threads. The first thread is to track the location of the cameras within each frame image. Based on Figure 3-1(a) contains three steps, including preprocessing input, prediction of poses or relocalization, tracking local map, and new keyframe decision. Within this thread, the key aim is to identify the features of these images and compare them with a region map to match elements and existing maps. When this thread is finished, it can reduce the errors in the reprojection process determined by the motion-only Bundle Adjustment (BA) method [44]. When this thread is finished, the information of the images caught by different cameras is analyzed and compared to existing maps of the region.

The second thread is to improve the local map. When the first thread has collected the information on the features of the objects in the region by analyzing the frame images caught by different cameras and comparing the image information with the local map, the second thread integrates the information from these images and the regional map to optimize it [44]. Figure 3-1(a) contains five steps, including keyframe insertion, recent map points culling, new points creation, local BA, and local keyframes. Finally, this thread is prepared for the third thread, enabling the mobile robots to correctly detect the textureless environment to optimize the solution of motion [121]. Overall, in this thread, in optimizing the local map, the local BA method is operated to modify the information of the local map.

The third thread is loop closing, which detects the available loops and then gradually adjusts them by implementing the pose-graph optimization operations. In this way, this thread can progressively correct the accumulated drifts to optimize the open circles. According to the information shown in Figure 3-1(a), this thread mainly contains four steps, including query database, compute SE3, loop fusion, and optimizing essential graph. Due to this, this thread first identifies the available loops according to the database constructed in the previous two lines and then gradually corrects the open circles. Finally, when the available coils are repaired, this thread can make a whole BA operation to optimize the local map and obtain an adjusted solution of the motion route for the objects in the textureless environment [44].

Overall, the above three threads are not superior relationships but parallel relationships. The three threads are calculated simultaneously when the input data is collected. When the images of the textureless environment are gathered, they are operated by the input pre-processing algorithm, transferring the image information into the information about key points to construct the habitat. The input pre-processing procedure comprises two parts, rectified stereo method and the registered RGB-D method, which are used to interpret stereo key points and mono vital points.



(a) System Threads and Modules. (b) Input pre-processing

Figure 3-1: The procedures of the ORB-SLAM2 algorithm (a) system threads and modules (b) input pre-processing, source: [44]

3.2 The mechanism of the ORB-SLAM2 algorithm

According to the above explanation of the general framework of the ORB-SLAM2 algorithm, the calculation of the following operations should be emphasized, including the analysis of the stereo keypoints, the calculation of the monocular keypoints (Mono keypoints), the motion-only BA operation, the loop closing operation, the whole BA operation, and the insertion of keyframes to form new maps. The details of these operations are explained as follows.

3.2.1 The state representation

A vector containing the location of a camera and the features of the location is used to represent the state of a submap X_B, which is shown in Equation 3.1 [122]. Here, X_c refers to the location of the camera, and $X_{f_{1:n}}$ refers to the features of the location of the camera, which can be divided into 1D parametrization and 3D parametrization.

$$X_B = \begin{bmatrix} X_c \\ X_{f_{1:n}} \end{bmatrix} = \begin{bmatrix} X_c \\ X_{1D} \\ X_{3D} \end{bmatrix} \tag{3.1}$$

In addition, the location of the camera, X_c, can be divided into four parts, including the Cartesian coordinates R, the Euler Angles Ψ, the linear velocity ν, and the angular velocity w. Hence, it can be presented as follows.

$$X_c = \begin{bmatrix} R \\ \Psi \\ \nu \\ w \end{bmatrix} \tag{3.2}$$

3.2.2 The calculation of the stereo keypoints

According to the information shown in Figure 3-1, stereo key points are essential inputs in the pre-processing stage to construct the local map. Generally, when a stereo keypoint is detected on an image shot by a stereo camera, it is labelled by a point with three coordinates, $X_{stereo} = (u_L, \nu\mathrm{L}, u_R)$. The three coordinates of the stereo point do not represent the 3-D coordinates of the point. In contrast, the three coordinates are the points on the left and right images where the subscript L represents the left image and R represents the right image. At the same time, u is the horizontal coordinate and ν is the vertical coordinate. In the process of detecting the stereo key points, ORB in two images are usually extracted and compared. In detail, the ORB information extracted from the left image is to be matched in the right image, and this matching process can be achieved by assuming that images are stereo rectified. Under this condition, the horizontal line becomes the epipolar line. This is why the horizontal coordinate on the right image (u_R) is selected as the third coordinate for a stereo key point. When the ORB on the right image has been found to match the coordinates of the point on the left image, a stereo key point is identified.

When the RGB camera extracts the stereo critical points from an image shot, the relationship between the two coordinates, u_L and u_R, is determined by the following Equation (Eq. 3.3) [120]. Here, f_x is the focal length in the horizontal direction, b is the distance between the infrared camera and the light projector within the environment, and d is the depth of a point on the RGB image. Usually, the value of b is determined by the camera, and its value is about 8cm for Asus Xtion and Kinect, which are two brands of cameras popularly used in automobiles.

$$u_R = u_L - \frac{f_{x^b}}{d} \tag{3.3}$$

The stereo key points are essential information to identify the features of the textureless environment. They can be divided into two categories, far stereo key points and close stereo key points. The close stereo key points refer to those whose depth values are less than 40 times the baseline of the stereo cameras and RGB-D cameras, while the far stereo key points are contrary to the close ones [122]. Both the near and far stereo key points are essential to identify the translation information and the scale of the images. In detail, when the depth value of the topics has been estimated, the close stereo key points can accurately provide the translation, rotation, and scale information when triangulated from one image frame. In contrast, the far stereo key points perform poorly in determining the translation, rotation, and scale information by one image frame. But they can provide the above information when more than one image frame is integrated. Due to this, when the far stereo key points are identified, they can offer the above statement by using images shot from different views. The integration of the close stereo key points and the far stereo key points can increase the accuracy of the translation information, rotation information, and scale information of the textureless environment.

Based on the identification of the stereo key points, the transformation information of the 3D points can be calculated by the following Equation, shown in Eq. 3.4. Here, $(u_r,\ \nu_r)$ is the surface coordinates of the points on the right image, and $(u_l,\ \nu_l)$ is that on the left image. $(u_o,\ \nu_o)$ is the coordinates of the central point on the image. Especially, d = u_l - u_r [122].

$$X_{3D} = f(u_r, \nu_r, u_l, \nu_l) = \left[x, y, z\right]^T = \left[\tfrac{b(u_r-u_o)}{d}, \tfrac{b(\nu_r-\nu_o)}{d}, \tfrac{f^b}{d}\right]^T \tag{3.4}$$

Meanwhile, the 1D feature of the location of the stereo camera can be directly presented as follows, shown in Equation 3.5. Here, R_i is the coordinates of the camera, θ_i, ϕ_i and ρ_i represent the azimuth, elevation, and depth of the point, respectively, while $\rho_i = \frac{1}{d_i}$

[122].

$$X_{1D} = \begin{bmatrix} R_i \\ \theta_i \\ \phi_i \\ \rho_i \end{bmatrix} \tag{3.5}$$

3.2.3 The calculation of the Mono keypoints

The Mono key points are different from the stereo critical points because they are only on the left images and are composed of two coordinates, defined as $X_{Mono} = (u_L, \nu_L)$. The Mono key points are supplementary to the stereo key points. The stereo key points refer to those key points which can be matched between the left and right images, and the Mono key points are the ones which cannot be checked on the right idea. This mainly happens in the photos shot by the RGB-D camera. According to Equation 3.1, the matched value of u_R can be calculated when the depth value of the point on the RGB-D cases can be identified. In contrast, when the depth value cannot be got, the matched value of u_R cannot be calculated. In this case, the Mono key points are usually points without valid depth values on the RGB-D images. Thus, the Mono keypoints can only provide the translation and rotation information, but not the scale information, since they do not have the depth value. Because of this, the Mono key points are always triangulated by the information collected from various views.

3.2.4 The motion-only BA operation

The motion-only BA operation is used to adjust the pose of cameras in the first thread, the tracking thread, and to prepare for the second thread, the local mapping thread. The keyframes and points related to a window are optimized in the motion-BA operation. As for the local BA, a local window is focused on. As for the full BA, all windows are treated.

The motion BA operation is focused on the orientation of a camera used to shoot images in the environment. In this case, the orientation of a camera is defined as R $\in$ SO(3). At the same time, the position of a camera is defined as t, which satisfies t $\in \mathbb{R}^3$. The motion-only BA operation is to optimize the position and orientations of the camera to reduce errors during the reprojection procedure. In general, the problem of the motion-only BA operation can be described in Equation 3.6. In this equation, $x^i_{(.)}$ represents the features of the key points, X^i represents the coordinates of the point to be matched with the key points. The subscript (.) is used to distinguish the Mono keypoints and the stereo keypoints. When it is s, it means stereo key points. When it is m, it means the Mono key points. Due to this, $x^i_m \in \mathbb{R}^2$ and $x^i_s \in \mathbb{R}^3$. $\pi_{(.)}$ is the projection function, which is defined in Equation 3.7.

$$\left\{R, t\right\} = \underset{R,t}{\arg\min} \sum\nolimits_{i\in x} \rho\left(\left\| x^i_{(.)} - \pi_{(.)}(RX^i + t)\right\|^2_{\Sigma}\right) \tag{3.6}$$

$$\pi_m\left(\begin{bmatrix} X \\ Y \\ Z \end{bmatrix}\right) = \begin{bmatrix} f_x \frac{x}{z} + c_x \\ f_y \frac{y}{z} + c_y \end{bmatrix}, \pi_s\left(\begin{bmatrix} X \\ Y \\ Z \end{bmatrix}\right) = \begin{bmatrix} f_x \frac{x}{z} + c_x \\ f_y \frac{y}{z} + c_y \\ f_x \frac{x-b}{z} + c_x \end{bmatrix} \tag{3.7}$$

In Equation, the focal length (f_x,f_y), the principal point (c_x,c_y), and the baseline b are all known when the position of a camera is calibrated.

When BA operation is used to deal with the points P_L within the covisible keyframes K_L, it becomes a local BA operation, and the BA operation is slightly changed, as shown in Equation 3.8. In this formula, K_F refers to those keyframes beyond K_L, and X_k indicates those matched points between the k keyframe and covisible keyframes K_L. According to the formula shown in Equation 3.8, the local BA operation has considered the contributions of the points in the surrounding keyframes on the projection of the points in the local keyframes to reduce projection errors.

$$\begin{cases} \left\{X^i, R_l, t_l | i \in P_L, l \in K_L\right\} = \underset{X^i, R_l, t_l}{\arg\min} \sum\nolimits_{k\in K_L \cup K_F} \sum\nolimits_{j \in X_k} \rho\left(E_{kj}\right) \\ E_{kj} = \left\| x^i_{(.)} - \pi_{(.)}(R_k X^j + t_k)\right\|^2_{\Sigma} \end{cases} \tag{3.8}$$

Overall, the local BA operation focuses on the local keyframes. Differently, the entire BA operations treat all keyframes to modify the information in the map. The complete BA considers all points in these keyframes.

3.2.5 The loop closing operation

The loop closing operation is one of the three threads in the ORB-SLAM2 algorithm. Optimizing the pose graph to reduce the possible accumulation of errors that occur in transformations [119][120]. Therefore, this operation is usually divided into two steps, one is to find a loop closure while the other is to correct it. The ORB-SLAM2 algorithm executes the loop closing operation by incorporating a whole BA operation and the keyframe update operation. The entire BA operation enables the ORB-SLAM2 algorithm to search new loops, and the keyframe update enables the balance of the information in the old and new maps. Because of this, the loop closing operation is usually together with the keyframe insertion operation.

3.2.6 The insertion of keyframes

The insertion of keyframes is frequently operated in the ORB-SLAM2 algorithm. It is performed when there are critical distances between the far stereo key points and the close stereo key points. When there are crucial distances between the distant stereo key points and the close stereo key points, the scenes between them are unclear, which may challenge the algorithm in identifying the environment between them because such situations have created a variety of possibilities for translation. This may impact the accuracy of translation estimation. In this case, inserting new keyframes is essential to identify the translation trajectory.

3.2.7 Localization mode

A localization mode is used in this algorithm. The localization mode does not create significant changes in the overall environment of the textureless region. Still, it can localize the mapped areas to reduce the drift efforts in matching the points in the unmapped territories. Furthermore, when the localization mode is used, the loop closing and local mapping threads can be contemporarily deactivated because the tracking thread can identify the camera's position by matching the points in the locally mapped areas according to the calculation of the visual odometry. This can reduce the matching operations to simplify the calculation process of the algorithm. In addition, the position of cameras can be tracked by continual relocalization [120][118][123].

3.2.8 Tools to develop the ORB-SLAM2 algorithm

This study, the ORB-SLAM2 algorithm is developed by combining the Visual C++ software and Python. The calculation procedures to find the points, key points, and keyframes are processed by Visual C++. This software also calculates the three threads. When these cameras' position information and objects are identified, the graph of these results is processed by Python. Also, the map and the moving trajectories are shown in Python.

3.3 Summary of the methodology

The ORB-SLAM2 algorithm helps detect the information of the textureless region. Taking the complex and dynamic indoor laboratory environment as the background, this paper explores the method of constructing a semantic map in the dynamic environment. Combined with the visual slam system based on an RGB-D camera and the SSD framework of deep convolution neural network based on multi-scale prediction based on regression prediction, this paper designs an algorithm to eliminate dynamic object points, fuse pose information and semantic information, and construct a semantic target database to construct a high-level semantic map in real-time. The method proposed in this paper has a specific reference value in creating semantic maps in a dynamic environment and can help robots achieve more intelligent navigation tasks. In the following chapter, an example of detecting the indoor textureless region is used to prove the effectiveness of the ORB-SLAM algorithm in solving the SLAM problem.

Chapter 4

Empirical analysis

In this chapter, an actual example is used to prove the effectiveness of the ORB-SLAM2 algorithm in solving the SLAM problem in a textureless environment. This chapter contains three main parts, including a brief introduction of the tested background, an illustration of the results of the ORB-SLAM2 algorithm and its comparison with other tracking methods, and a comprehensive discussion.

4.1 A brief explanation of the textureless environment

The situation of an indoor textureless environment is shown in Figure 4-1. The objects in the domain contain several chairs, boxes, a sofa, several TV monitors, several potted plants, a door, two lights, and a locker. The following rules are used to set critical points to identify the positions of these objects. First, the turning points on the surface of an object are treated as essential points to determine the object's shape. Second, the changes in the light and shade regions are treated as critical points. Third, the contact points between an object and another are treated as essential points.

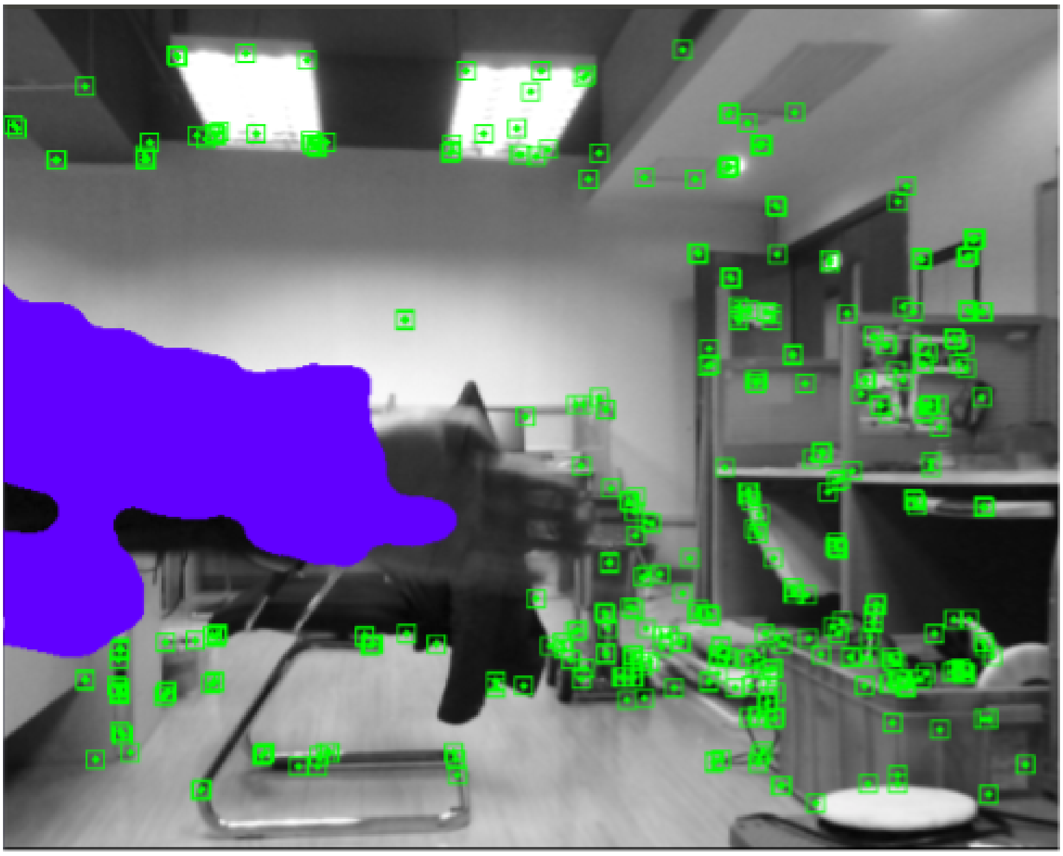


Figure 4-1: The textureless environment of an indoor region

According to the above principles, the critical points in the visual image are identified, shown in Figure 4-1, and labelled as green points. Then, based on identifying the essential attributes of these objects in the indoor environment, the projections of these objects are obtained, as shown in Figure 4-2. Then, the map of the indoor environment is constructed, shown in Figure 4-3.

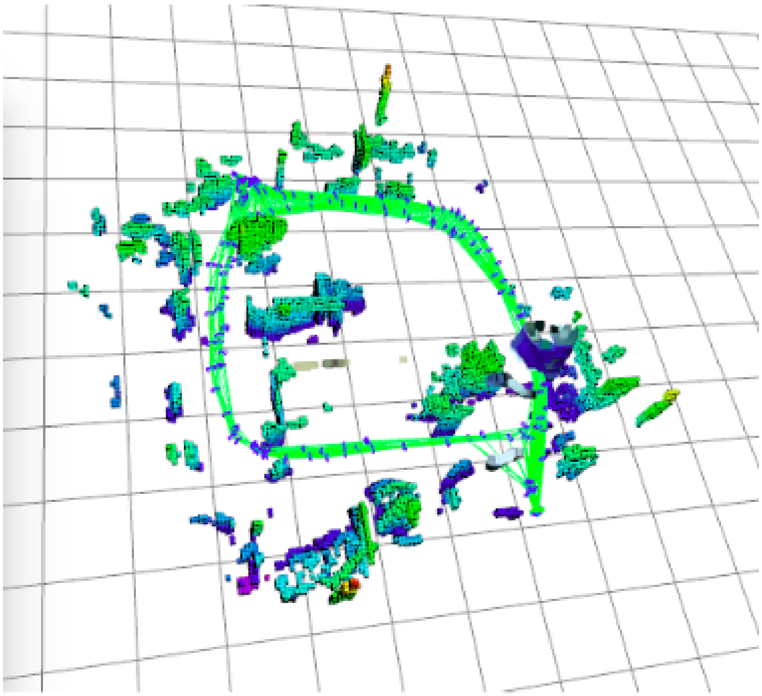


Figure 4-2: The projections of these objects in the indoor environment

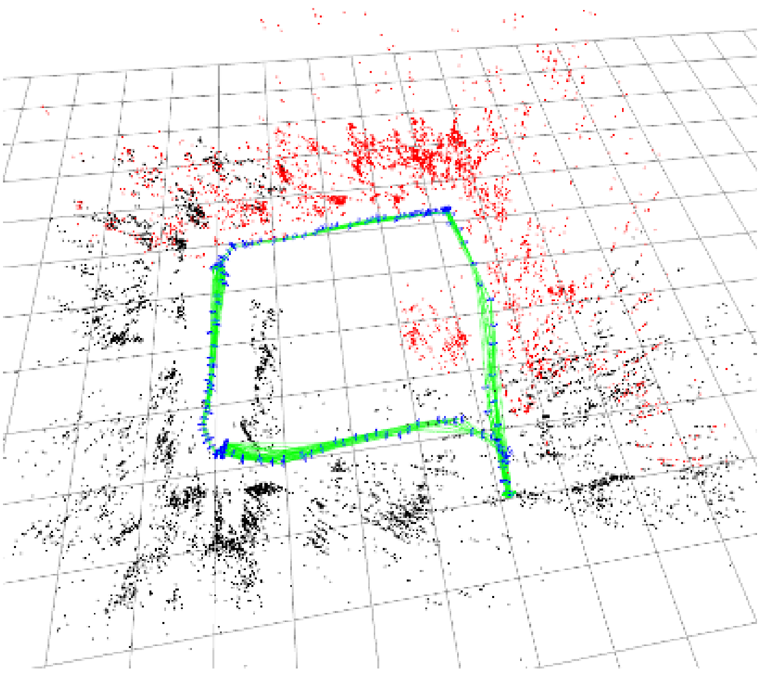


Figure 4-3: The map of the indoor environment

The positioning results are shown in Figure 4-4. To illustrate the accuracy of the ORB-SLAM2 algorithm in identifying the moving points, the results of several different methods are compared, including the camera with optical flow moving point detection and multi-view geometric moving point detection, shown in Figure 4-4. In this figure, the red curve represented by ground truth is the actual camera track recorded by the motion capture device, the green curve represented by orb-slam2-src is the camera track obtained by the original orb-slam2 algorithm, the blue curve represented by orb-slam2-flow is the camera track obtained by the orb-slam2 algorithm using the optical flow algorithm to eliminate dynamic points, and the yellow curve represented by orb-slam2-geom is the camera track obtained by orb-slam2 algorithm, which uses multi-view geometry algorithm to eliminate emotional issues. Then, the moving trajectories obtained from different methods are shown in Figure 4-5.

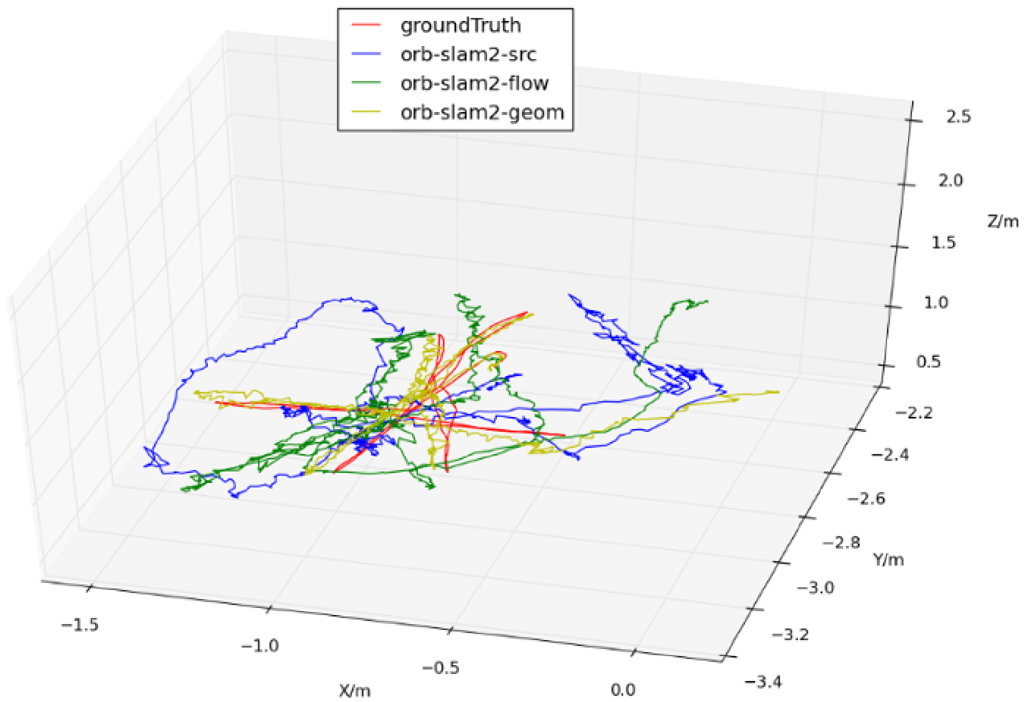


Figure 4-4: The positioning results of the ORB-SLAM2 algorithm and some other methods

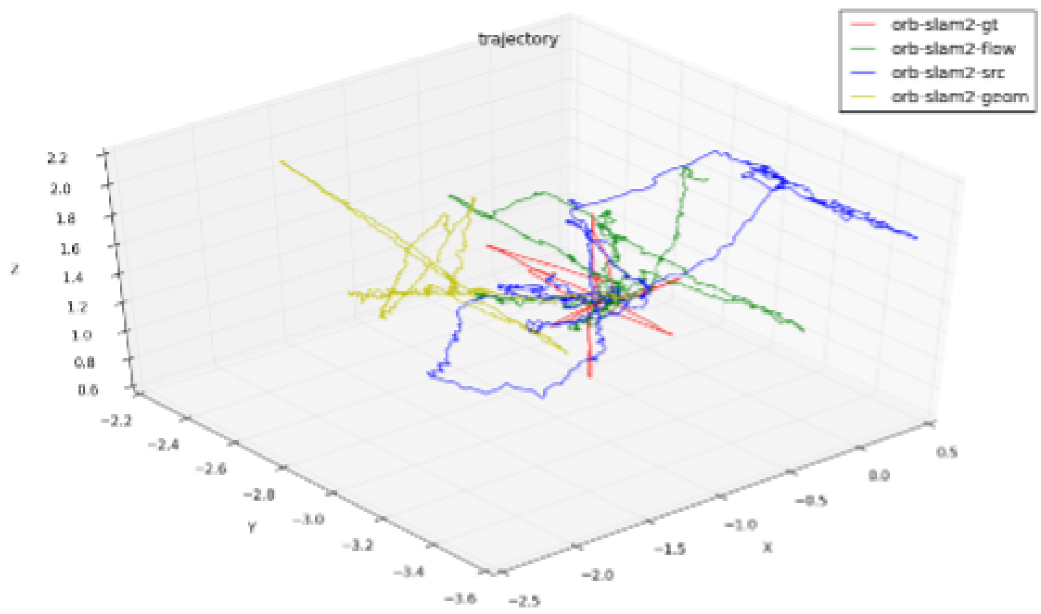


Figure 4-5: The moving trajectories in the indoor environment obtained from different methods

4.2 Analysis of the results of the ORB-SLAM2 algorithm

According to the positioning results shown in Figure 4-4, a general impression is that the ORB-SLAM2 algorithm helps identify the complex situation of a complex indoor environment. Still, the accuracy of the ORB-SLAM2 algorithm is changed when it is combined with other algorithms to deal with the dynamic points in the background. First, comparing the results of ORB-SLAM2-SRC, ORB-SLAM2-FLOW, and ORB-SLAM2-GEOM in Figure 4-4, the results of ORB-SLAM2-GEOM are the best in identifying the positions because it is the most similar one compared to the ground truth. In contrast, the results of the original ORB-SLAM2 algorithm are the worst because the functions identified by this algorithm have a large gap with the positions of the ground truth. This reflects that the original ORB-SLAM2 algorithm cannot perform well in reducing the influence of the dynamic points. The results of the ORB-SLAM2 algorithm are improved when combined with the optical flow algorithm in that the emotional issues are removed. However, the visual algorithm does not perform well in reducing the accumulated drift errors in the translational procedures. This is why the walking trajectory of the camera for this method has a large gap compared to the walking revolution of the ground truth. In contrast, the combination of the ORB-

SLAM2 algorithm and the multi-view geometry algorithm is much better in reducing the accumulated drift errors, and the walking trajectory of this method is close to the ground truth.

In addition, the details of the comparison between the walking trajectories of the three ORB-SLAM2 algorithms and the ground truth are shown in Table 4.1. According to the comparison results shown in Table 4.1, when the influence of dynamic points is not removed, the accuracy of the ORB-SLAM2 algorithm is inferior, and the mean gap between the captured moments and the ground truth is about 0.582247m. Integrating the information shown in Figures 4-4 and 4-5, the shape of the original ORB-SLAM2 algorithm is also quite different from the ground truth.

Table 4.1: The comparison of the translational errors of the three algorithms and the ground truth

	ORB-SLAM2-SRC vs. Ground truth	ORB-SLAM2-FLOW vs. Ground truth	ORB-SLAM2-GEOM vs. Ground truth
Comparison pairs	826	826	826
Mean (m)	0.582247	0.316048	0.070537
Median (m)	0.520550	0.203472	0.041738
Std. Dev. (m)	0.392580	0.224227	0.134097
Min (m)	0.077351	0.070175	0.002700
Max. (m)	1.476973	1.052806	0.889107
RMSE (m)	0.702233	0.387510	0.151517

However, the situation is greatly improved when the optical flow algorithm and the multi-view geometry algorithm are used to remove the influence of dynamic points. When the optical flow algorithm is used, the mean distance between the captured points and the ground truth is nearly halved. When using the multi-view geometry algorithm, the mean distance between the captured points and the ground truth is almost reduced to a minimal value. In addition, integrating the illustration shown in Figures 4-4 and 4-5, the shapes of the walking trajectories of the two combined algorithms are similar to the walking circuit of the ground truth. Considering this perspective, the two algorithms help improve the accuracy of the ORB-SLAM2 algorithm, and the multi-view geometry algorithm performs better than the optical flow algorithm in reducing the accumulated drift errors.

Chapter 5

Conclusion

This paper has studied the SLAM problem and tested the efficiency of the ORB-SLAM2 algorithm in solving this problem. Based on the above results of the empirical analysis, the ORB-SLAM2 algorithm helps solve the SLAM problem in an indoor textureless environment. First, when the original ORB-SLAM2 algorithm is used, it can generally identify the positions of the stereo keyframes and RGB-D keyframes, as well as the key points to construct the general map of the textureless environment. However, the accuracy of the original ORB-SLAM2 algorithm is not too high when it has not removed the influence of the dynamic points. But this problem can be solved by integrating the original ORB-SLAM2 algorithm with other algorithms, such as the optical flow algorithm and the multi-view geometry algorithm, to reduce the influence of the dynamic points on the identification of the positions and trajectories. Especially the accuracy of the combination of the original ORB-SLAM2 algorithm and the multi-view geometry algorithm is much higher than the accuracy of the combination of the original ORB-SLAM2 algorithm and the optical flow algorithm. Overall, the results of the above empirical analysis illustrate that the ORB-SLAM2 algorithm is available to solve the SLAM problem in the indoor textureless environment when the stereo cameras shoot only visual images and RGB-D cameras are used. Therefore, the ORB-SLAM2 should be appreciated in solving the SLAM problem in the textureless environment.

However, this study should not ignore the following limitations. On the one hand, the example of an empirical analysis used in this study is a small room representing a region on a small scale. Considering this perspective, the results of the empirical research can support that the ORB-SLAM2 algorithm helps solve the SLAM problem in the textureless environment on a small scale. However, on the other hand, the effectiveness of the ORB-SLAM2 algorithm in solving the SLAM problem in the textureless environment on a large scale is not identified. To deal with this limitation, a future study can research the effectiveness of this ORB-SLAM2 algorithm by testing an open scene, for example, a street at night without any street lights. On the other hand, this study has focused on the effectiveness of the original ORB-SLAM2 algorithm in solving the SLAM problem. The visual images used in this example are mainly from stereo and RGG-D cameras. In the future study, this algorithm can be extended to the scenes using other types of cameras to increase the use of this algorithm in solving the SLAM problem in different stages.

Appendix A

ORB_SLAM2 CODE

A.1 Map.h

orb_slam_2_ros/orb_slam2/include/Map.h

```
public:
    void LoadMap(SystemSetting* systemSetting,
            const string &fileName);
    MapPoint* LoadMapPoint(ifstream &fin);
    KeyFrame* LoadKeyFrame(SystemSetting* systemSetting,
            ifstream &fin);
    void SaveMap(const string &fileName);
    std::map<MapPoint*,unsigned long int> mapPointsIndex;

protected:
    void SaveMapPoint(ofstream &fout, MapPoint* mapPoint);
    void SaveKeyFrame(ofstream &fout, KeyFrame* keyFrame);
```

A.2 Map.cc

orb_slam_2_ros/orb_slam2/src/Map.cc

```
void Map::SaveMap(const string& fileName)
{
    cerr<<"Map.cc :: SaveMap to "<<fileName <<endl;
    ofstream fout;
```

```
    fout.open(fileName.c_str(), ios_base::out|ios::binary);
    unsigned long int mapPointsSize = mspMapPoints.size();
    fout.write((char*)&mapPointsSize, sizeof(mapPointsSize));
    for (auto mapPoint: mspMapPoints){
        SaveMapPoint(fout, mapPoint);
    }
    cerr << "Map.cc::mapPointsSize is :"<<mapPointsSize<<endl;
    GetMapPointsIndex();
    unsigned long int keyFramesSize = mspKeyFrames.size();
    cerr <<"Map.cc::keyFramesSize is :"<<keyFramesSize<<endl;
    fout.write((char*)&keyFramesSize, sizeof(keyFramesSize));
    for (auto keyFrame: mspKeyFrames)
        SaveKeyFrame(fout, keyFrame);
    for (auto keyFrame:mspKeyFrames)
    {
        KeyFrame* kfParent = keyFrame->GetParent();
        unsigned long int parentId = ULONG_MAX;
        if (kfParent)
            parentId = kfParent->mnId;
        fout.write((char*)&parentId, sizeof(parentId));
        unsigned long int ckfSize =
            keyFrame->GetConnectedKeyFrames().size();
        fout.write((char*)&ckfSize, sizeof(ckfSize));
        for (auto ckf: keyFrame->GetConnectedKeyFrames())
        {
            int weight = keyFrame->GetWeight(ckf);
            fout.write((char*)&ckf->mnId, sizeof(ckf->mnId));
            fout.write((char*)&weight, sizeof(weight));
        }
    }
    fout.close();
    cerr<<"Map.cc :: Map Saving Map Finished!"<<endl;
}

void Map::SaveMapPoint(ofstream& fout, MapPoint* mapPoint)
{
    fout.write((char*)&mapPoint->mnId, sizeof(mapPoint->mnId));
```

```
    cv::Mat matWorldPos = mapPoint->GetWorldPos();
    fout.write((char*)& matWorldPos.at<float>(0),
                        sizeof(float));
    fout.write((char*)& matWorldPos.at<float>(1),
                        sizeof(float));
    fout.write((char*)& matWorldPos.at<float>(2),
                        sizeof(float));
}

void Map::SaveKeyFrame(ofstream &fout, KeyFrame* keyFrame)
{
    fout.write((char*)&keyFrame->mnId, sizeof(keyFrame->mnId));
    fout.write((char*)&keyFrame->mTimeStamp,
                    sizeof(keyFrame->mTimeStamp));
    cv::Mat matPose = keyFrame->GetPose();
    std::vector<float> stdQ = Converter::toQuaternion(matPose);
    for (int i = 0; i < 3; i ++)
        fout.write((char*)&matPose.at<float>(i,3),
                    sizeof(float));
    for (int i = 0; i < 4; i ++)
        fout.write((char*)&stdQ[i],sizeof(float));
    fout.write((char*)&keyFrame->N, sizeof(keyFrame->N));
    for(int i = 0; i < keyFrame->N; i++)
    {
        cv::KeyPoint keyPoint = keyFrame->mvKeys[i];
        fout.write((char*)&keyPoint.pt.x,
                sizeof(keyPoint.pt.x));
        fout.write((char*)&keyPoint.pt.y,
                sizeof(keyPoint.pt.y));
        fout.write((char*)&keyPoint.size,
                sizeof(keyPoint.size));
        fout.write((char*)&keyPoint.angle,
                sizeof(keyPoint.angle));
        fout.write((char*)&keyPoint.response,
                sizeof(keyPoint.response));
        fout.write((char*)&keyPoint.octave,
                sizeof(keyPoint.octave));
```

```
        fout.write((char*)&keyPoint->mDescriptors.cols,
        sizeof(keyFrame->mDescriptors.cols));
        for (int j = 0; j < keyFrame->mDescriptors.cols; j++)
            fout.write((char*)&keyFrame->
                mDescriptors.at<unsigned char>(i,j),
                sizeof(char));
        unsigned long int mnIndex;
        MapPoint* mapPoint = keyFrame->GetMapPoint(i);
        if (mapPoint == NULL)
            mnIndex = ULONG_MAX;
        else
            mnIndex = mapPointsIndex[mapPoint];
        fout.write((char*)&mnIndex, sizeof(mnIndex));
    }
}

void Map::GetMapPointsIndex()
{
    unique_lock<mutex> lock(mMutexMap);
    unsigned long int i = 0;
    for (auto mapPoint: mspMapPoints)
    {
        mapPointsIndex[mapPoint] = i;
        i += 1;
    }
}

void Map::LoadMap(SystemSetting* systemSetting,
                  const string &fileName)
{
    cerr << "Map.cc :: Map load from:"<<fileName<<endl;
    ifstream fin;
    fin.open(fileName.c_str(),ios::binary);
    unsigned long int mapPoints;
    fin.read((char*)&mapPoints, sizeof(mapPoints));
    cerr<<"Map.cc :: mapPoints is :"<<mapPoints<<endl;
    for (unsigned int i = 0; i < mapPoints; i++)
```

```
    {
        MapPoint* mapPoint = LoadMapPoint(f);
        AddMapPoint(mapPoint);
    }
    unsigned long int keyFrames;
    fin.read((char*)&keyFrames, sizeof(keyFrames));
    cerr<<"Map.cc :: keyFrames is :"<<keyFrames<<endl;
    vector<KeyFrame*>kfByOrder;
    for(unsigned int i = 0; i < keyFrames; i++)
    {
        KeyFrame* keyFrame = LoadKeyFrame(fin, systemSetting);
        AddKeyFrame(keyFrame);
        kfByOrder.push_back(keyFrame);
    }
    cerr<<"Map.cc :: KeyFrame Load Finished!"<<endl;
    map<unsigned long int, KeyFrame*> kfById;
    for (auto keyFrame: mspKeyFrames)
        kfById[keyFrame->mnId] = keyFrame;
    cerr<<"Map.cc :: Map Start Load The Parent!"<<endl;
    for(auto kf: kfByOrder)
    {
        unsigned long int parentId;
        fin.read((char*)&parentId, sizeof(parentId));
        if (parentId != ULONG_MAX)
            kf->ChangeParent(kfById[parentId]);
        unsigned long int con;
        fin.read((char*)&con, sizeof(con));
        for (unsigned long int i = 0; i < con; i++)
        {
            unsigned long int id;
            int weight;
            fin.read((char*)&id, sizeof(id));
            fin.read((char*)&weight, sizeof(weight));
            kf->AddConnection(kfById[id],weight);
        }
   }
   cerr<<"Map.cc :: Map Parent Load Finished!"<<endl;
```

```
    std::vector<MapPoint*> vAllMapPoints = GetAllMapPoints();
    for (auto mapPoint: vAllMapPoints)
    {
        if(mapPoint)
        {
            mapPoint->ComputeDistinctiveDescriptors();
            mapPoint->UpdateNormalAndDepth();
         }
    }
     fin.close();
     cerr<<"Map.cc :: Load Map Finished!"<<endl;
     return;
}

MapPoint* Map::LoadMapPoint(ifstream &fin)
{
     cv::Mat Position(3,1,CV_32F);
     long unsigned int id;
     fin.read((char*)&id, sizeof(id));
     fin.read((char*)&Position.at<float>(0), sizeof(float));
     fin.read((char*)&Position.at<float>(1), sizeof(float));
     fin.read((char*)&Position.at<float>(2), sizeof(float));
     MapPoint* mapPoint = new MapPoint(Position, this);
     mapPoint->mnId = id;
     mapPoint->SetWorldPos(Position);
     return mapPoint;
}


KeyFrame* Map::LoadKeyFrame(SystemSetting* ystemSetting,
                                  ifstream &fin)
{
     KeyFrameInitialization kfInit(*systemSetting);
     fin.read((char*)&kfInit.nId, sizeof(kfInit.nId));
     fin.read((char*)&kfInit.timeStampNum, sizeof(double));
     cv::Mat cvMat = cv::Mat::zeros(4,4,CV_32F);
     std::vector<float> stdQuat(4);
```

```
    for (int i = 0; i < 4; i++)
        fin.read((char*)&stdQuat[i], sizeof(float));
    cv::Mat cvMatChild = Converter::toCvMat(stdQuat);
    for (int i = 0; i < 3; i++)
        fin.read((char*)&T.at<float>(i,3), sizeof(float));
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            cvMat.at<float>(i,j) = cvMatChild.at<float>(i,j);
    cvMat.at<float>(3,3) = 1;
    fin.read((char*)&kfInit.N, sizeof(kfInit.N));
    kfInit.vKps.reserve(kfInit.N);
    kfInit.descriptors.create(kfInit.N, 32, CV_8UC1);
    vector<float>KeypointDepth;
    std::vector<MapPoint*> vpMapPoints;
    vpMapPoints = vector<MapPoint*>(kfInit.N,
                        static_cast<MapPoint*>(NULL));
    std::vector<MapPoint*> vAllMapPoints = GetAllMapPoints();
    for(int i = 0; i < kfInit.N; i++)
    {
        cv::KeyPoint keyPoint;
        fin.read((char*)&keyPoint.pt.x, sizeof(keyPoint.pt.x));
        fin.read((char*)&keyPoint.pt.y, sizeof(keyPoint.pt.y));
        fin.read((char*)&keyPoint.size, sizeof(keyPoint.size));
        fin.read((char*)&keyPoint.angle,
                        sizeof(keyPoint.angle));
        fin.read((char*)&keyPoint.response,
                        sizeof(keyPoint.response));
        fin.read((char*)&keyPoint.octave,
                        sizeof(keyPoint.octave));
        kfInit.vKps.push_back(keyPoint);
        fin.read((char*)&kfInit.descriptors.cols,
                    sizeof(kfInit.descriptors.cols));
        for (int j = 0; j < kfInit.descriptors.cols; j++)
            fin.read((char*)&kfInit.descriptors.
            at<unsigned char>(i,j),sizeof(char));
        unsigned long int mpIndex;
        f.read((char*)&mpIndex, sizeof(mpidx));
```

```
            if(mpIndex == ULONG_MAX)
                vpMapPoints[i] = NULL;
            else
                vpMapPoints[i] = vAllMapPoints[mpIndex];
    }
    kfInit.vRight = vector<float>(kfInit.N,-1);
    kfInit.vDepth = vector<float>(kfInit.N,-1);
    kfInit.UndistortKeyPoints();
    kfInit.AssignFeaturesToGrid();
    KeyFrame* keyFrame = new KeyFrame(kfInit,
                        this, NULL, vpMapPoints);
    keyFrame->mnId = kfInit.nId;
    keyFrame->SetPose(cvMat);
    keyFrame->ComputeBoW();
    for (int i = 0; i < kfInit.N; i++)
    {
        if (vpMapPoints[i])
        {
            vpMapPoints[i]->AddObservation(keyFrame,i);
            if(!vpMapPoints[i]->GetReferenceKeyFrame())
                vpMapPoints[i]->SetReferenceKeyFrame(keyFrame);
        }
    }
    return keyFrame;
}
```

A.3 Converter.cc

orb_slam_2_ros/orb_slam2/src/Converter.cc

```
cv::Mat Converter::toCvMat(const std::vector<float>& stdV)
{
    Eigen::Quaterniond eq;
    eq.x() = stdV[0];
    eq.y() = stdV[1];
    eq.z() = stdV[2];
    eq.w() = stdV[3];
    Eigen::Matrix<double,3,3> eigMat(eq);
```

```
    cv::Mat cvMat = toCvMat(eigMat);
    return cvMat;
}
```

A.4 System.h

orb_slam_2_ros/orb_slam2/include/System.h

```
public:
    void SaveSystemMap(const string &fileName);
    void LoadSystemMap(const string &fileName);

private:
    std::string strSettingsFile;
```

A.5 System.cc

orb_slam_2_ros/orb_slam2/src/System.cc

```
System::System(const string strVocFile,
               const string strSettingsFile,
               const eSensor sensor,
               const std::string & map_file,
               bool load_map):
               mSensor(sensor),
               mbReset(false),
               mbActivateLocalizationMode(false),
               mbDeactivateLocalizationMode(false),
               map_file(map_file), load_map(load_map)
{
    // Add to System constructor
    strSettingsFile = strSettingsFile;
}

void System::SaveSystemMap(const string &fileName)
{
    mpMap->SaveMap(fileName);
}
```

```

void System::LoadSystemMap(const string &fileName)
{
      SystemSetting* systemSetting =
         new SystemSetting(mpVocabulary);
      systemSetting->LoadSystemSetting(strSettingsFile);
      mpMap->LoadMap(fileName, systemSetting);
}
```

A.6 MapPoint.h

orb_slam_2_ros/orb_slam2/include/MapPoint.h

```
public:
     MapPoint(const cv::Mat &matPos,Map* map);
     KeyFrame* SetKeyFrame(KeyFrame* keyFrame);
```

A.7 MapPoint.cc

orb_slam_2_ros/orb_slam2/src/MapPoint.cc

```
MapPoint::MapPoint(const cv::Mat &matPos,Map* map):
                       mnFirstKFid(0),
                       mnFirstFrame(0),
                       nObs(0),
                       mnTrackReferenceForFrame(0),
                       mnLastFrameSeen(0),
                       mnBALocalForKF(0),
                       mnFuseCandidateForKF(0),
                       mnLoopPointForKF(0),
                       mnCorrectedByKF(0),
                       mnCorrectedReference(0),
                       mnBAGlobalForKF(0),
                       mpRefKF(static_cast<KeyFrame*>(NULL)),
                       mnVisible(1),
                       mnFound(1),
                       mbBad(false),
                       mpReplaced(static_cast<MapPoint*>(NULL)),
```

```
                        mfMinDistance(0),
                        mfMaxDistance(0),
                        mpMap(map)
{
      Pos.copyTo(mWorldPos);
      mNormalVector = cv::Mat::zeros(3,1,CV_32F);
      unique_lock<mutex> lock(mpMap->mMutexPointCreation);
      mnId = nNextId++;
}

KeyFrame* MapPoint::SetKeyFrame(KeyFrame* keyFrame)
{
     return mpRefKF = keyFrame;
}
```

A.8 KeyFrame.h

orb_slam_2_ros/orb_slam2/include/KeyFrame.h

```
#include "KeyFrameInitialization.h"

class KeyFrameInitialization;

public:
     KeyFrame(KeyFrameInitialization &kfInit,
          Map *map, KeyFrameDatabase* kfDB,
          vector<MapPoint*>& vpMapPoints);
```

A.9 KeyFrame.cc

orb_slam_2_ros/orb_slam2/src/KeyFrame.cc

```
KeyFrame::KeyFrame(KeyFrameInitialization &kfInit,
                        Map *map,
                        KeyFrameDatabase *kfDB,
                        vector<MapPoint*> &vpMapPoints):
               mnFrameId(0),
               mTimeStamp(kfInit.timeStampNum),
```

```
            mnGridCols(FRAME_GRID_COLS),
            mnGridRows(FRAME_GRID_ROWS),
        mfGridElementWidthInv(kfInit.gridElementWidthInv),
        mfGridElementHeightInv(kfInit.gridElementHeightInv,
            mnTrackReferenceForFrame(0),
            mnFuseTargetForKF(0),
            mnBALocalForKF(0),
            mnBAFixedForKF(0),
            mnLoopQuery(0),
            mnLoopWords(0),
            mnRelocQuery(0),
            mnRelocWords(0),
            mnBAGlobalForKF(0),
            fx(kfInit.fbx),
            fy(kfInit.fby),
            cx(kfInit.x),
            cy(kfInit.y),
            invfx(kfInit.fx),
            invfy(kfInit.fy),
            mbf(kfInit.bf),
            mb(kfInit.b),
            mThDepth(kfInit.thDepth),
            N(kfInit.N),
            mvKeys(kfInit.stdvKeyPoint),
            mvKeysUn(kfInit.stdvKeyPointUn),
            mvuRight(kfInit.stdvRight),
            mvDepth(kfInit.stdvDepth),
            mDescriptors(kfInit.matDescriptors.clone()),
            mBowVec(kfInit.BowVec),
            mFeatVec(kfInit.FeatVec),
            mnScaleLevels(kfInit.scaleLevels),
            mfScaleFactor(kfInit.scaleFactor),
            mfLogScaleFactor(kfInit.logScaleFactor),
            mvScaleFactors(kfInit.scaleFactors),
            mvLevelSigma2(kfInit.levelSigma2),
            mvInvLevelSigma2(kfInit.invLevelSigma2),
            mnMinX(kfInit.minX),
```

```
                mnMinY(kfInit.minY),
                mnMaxX(kfInit.maxX),
                mnMaxY(kfInit.maxY),
                mK(kfInit.K),
                mvpMapPoints(vpMapPoints),
                mpKeyFrameDB(kfDB),
                mpORBvocabulary(kfInit.vocabulary),
                mbFirstConnection(true),
                mpParent(NULL),
                mbNotErase(false),
                mbToBeErased(false),
                mbBad(false),
                mHalfBaseline(kfInit.b/2),
                mpMap(map)
{
    mnId = nNextId++;
}
```

A.10 SystemSetting.h

orb_slam_2_ros/orb_slam2/include/SystemSetting.h

```
#ifndef SYSTEMSETTING_H
#define SYSTEMSETTING_H

#include<string>
#include"ORBVocabulary.h"
#include<opencv2/opencv.hpp>


namespace ORB_SLAM2 {

    class SystemSetting{

        public:
            SystemSetting(ORBVocabulary* voc);
            bool LoadSysSetting(const std::string path);

```

```
        public:
            ORBVocabulary* vocavulary;
            float width, height;
            float x, y;
            float wx, hy;
            float b, bf, fps;
            float fx, fy;
            cv::Mat K;
            cv::Mat matDistCoef;
            bool initialized;
            float depth = -1;
            float depthMap = -1;
            float scaleFactor;
            int RGB, features, levels;
            float minThFAST;
            float iniThFAST;
    };

}//namespace ORB_SLAM2

#endif //SystemSetting
```

A.11 SystemSetting.cc

orb_slam_2_ros/orb_slam2/src/SystemSetting.cc

```
#include<iostream>
#include"SystemSetting.h"

using namespace std;

namespace ORB_SLAM2 {

    SystemSetting::SystemSetting(ORBVocabulary* voc):
                            vocavulary(voc)
    {

    }
```

```

bool SystemSetting::LoadSysSetting(const std::string path)
{
    cout<<endl<<"Loading Sys Setting form:"<<path<<endl;
    cv::FileStorage fSettings(path,
        cv::FileStorage::READ);
    width  = fSettings["Camera.width"];
    height = fSettings["Camera.height"];
    x      = fSettings["Camera.cx"];
    y      = fSettings["Camera.cy"];
    wx     = fSettings["Camera.fx"];
    hy     = fSettings["Camera.fy"];
    cv::Mat cvMat = cv::Mat::eye(3,3,CV_32F);
    cvMat.at<float>(0,2) = x;
    cvMat.at<float>(1,2) = y;
    cvMat.at<float>(0,0) = wx;
    cvMat.at<float>(1,1) = hy;
    cvMat.copyTo(K);
    cv::Mat tmpDistCoef(4,1,CV_32F);
    tmpDistCoef.at<float>(2) = fSettings["Camera.p1"];
    tmpDistCoef.at<float>(3) = fSettings["Camera.p2"];
    tmpDistCoef.at<float>(0) = fSettings["Camera.k1"];
    tmpDistCoef.at<float>(1) = fSettings["Camera.k2"];
    const float k3 = fSettings["Camera.k3"];
    if(k3!=0)
    {
        tmpDistCoef.resize(5);
        tmpDistCoef.at<float>(4) = k3;
    }
    tmpDistCoef.copyTo(matDistCoef);
    bf = fSettings["Camera.bf"];
    fps= fSettings["Camera.fps"];
    fx = 1.0f/wx;
    fy = 1.0f/wy;
    b     = bf/wx;
    initialized = true;
    if(matDistCoef.rows==5)
```

```
                cout<<"- k3: "<<matDistCoef.at<float>(4)<<endl;
            cout << "- p1: " << matDistCoef.at<float>(2) << endl;
            cout << "- p2: " << matDistCoef.at<float>(3) << endl;
            cout << "- bf: " << bf << endl;
            RGB = fSettings["Camera.RGB"];
            scaleFactor = fSettings["ORBextractor.scaleFactor"];
            features = fSettings["ORBextractor.nFeatures"];
            levels = fSettings["ORBextractor.nLevels"];
            minThFAST = fSettings["ORBextractor.minThFAST"];
            iniThFAST = fSettings["ORBextractor.iniThFAST"];
            cout << endl  << "ORB Extractor Parameters: " << endl;
            cout << "- Scale Factor: " << scaleFactor << endl;
            cout << "- Number of Features: " << features << endl;
            cout << "- Scale Levels: " << levels << endl;
            cout << "- Minimum Fast Threshold: "<<minThFAST<<endl;
            cout << "- Initial Fast Threshold: "<<iniThFAST<<endl;
            fSettings.release();
            return true;
    }
}
```

A.12 KeyFrameInitialization.h

orb_slam_2_ros/orb_slam2/include/KeyFrameInitialization.h

```
#ifndef KEYFRAMEINITIALIZATION_H
#define KEYFRAMEINITIALIZATION_H

#include "SystemSetting.h"
#include <opencv2/opencv.hpp>
#include "ORBVocabulary.h"
#include "KeyFrameDatabase.h"
#include "Thirdparty/DBoW2/DBoW2/BowVector.h"
#include "Thirdparty/DBoW2/DBoW2/FeatureVector.h"

namespace ORB_SLAM2
{

```

```
#define FRAME_GRID_ROWS 48
#define FRAME_GRID_COLS 64

class SystemSetting;
class KeyFrameDatabase;

class KeyFrameInitialization
{
public:
    KeyFrameInitialization(SystemSetting &systemSetting);
    void KeyPointsUndistort();
    bool PosInGrid(const cv::KeyPoint& keyPoint,
        int &posX, int &posY);
    void FeaturesToGridAssign();

public:
    ORBVocabulary* vocabulary;
    long unsigned int id;
    double timeStampNum;
    float gridElementWidthInv;
    float gridElementHeightInv;
    std::vector<std::size_t> stdvGrid[FRAME_GRID_COLS]
                                      [FRAME_GRID_ROWS];
    float fbx;
    float fby;
    float x, y;
    float fx, fy;
    float bf;
    float b;
    float thDepth;
    int N;
    std::vector<cv::KeyPoint> stdvKeyPoint;
    std::vector<cv::KeyPoint> stdvKeyPointUn;
    cv::Mat matDescriptors;
    std::vector<float> stdvRight;
    std::vector<float> stdvDepth;
    int scaleLevels;
```

```
    float fcaleFactor;
    float logScaleFactor;
    std::vector<float> scaleFactors;
    std::vector<float> levelSigma2;
    std::vector<float> invLevelSigma2;
    std::vector<float> invScaleFactors;
    int minX, minY, maxX, maxY;
    cv::Mat K;
    cv::Mat matDistCoef;

    DBoW2::BowVector BowVec;
    DBoW2::FeatureVector FeatVec;
};

} //namespace ORB_SLAM2
#endif //KEYFRAMEINITIALIZATION_H
```

A.13 KeyFrameInitialization.cc

orb_slam_2_ros/orb_slam2/src/KeyFrameInitialization.cc

```
#include "KeyFrameInitialization.h"
#include <opencv2/opencv.hpp>
#include "SystemSetting.h"

namespace ORB_SLAM2
{

KeyFrameInitialization::KeyFrameInitialization(
                        SystemSetting &systemSetting):
vocavulary(systemSetting.vocavulary)
{
    fbx = systemSetting.wx;
    fby = systemSetting.wy;
    x = systemSetting.x;
    y = systemSetting.y;
    fx = systemSetting.fx;
    fy = systemSetting.fy;
```

```
    bf = systemSetting.bf;
    b  = systemSetting.b;
    stdvDepth = systemSetting.depth;
    scaleLevels = systemSetting.levels;
    scaleFactor = systemSetting.scaleFactor;
    logScaleFactor = log(systemSetting.scaleFactor);
    scaleFactors.resize(scaleLevels);
    levelSigma2.resize(scaleLevels);
    scaleFactors[0] = 1.0f;
    levelSigma2[0]  = 1.0f;
    for (int i = 1; i < scaleLevels; i++)
    {
        scaleFactors[i] = scaleFactors[i-1]*scaleFactor;
        levelSigma2[i]  = scaleFactors[i]*scaleFactors[i];
    }
    invScaleFactors.resize(scaleLevels);
    invLevelSigma2.resize(scaleLevels);
    for (int i = 0; i < scaleLevels; i++)
    {
        invScaleFactors[i] = 1.0f/scaleFactors[i];
        invLevelSigma2[i]  = 1.0f/levelSigma2[i];
    }
    K = systemSetting.K;
    matDistCoef = systemSetting.matDistCoef;
    if(systemSetting.matDistCoef.at<float>(0)!=0.0)
    {
        cv::Mat mat(4,2,CV_32F);
        mat.at<float>(0,0) = 0.0;
        mat.at<float>(0,1) = 0.0;
        mat.at<float>(1,0) = systemSetting.width;
        mat.at<float>(1,1) = 0.0;
        mat.at<float>(2,0) = 0.0;
        mat.at<float>(2,1) = systemSetting.height;
        mat.at<float>(3,0) = systemSetting.width;
        mat.at<float>(3,1) = systemSetting.height;
        mat = mat.reshape(2);
        cv::undistortPoints(mat, mat, systemSetting.K,
```

```
            systemSetting.matDistCoef, cv::Mat(), systemSetting.K);
        mat = mat.reshape(1);
        minX = min(mat.at<float>(0,0), mat.at<float>(2,0));
        maxX = max(mat.at<float>(1,0), mat.at<float>(3,0));
        minY = min(mat.at<float>(0,1), mat.at<float>(1,1));
        maxY = max(mat.at<float>(2,1), mat.at<float>(3,1));
    }
    else
    {
        minX = 0.0f;
        maxX = systemSetting.width;
        minY = 0.0f;
        maxY = systemSetting.height;
    }
    gridElementWidthInv = static_cast<float(FRAME_GRID_COLS)
        /(maxX-minX);
    gridElementHeightInv = static_cast<float>(FRAME_GRID_ROWS)
        /(maxY-minY);

}

void KeyFrameInitialization::KeyPointsUndistort()
{
    if(matDistCoef.at<float>(0) == 0.0)
    {
        stdvKeyPointUn = stdvKeyPoint;
        return;
    }
    cv::Mat mat(N,2,CV_32F);
    for (int i = 0; i < N; i++)
    {
        mat.at<float>(i,0) = stdvKeyPoint[i].pt.x;
        mat.at<float>(i,1) = stdvKeyPoint[i].pt.y;
    }
    mat = mat.reshape(2);
    cv::undistortPoints(mat,mat,K,matDistCoef,cv::Mat(),K);
    mat = mat.reshape(1);
```

```
    stdvKeyPointUn.resize(N);
    for(int i = 0; i < N; i++)
    {
        cv::KeyPoint keyPoint = stdvKeyPoint[i];
        keyPoint.pt.x = mat.at<float>(i,0);
        keyPoint.pt.y = mat.at<float>(i,1);
        stdvKeyPointUn[i] = keyPoint;
    }
}

void KeyFrameInitialization::FeaturesToGridAssign()
{
    int gReserve = 0.5f*N/(FRAME_GRID_COLS*FRAME_GRID_ROWS);
    for (unsigned int i = 0; i < FRAME_GRID_COLS; i++)
    {
        for (unsigned int j = 0; j < FRAME_GRID_ROWS; j++)
            stdvGrid[i][j].reserve(gReserve);
    }
    for (int i = 0; i < N; i++)
    {
        const cv::KeyPoint& keyPoint = stdvKeyPointUn[i];
        int gridPosX, gridPosY;
    if(PosInGrid(keyPoint, gridPosX, gridPosY))
        stdvGrid[gridPosX][gridPosY].push_back(i);
    }
}

bool KeyFrameInitialization::
    PosInGrid(const cv::KeyPoint &keyPoint,
              int &posX, int &posY)
{
    posX = round((keyPoint.pt.x-minX)*gridElementWidthInv);
    posY = round((keyPoint.pt.y-minY)*gridElementHeightInv);
    if(posX<0 ||
       posX>=FRAME_GRID_COLS ||
       posY<0 ||
       posY>=FRAME_GRID_ROWS)
```

```
        return false;
    return true;
}
}
```

Bibliography

[1] Tao Yang et al. "Monocular vision SLAM-based UAV autonomous landing in emergencies and unknown environments". In: *Electronics* 7.5 (2018), p. 73.

[2] Abdellah Chehri and Hussein T Mouftah. "Autonomous vehicles in the sustainable cities, the beginning of a green adventure". In: *Sustainable Cities and Society* 51 (2019), p. 101751.

[3] Pedro HE Becker, Jose Maria Arnau, and Antonio González. "Demystifying power and performance bottlenecks in autonomous driving systems". In: *2020 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE. 2020, pp. 205–215.

[4] Claudine Badue et al. "Self-driving cars: A survey". In: *Expert Systems with Applications* 165 (2021), p. 113816.

[5] Ashutosh Singandhupe and Hung Manh La. "A review of slam techniques and security in autonomous driving". In: *2019 third IEEE international conference on robotic computing (IRC)*. IEEE. 2019, pp. 602–607.

[6] Cesar Cadena et al. "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age". In: *IEEE Transactions on robotics* 32.6 (2016), pp. 1309–1332.

[7] Supun Kamburugamuve et al. "Cloud-based parallel implementation of slam for mobile robots". In: *Proceedings of the International Conference on Internet of things and Cloud Computing*. 2016, pp. 1–7.

[8] Abu Bakar Sayuti HM Saman and Ahmed Hesham Lotfy. "An implementation of SLAM with extended Kalman filter". In: *2016 6th International Conference on Intelligent and Advanced Systems (ICIAS)*. IEEE. 2016, pp. 1–4.

[9] Stefan Leutenegger et al. "Keyframe-based visual–inertial odometry using nonlinear optimization". In: *The International Journal of Robotics Research* 34.3 (2015), pp. 314–334.

[10] Stephan Weiss et al. "Real-time onboard visual-inertial state estimation and self-calibration of mavs in unknown environments". In: *2012 IEEE international conference on robotics and automation*. IEEE. 2012, pp. 957–964.

[11] Chufan Rui et al. "A multi-sensory blind guidance system based on YOLO and ORB-SLAM". In: *2021 IEEE International Conference on Progress in Informatics and Computing (PIC)*. IEEE. 2021, pp. 409–414.

[12] Cheng Xu et al. "Towards human motion tracking: Multi-sensory IMU/TOA fusion method and fundamental limits". In: *Electronics* 8.2 (2019), p. 142.

[13] Erik Einhorn and Horst-Michael Gross. "Generic 2D/3D SLAM with NDT maps for lifelong application". In: *2013 European Conference on Mobile Robots*. IEEE. 2013, pp. 240–247.

[14] Alexander JB Trevor, John G Rogers, and Henrik I Christensen. "Planar surface SLAM with 3D and 2D sensors". In: *2012 IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 3041–3048.

[15] Keonyong Lee et al. "A practical 2D/3D SLAM using directional patterns of an indoor structure". In: *Intelligent Service Robotics* 11.1 (2018), pp. 1–24.

[16] Si-Jie Yu et al. "3D reconstruction of road surfaces using an integrated multi-sensory approach". In: *Optics and lasers in engineering* 45.7 (2007), pp. 808–818.

[17] Lukas von Stumberg et al. "From monocular SLAM to autonomous drone exploration". In: *2017 European Conference on Mobile Robots (ECMR)*. IEEE. 2017, pp. 1–8.

[18] Tong Qin et al. "Avp-slam: Semantic visual mapping and localization for autonomous vehicles in the parking lot". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 5939–5945.

[19] Guangqiang Li, Lei Yu, and Shumin Fei. "A deep-learning real-time visual SLAM system based on multi-task feature extraction network and self-supervised feature points". In: *Measurement* 168 (2021), p. 108403.

[20] Fangwen Shu et al. "Slam in the field: an evaluation of monocular mapping and localization on challenging dynamic agricultural environment". In: *Proceedings of the IEEE/CVF winter conference on applications of computer vision*. 2021, pp. 1761–1771.

[21] Nader Mahmoud et al. "SLAM based quasi dense reconstruction for minimally invasive surgery scenes". In: *arXiv preprint arXiv:1705.09107* (2017).

[22] Hugh Durrant-Whyte and Tim Bailey. "Simultaneous localization and mapping: part I". In: *IEEE robotics & automation magazine* 13.2 (2006), pp. 99–110.

[23] Haifeng Li et al. “A two-view based multilayer feature graph for robot navigation”. In: *2012 IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 3580–3587.

[24] Novian Habibie et al. “Fruit mapping mobile robot on simulated agricultural area in Gazebo simulator using simultaneous localization and mapping (SLAM)”. In: *2017 International Symposium on Micro-NanoMechatronics and Human Science (MHS)*. IEEE. 2017, pp. 1–7.

[25] Jorge Fuentes-Pacheco, José Ruiz-Ascencio, and Juan Manuel Rendón-Mancha. “Visual simultaneous localization and mapping: a survey”. In: *Artificial intelligence review* 43.1 (2015), pp. 55–81.

[26] TJ Chong et al. “Sensor technologies and simultaneous localization and mapping (SLAM)”. In: *Procedia Computer Science* 76 (2015), pp. 174–179.

[27] Eagle S Jones and Stefano Soatto. “Visual-inertial navigation, mapping and localization: A scalable real-time causal approach”. In: *The International Journal of Robotics Research* 30.4 (2011), pp. 407–430.

[28] Hamid Taheri and Zhao Chun Xia. “SLAM; definition and evolution”. In: *Engineering Applications of Artificial Intelligence* 97 (2021), p. 104032.

[29] Sajad Saeedi et al. “Multiple-robot simultaneous localization and mapping: A review”. In: *Journal of Field Robotics* 33.1 (2016), pp. 3–46.

[30] Randall Smith, Matthew Self, and Peter Cheeseman. “A stochastic map for uncertain spatial relationships”. In: *Proceedings of the 4th international symposium on Robotics Research*. 1988, pp. 467–474.

[31] Ryan M Eustice, Hanumant Singh, and John J Leonard. “Exactly sparse delayed-state filters for view-based SLAM”. In: *IEEE Transactions on Robotics* 22.6 (2006), pp. 1100–1114.

[32] Paul Newman, David Cole, and Kin Ho. “Outdoor SLAM using visual appearance and laser ranging”. In: *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006*. IEEE. 2006, pp. 1180–1187.

[33] Jan Weingarten and Roland Siegwart. “3D SLAM using planar segments”. In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2006, pp. 3062–3067.

[34] Jose E Guivant and Eduardo Mario Nebot. “Optimization of the simultaneous localization and map-building algorithm for real-time implementation”. In: *IEEE transactions on robotics and automation* 17.3 (2001), pp. 242–257.

[35] Jose Guivant and Eduardo Nebot. “Compressed filter for real time implementation of simultaneous localization and map building”. In: *FSR 2001 International Conference on Field and Service Robots*. Vol. 1. 2001, pp. 309–314.

[36] Tim Bailey. “Mobile robot localisation and mapping in extensive outdoor environments”. PhD thesis. Citeseer, 2002.

[37] Andrew J Davison and David W. Murray. “Simultaneous localization and map-building using active vision”. In: *IEEE transactions on pattern analysis and machine intelligence* 24.7 (2002), pp. 865–880.

[38] Paul Newman, John J Leonard, and Richard Rikoski. “Towards constant-time SLAM on an autonomous underwater vehicle using synthetic aperture sonar”. In: *Proceedings of the eleventh international symposium on robotics research, Sienna, Italy*. 2003, pp. 19–22.

[39] Luc Jaulin. “A nonlinear set membership approach for the localization and map building of underwater robots”. In: *IEEE Transactions on Robotics* 25.1 (2009), pp. 88–98.

[40] Eric Westman, Akshay Hinduja, and Michael Kaess. “Feature-based SLAM for imaging sonar with under-constrained landmarks”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 3629–3636.

[41] Matthew R Walter, Ryan M Eustice, and John J Leonard. “Exactly sparse extended information filters for feature-based SLAM”. In: *The International Journal of Robotics Research* 26.4 (2007), pp. 335–359.

[42] Young-Ho Choi, Tae-Kyeong Lee, and Se-Young Oh. “A line feature based SLAM with low grade range sensors using geometric constraints and active exploration for mobile robot”. In: *Autonomous Robots* 24.1 (2008), pp. 13–27.

[43] Liang Zhao, Shoudong Huang, and Gamini Dissanayake. “Linear SLAM: A linear solution to the feature-based and pose graph SLAM based on submap joining”. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2013, pp. 24–30.

[44] Raúl Mur-Artal and Juan D Tardós. “Probabilistic Semi-Dense Mapping from Highly Accurate Feature-Based Monocular SLAM.” In: *Robotics: Science and Systems*. Vol. 2015. Rome. 2015.

[45] Junhao Cheng et al. “DM-SLAM: A feature-based SLAM system for rigid dynamic scenes”. In: *ISPRS International Journal of Geo-Information* 9.4 (2020), p. 202.

[46] Feng Lu and Evangelos E Milios. “Optimal spline fitting to planar shape”. In: *Signal Processing* 37.1 (1994), pp. 129–140.

[47] Feng Lu and Evangelos Milios. “Globally consistent range scan alignment for environment mapping”. In: *Autonomous robots* 4.4 (1997), pp. 333–349.

[48] Dirk Hahnel et al. “An efficient FastSLAM algorithm for generating maps of large-scale cyclic environments from raw laser range measurements”. In: *Pro-*

ceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453). Vol. 1. IEEE. 2003, pp. 206–211.

[49] Sebastian Thrun and Yufeng Liu. “Multi-robot SLAM with sparse extended information filers”. In: *Robotics Research. The Eleventh International Symposium*. Springer. 2005, pp. 254–266.

[50] Ian Mahon et al. “Efficient view-based SLAM using visual loop closures”. In: *IEEE Transactions on Robotics* 24.5 (2008), pp. 1002–1014.

[51] Stefan Kohlbrecher et al. “A flexible and scalable SLAM system with full 3D motion estimation”. In: *2011 IEEE international symposium on safety, security, and rescue robotics*. IEEE. 2011, pp. 155–160.

[52] David Valiente et al. “A comparison of EKF and SGD applied to a view-based SLAM approach with omnidirectional images”. In: *Robotics and Autonomous Systems* 62.2 (2014), pp. 108–119.

[53] Olaf Booij, Zoran Zivkovic, and Ben Kröse. “Efficient data association for view based SLAM using connected dominating sets”. In: *Robotics and Autonomous Systems* 57.12 (2009), pp. 1225–1234.

[54] Jie Li et al. “An overview of the simultaneous localization and mapping on mobile robot”. In: *2012 Proceedings of International Conference on Modelling, Identification and Control*. IEEE. 2012, pp. 358–364.

[55] David Valiente et al. “Improved omnidirectional odometry for a view-based mapping approach”. In: *Sensors* 17.2 (2017), p. 325.

[56] Mark Cummins and Paul Newman. “Appearance-only SLAM at large scale with FAB-MAP 2.0”. In: *The International Journal of Robotics Research* 30.9 (2011), pp. 1100–1123.

[57] Will Maddern, Michael Milford, and Gordon Wyeth. “CAT-SLAM: probabilistic localisation and mapping using a continuous appearance-based trajectory”. In: *The International Journal of Robotics Research* 31.4 (2012), pp. 429–451.

[58] Henrik Andreasson, Tom Duckett, and Achim J Lilienthal. “A minimalistic approach to appearance-based visual SLAM”. In: *IEEE Transactions on Robotics* 24.5 (2008), pp. 991–1001.

[59] Mathieu Labbé and François Michaud. “Memory management for real-time appearance-based loop closure detection”. In: *2011 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2011, pp. 1271–1276.

[60] Kiana Hajebi and Hong Zhang. “An efficient index for visual search in appearance-based SLAM”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 353–358.

[61] Sheraz Khan and Dirk Wollherr. "IBuILD: Incremental bag of binary words for appearance based loop closure detection". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 5441–5447.

[62] Ravi Kaushik et al. "Polygon-based 3D scan registration with dual-robots in structured indoor environments". In: *International Journal of Robotics and Automation* 27.1 (2012), p. 101.

[63] Dirk Hähnel, Wolfram Burgard, and Sebastian Thrun. "Learning compact 3D models of indoor and outdoor environments with a mobile robot". In: *Robotics and Autonomous Systems* 44.1 (2003), pp. 15–27.

[64] Johann Dichtl et al. "Polyslam: A 2d polygon-based SLAM algorithm". In: *2019 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*. IEEE. 2019, pp. 1–6.

[65] Kaustubh Pathak et al. "Online three-dimensional SLAM by registration of large planar surface segments and closed-form pose-graph relaxation". In: *Journal of Field Robotics* 27.1 (2010), pp. 52–84.

[66] Josep Aulinas et al. "The SLAM problem: a survey". In: *Artificial Intelligence Research and Development* (2008), pp. 363–371.

[67] John Mullane et al. "A random-finite-set approach to Bayesian SLAM". In: *IEEE transactions on robotics* 27.2 (2011), pp. 268–282.

[68] Austin Eliazar and Ronald Parr. "DP-SLAM: Fast, robust simultaneous localization and mapping without predetermined landmarks". In: *IJCAI*. Vol. 3. Citeseer. 2003, pp. 1135–1142.

[69] Michael Montemerlo et al. "FastSLAM: A factored solution to the simultaneous localization and mapping problem". In: *Aaai/iaai* 593598 (2002).

[70] Giorgio Grisetti et al. "Fast and accurate SLAM with Rao–Blackwellized particle filters". In: *Robotics and Autonomous Systems* 55.1 (2007), pp. 30–38.

[71] Peter Karkus, Shaojun Cai, and David Hsu. "Differentiable slam-net: Learning particle slam for visual navigation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 2815–2825.

[72] Guillem Vallicrosa Massaguer and Pere Ridao Rodríguez. "H-SLAM: Rao-Blackwellized particle filter SLAM using Hilbert Maps". In: *Sensors (Switzerland), 2018, vol. 18, núm. 5, p. 1386* (2018).

[73] Ronald PS Mahler. *Statistical multisource-multitarget information fusion*. Vol. 685. Artech House Norwood, MA, USA, 2007.

[74] Martin Adams, John Mullane, and Ba-Ngu Vo. "Circumventing the feature association problem in SLAM". In: *IEEE Intelligent transportation systems magazine* 5.3 (2013), pp. 40–58.

[75] Lin Gao, Giorgio Battistelli, and Luigi Chisci. "Random-finite-set-based distributed multirobot SLAM". In: *IEEE Transactions on Robotics* 36.6 (2020), pp. 1758–1777.

[76] Egidio D'Amato et al. "UAV sensor FDI in duplex attitude estimation architectures using a set-based approach". In: *IEEE Transactions on Instrumentation and Measurement* 67.10 (2018), pp. 2465–2475.

[77] Keith YK Leung, Felipe Inostroza, and Martin Adams. "Evaluating set measurement likelihoods in random-finite-set slam". In: *17th International Conference on Information Fusion (FUSION)*. IEEE. 2014, pp. 1–8.

[78] Martin Adams et al. "SLAM gets a PHD: New concepts in map estimation". In: *IEEE Robotics & Automation Magazine* 21.2 (2014), pp. 26–37.

[79] J-S Gutmann and Kurt Konolige. "Incremental mapping of large cyclic environments". In: *Proceedings 1999 IEEE International Symposium on Computational Intelligence in Robotics and Automation. CIRA'99 (Cat. No. 99EX375)*. IEEE. 1999, pp. 318–325.

[80] Andrew Howard, Maja J Mataric, and Gaurav Sukhatme. "Relaxation on a mesh: a formalism for generalized localization". In: *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No. 01CH37180)*. Vol. 2. IEEE. 2001, pp. 1055–1060.

[81] Udo Frese, Per Larsson, and Tom Duckett. "A multilevel relaxation algorithm for simultaneous localization and mapping". In: *IEEE Transactions on Robotics* 21.2 (2005), pp. 196–207.

[82] Edwin Olson, John Leonard, and Seth Teller. "Fast iterative alignment of pose graphs with poor initial estimates". In: *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* IEEE. 2006, pp. 2262–2269.

[83] Giorgio Grisetti et al. "A tutorial on graph-based SLAM". In: *IEEE Intelligent Transportation Systems Magazine* 2.4 (2010), pp. 31–43.

[84] Nicholas Carlevaris-Bianco, Michael Kaess, and Ryan M Eustice. "Generic node removal for factor-graph SLAM". In: *IEEE Transactions on Robotics* 30.6 (2014), pp. 1371–1385.

[85] Ernesto Estrada and Juan A Rodriguez-Velazquez. "Subgraph centrality in complex networks". In: *Physical Review E* 71.5 (2005), p. 056103.

[86] Kai Ni, Drew Steedly, and Frank Dellaert. "Out-of-core bundle adjustment for large-scale 3d reconstruction". In: *2007 IEEE 11th International Conference on Computer Vision*. IEEE. 2007, pp. 1–8.

[87] Michael Bosse and Robert Zlot. "Map matching and data association for large-scale two-dimensional laser scan-based slam". In: *The International Journal of Robotics Research* 27.6 (2008), pp. 667–691.

[88] Nathaniel Fairfield, David Wettergreen, and George Kantor. "Segmented SLAM in three-dimensional environments". In: *Journal of Field Robotics* 27.1 (2010), pp. 85–103.

[89] Jan Oberländer, Arne Roennau, and Rüdiger Dillmann. "Hierarchical SLAM using spectral submap matching with opportunities for long-term operation". In: *2013 16th international conference on advanced robotics (ICAR)*. IEEE. 2013, pp. 1–7.

[90] Christoph Brand et al. "Submap matching for stereo-vision based indoor/outdoor SLAM". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2015, pp. 5670–5677.

[91] Kamak Ebadi et al. "LAMP: Large-scale autonomous mapping and positioning for exploration of perceptually-degraded subterranean environments". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 80–86.

[92] Gordon Wyeth and Michael Milford. "Spatial cognition for robots". In: *IEEE Robotics & Automation Magazine* 16.3 (2009), pp. 24–32.

[93] Xiwu Zhang, Yan Su, and Xinhua Zhu. "Loop closure detection for visual SLAM systems using convolutional neural network". In: *2017 23rd International Conference on Automation and Computing (ICAC)*. IEEE. 2017, pp. 1–6.

[94] Grigorios Mingas, Emmanouil Tsardoulias, and Loukas Petrou. "An FPGA implementation of the SMG-SLAM algorithm". In: *Microprocessors and Microsystems* 36.3 (2012), pp. 190–204.

[95] Chao Duan et al. "Deep learning for visual SLAM in transportation robotics: a review". In: *Transportation Safety and Environment* 1.3 (2019), pp. 177–184.

[96] Xiang Dong et al. "FSD-SLAM: a fast semi-direct SLAM algorithm". In: *Complex & Intelligent Systems* 8.3 (2022), pp. 1823–1834.

[97] Andrew J Davison et al. "MonoSLAM: Real-time single camera SLAM". In: *IEEE transactions on pattern analysis and machine intelligence* 29.6 (2007), pp. 1052–1067.

[98] Gui-Hai Li and Song-Lin Chen. "Visual Slam in Dynamic Scenes Based on Object Tracking and Static Points Detection". In: *Journal of Intelligent & Robotic Systems* 104.2 (2022), pp. 1–10.

[99] Christine Evers and Patrick A Naylor. “Optimized self-localization for SLAM in dynamic scenes using probability hypothesis density filters”. In: *IEEE Transactions on Signal Processing* 66.4 (2017), pp. 863–878.

[100] Zhongqun Zhang, Jingtao Zhang, and Qirong Tang. “Mask R-CNN based semantic RGB-D SLAM for dynamic scenes”. In: *2019 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*. IEEE. 2019, pp. 1151–1156.

[101] Mehul Arora et al. “Mapping the static parts of dynamic scenes from 3D LiDAR point clouds exploiting ground segmentation”. In: *2021 European Conference on Mobile Robots (ECMR)*. IEEE. 2021, pp. 1–6.

[102] Yu Liu, Yilin Wu, and Wenzhao Pan. “Dynamic RGB-D SLAM based on static probability and observation number”. In: *IEEE Transactions on Instrumentation and Measurement* 70 (2021), pp. 1–11.

[103] Arturo Gil et al. “A comparative evaluation of interest point detectors and local descriptors for visual SLAM”. In: *Machine Vision and Applications* 21.6 (2010), pp. 905–920.

[104] Tran Duc Dung et al. “Multifeature image indexing for robot localization in textureless environments”. In: *Robotics* 8.2 (2019), p. 37.

[105] Ziyun Cai et al. “RGB-D datasets using microsoft kinect or similar sensors: a survey”. In: *Multimedia Tools and Applications* 76.3 (2017), pp. 4313–4355.

[106] Qinxuan Sun et al. “Plane-Edge-SLAM: Seamless fusion of planes and edges for SLAM in indoor environments”. In: *IEEE Transactions on Automation Science and Engineering* 18.4 (2020), pp. 2061–2075.

[107] Wenshan Wang et al. “Tartanair: A dataset to push the limits of visual slam”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 4909–4916.

[108] Huizhong Zhou et al. “StructSLAM: Visual SLAM with building structure lines”. In: *IEEE Transactions on Vehicular Technology* 64.4 (2015), pp. 1364–1375.

[109] Ziang Cheng et al. “Multi-view 3D Reconstruction of a Texture-less Smooth Surface of Unknown Generic Reflectance”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 16226–16235.

[110] Khalid Yousif, Alireza Bab-Hadiashar, and Reza Hoseinnezhad. “3D SLAM in texture-less environments using rank order statistics”. In: *Robotica* 35.4 (2017), pp. 809–831.

[111] Han Jiale, Zhang Jingmei, and Zhao Yongqiang. “Non-textured objects visual SLAM using pola-rization 3D reconstruction”. In: *Array* 10 (2021), p. 100066.

[112] Luwei Yang et al. "Polarimetric dense monocular slam". In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2018, pp. 3857–3866.

[113] Yuichi Taguchi et al. "Point-plane SLAM for hand-held 3D sensors". In: *2013 IEEE international conference on robotics and automation.* IEEE. 2013, pp. 5182–5189.

[114] Alejo Concha and Javier Civera. "Using superpixels in monocular SLAM". In: *2014 IEEE international conference on robotics and automation (ICRA).* IEEE. 2014, pp. 365–372.

[115] Alejo Concha and Javier Civera. "DPPTAM: Dense piecewise planar tracking and mapping from a monocular sequence". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* IEEE. 2015, pp. 5686–5693.

[116] Ruihao Li et al. "A novel RGB-D SLAM algorithm based on points and plane-patches". In: *2016 IEEE International Conference on Automation Science and Engineering (CASE).* IEEE. 2016, pp. 1348–1353.

[117] Yuan-Peng Yu. "Orb-Slam Examination in the Indoor Textureless Environment". PhD thesis. 2017.

[118] Felix Nobis et al. "Persistent map saving for visual localization for autonomous vehicles: An orb-slam 2 extension". In: *2020 Fifteenth International Conference on Ecological Vehicles and Renewable Energies (EVER).* IEEE. 2020, pp. 1–9.

[119] Nicolas Ragot et al. "Benchmark of visual slam algorithms: Orb-slam2 vs rtab-map". In: *2019 Eighth International Conference on Emerging Security Technologies (EST).* IEEE. 2019, pp. 1–6.

[120] Raul Mur-Artal and Juan D Tardós. "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras". In: *IEEE transactions on robotics* 33.5 (2017), pp. 1255–1262.

[121] Xu Cui, Chenggang Lu, and Jinxiang Wang. "3D semantic map construction using improved ORB-SLAM2 for mobile robot in edge computing environment". In: *IEEE Access* 8 (2020), pp. 67179–67191.

[122] Lina M Paz et al. "Large-scale 6-DOF SLAM with stereo-in-hand". In: *IEEE transactions on robotics* 24.5 (2008), pp. 946–957.

[123] Junchao Zhu et al. "A New System to Construct Dense Map with Pyramid Stereo Matching Network and ORB-SLAM2". In: *2021 7th International Conference on Computer and Communications (ICCC).* IEEE. 2021, pp. 430–435.