A Tutorial: Mobile Robotics, SLAM, Bayesian Filter, Keyframe Bundle Adjustment and ROS Applications

Muhammet Fatih Aslan, Akif Durdu, Abdullah Yusefi, Kadir Sabanci, and Cemil Sungur

Abstract Autonomous mobile robots, an important research topic today, are often developed for smart industrial environments where they interact with humans. For autonomous movement of a mobile robot in an unknown environment, mobile robots must solve three main problems; localization, mapping and path planning. Robust path planning depends on successful localization and mapping. Both problems can be overcome with Simultaneous Localization and Mapping (SLAM) techniques. Since sequential sensor information is required for SLAM, eliminating these sensor noises is crucial for the next measurement and prediction. Recursive Bayesian filter is a statistical method used for sequential state prediction. Therefore, it is an essential method for the autonomous mobile robots and SLAM techniques. This study deals with the relationship between SLAM and Bayes methods for autonomous robots. Additionally, keyframe Bundle Adjustment (BA) based SLAM, which includes state-of-art methods, is also investigated. SLAM is an active research area and new algorithms are constantly being developed to increase accuracy rates, so new researchers need to understand this issue with ease. This study is a detailed and easily understandable resource for new SLAM researchers. ROS (Robot Operating System)-based SLAM applications are also given for better understanding. In this way, the

M. F. Aslan (✉) · K. Sabanci
Robotics Automation Control Laboratory (RAC-LAB), Electrical and Electronics Engineering Karamanoglu Mehmetbey University, Karaman, Turkey
e-mail: mfatihaslan@kmu.edu.tr

K. Sabanci
e-mail: kadirsabanci@kmu.edu.tr

A. Durdu · C. Sungur
Robotics Automation Control Laboratory (RAC-LAB), Electrical and Electronics Engineering Konya Technical University, Konya, Turkey
e-mail: adurdu@ktun.edu.tr

C. Sungur
e-mail: csungur@ktun.edu.tr

A. Yusefi
Robotics Automation Control Laboratory (RAC-LAB), Computer Engineering Konya Technical University, Konya, Turkey
e-mail: a.yusefi1991@gmail.com

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2021

A. Koubaa (ed.), *Robot Operating System (ROS)*, Studies in Computational Intelligence 962, https://doi.org/10.1007/978-3-030-75472-3_7

reader obtains the theoretical basis and application experience to develop alternative methods related to SLAM.

Keywords Bayes filter · ROS · SLAM · Tutorial

1 Introduction

The robot is an electro-mechanical device that can perform certain pre-programmed tasks or autonomous tasks. Robots that perform pre-programmed tasks are generally used in industrial product production. The rapid production of industrial products today is thanks to the changes and developments in industrial automation. Industrial automation owes this development process to Computer Aided Design (CAD) systems and Computer Aided Manufacturing (CAM) systems. The resulting industrial robots have a decisive role in the world economy by providing high speed and high accuracy for mass production. With the Industry 4.0 revolution in the future, industrial robots will assist production in the unmanned factory. By the way, the production will be faster and cheaper. In addition, an employed person is now more costly and operates under certain business risks. On the other hand, with the advancement of technology, the robot costs decrease gradually, the efficiency of the work done by the robot increases and unwanted situations such as industrial accident do not occur. In the operation of these types of industrial robots, the mechanical parts are more prominent. Because these robots are generally designed specifically for the application area and, the software is only required for certain tasks. Examples of industrial robots are manipulators such as welding robot, painting robot, assembly robot and packaging robot. These robots having three or more axes are fixed at one point. Therefore, industrial robots generally do not have mobility and autonomous features. However, different applications require the robots to be autonomous and mobile [1, 2].

Advances in software technology and increasing processor speeds have accelerated mobile autonomous robotics applications. In recent years, unlike robots used for industrial purposes in factories and assembly lines, robots have also been used in homes and offices. These robots, called service robots, should be able to perform their tasks autonomously in an unstructured environment while sharing task areas with people. Of course, the basic condition of the independent movement is the mobile movement. Usually, in designs and works related to this, people and animals are an inspiration. Thanks to the mobile capability provided to the robot, the robot can change its position. To achieve this, studies such as wheeled robots [3], air robots [4], legged robots [5], floating robots [6] have been carried out until now.

In mobile robotics, wheeled robots are most preferred due to both easier application and wider application area. However, today, Unmanned Aerial Vehicles (UAV) have started to attract great attention. In general, the purpose of mobile robots such as wheeled robots and UAVs is to perform mobile activities and tasks like humans. For this reason, human-oriented studies that resemble people, and can make decisions

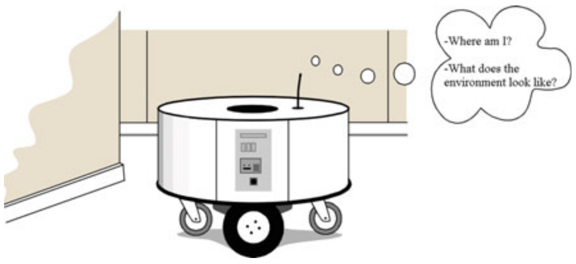


Fig. 1 A robot that moves in an unknown environment [2]

by thinking like a human, have been a very active field of research. A person can make a completely independent decision, and as a result, perceive his/her environment and take an action. In addition, people not only use their previous knowledge at the decision stage, but also tend to learn new information. Based on this, various mobile robots are designed to perform surveillance [7], museum guides [8], shopping assistant [9], home security [10], transportation [11], etc. [12]. However, the desired target across such studies is the fully autonomous operation of the system. In other words, the robot must recognize its environment, find its own position according to the environment, and finally perform the task assigned to it. Therefore, a mobile autonomous robot should be able to answer the questions: "Where am I?" (Localization) and "What does the environment look like?" (Mapping) (see Fig. 1) [2].

In order to solve the questions in Fig. 1, the mobile robot must have hardware and software components. Figure 2 contains the components of any robot that can move autonomously in an unstructured environment. What is expected from a robot designed as in Fig. 2 is a successful navigation. In navigation, the robot is expected to make a series of decisions (e.g., turn left, then turn right) to reach a target point. For this, the robot needs to perceive useful information in the environment. Thanks to its high perception potential, the robot learns the structure of the environment and localizes itself. However, this perception is carried out not only once, but many times until the map of the environment is learned. As a result, tasks called mapping and localization are completed. Various sensors or cameras can be used for perception. According to the perceived information, the information should be sent to the actuators for the movement of the robot. A microcontroller can also be used for this process. The optimum path should be chosen for successful navigation. In other words, path planning is required for active localization and mapping. For this reason, the topics of localization and mapping are very important for navigation.

The questions asked by the robot in Fig. 1 has been an important problem for mobile robotics for a long time. However, it is still an active research topic due to its necessity and difficulty [13]. The reason why it is difficult is that the robot creates the map in an environment without any prior knowledge and localizes itself according to this map. There is also a close relationship between mapping and localization. Therefore, a mobile robot must perform both tasks simultaneously. In fact, addressing both problems separately reduces the complexity and the difficulty considerably.

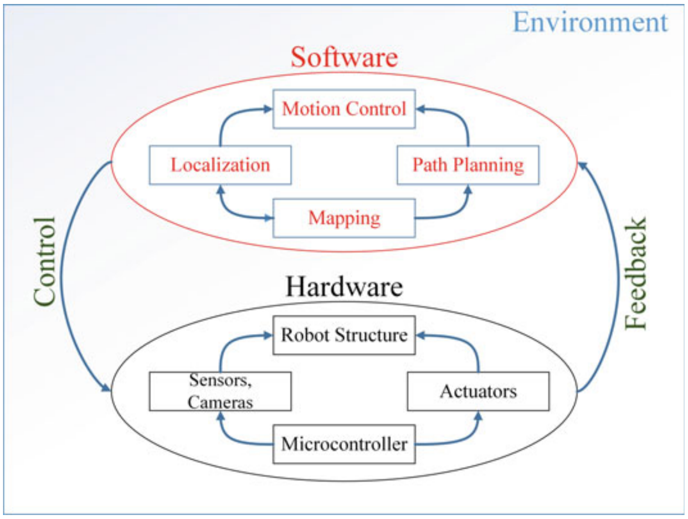


Fig. 2 Component of an autonomous mobile robot in unstructured environment

Because when one problem is dealt with, it is assumed that the other problem is known. However, for an unknown environment, as the robot moves, the localization and mapping information should be created in an iterative way as a result of the perceptions made by the robot. Since localization and mapping are closely related to each other and both information is needed simultaneously, this problem is discussed under a separate title as Simultaneous Localization and Mapping (SLAM) in the literature.

SLAM contains many application areas. Generally, in environments where the Global Positioning System (GPS) is insufficient, the location is tried to be found through SLAM methods. SLAM is frequently preferred in indoor—outdoor environments, submarine [14], space [15], underground [16] studies and augmented reality applications [17]. In addition, autonomous indoor mapping using mobile robots is a necessary tool in areas where human access may be restricted due to area conditions or security reasons. Considering the application areas, autonomous mobile robots need the SLAM method with high accuracy. In addition, service robots, which are becoming more and more important nowadays, must be able to perceive their environment strongly and perform tasks autonomously [18]. In particular, applications with autonomous UAV, which is becoming widespread today, also need a powerful SLAM algorithm [19, 20].

In most studies for SLAM, various solutions were offered using sonar sensors, LIDAR, IR sensors and laser scanners [21]. Recently, Visual SLAM (VSLAM) becomes prominent due to the fact that color, texture and other visual elements obtained from low-cost cameras compared to GPS, laser, ultrasonic and sonar sensors contain richer visual information. On the other hand, solutions with VSLAM brought complex algorithms such as computational difficulty, image processing, optimization and feature extraction. However, nowadays there are high-performance computers and boards that run the training algorithm using thousands of images for deep learning. Moreover, thanks to the Robot Operating System (ROS), time-consuming applications are realized in real-time. Such a study was presented by Giubilato, et al. [22].

Due to the advantages of VSLAM over traditional SLAM, VSLAM's work has accelerated. Two methodologies, filtering and Bundle Adjustment (BA), are common in studies for VSLAM. Filtering-based approaches require fewer computational resources due to the continuous marginalization of the past state, but its accuracy is low due to the linearization error. BA is a nonlinear batch optimization method that uses feature locations, camera locations, and real camera parameters. Strasdat, et al. [23] showed that BA-based approaches provide better accuracy than filtering approaches. This is because optimization-based methods re-linearize in each iteration to avoid the error stemming from the integral resulting from linearization. This provides an advantage in terms of accuracy, but reduces the processing speed. BA methods should constantly optimize the predicted value to approach an optimal value, i.e. it is iterative. Since the update in the filtering method is equivalent to one iteration of the BA state estimate, the BA methods are still not as effective as filtering-based methods in terms of time. For this, various studies have been carried out to increase the accuracy of Extended Kalman Filter (EKF)-based SLAM (EKF-SLAM) [24, 25], which is one of the most prominent filtering methods [26]. However, with the keyframe BA study (Parallel Tracking and Mapping (PTAM)) applied for the first time by Klein and Murray [27], BA methods came to the fore. Strasdat, et al. [28] stated that keyframe BA-based methods are better than filter-based methods. After that, new state-of-art methods are proposed for monocular SLAM systems, mostly based on keyframe BA, which are very advantageous in terms of speed and computational cost. ORB-SLAM [29], Large Scale Direct monocular SLAM (LSD-SLAM) [30], Semi-direct Visual Odometry (SVO) [31], etc. are some of them.

The development of SLAM applications is achieved through advances in software rather than hardware. Especially with the emergence of VSLAM, studies with lower costs and easier design but more complex software gained importance. This increased the inclination to different programming languages used in computer science. The most preferred languages for SLAM application can be given as C, C ++, Python, MATLAB. Among them, C ++ and Python are favorites because of their speed capability and open source software availability. Of course, there are different Integrated Development Environment (IDE) software applications for the development of these languages. IDEs provide great convenience to programmers for software development. Especially the most preferred IDEs for robotic applications can be given as Visual Studio Code (VS Code), Eclipse, PyCharm, Spyder, IDLE. In robotic-based applications, the environment in which the software is developed is important. Therefore, the operating system on which the programs are run has a major impact on application performance. Robotic complex tasks that usually need to run in real time are performed on the Linux operating system rather than the Windows. Especially because ROS is Linux compatible, the Linux operating system is more preferred in applications such as SLAM, manipulator, robot modeling.

Many methods have been proposed for SLAM so far. Despite significant improvements, there are still major challenges, so it needs improvement. Due to the fact that it is still an active field and its importance, SLAM attracts the attention of new researchers. However, for a new researcher reviewing the studies, SLAM studies are highly complex and cause confusion, because of many different proposed methods in the literature [32]. A study showing the many different SLAM methods proposed so far was done by Huang, et al. [33]. At this point, one of the goals in this is to prepare a basic tutorial for a new researcher. For this purpose, the SLAM problem has been introduced comprehensively in this study. In addition, the probabilistic Bayesian filter, which forms the basis of filtering based methods, is explained in detail. In this context, the concept of uncertainty, Bayes rule and recursive Bayes filter are explained with a simple expression style. In addition, theoretical information for keyframe BA based SLAM, which includes state-of-art methods, is explained under a separate title. Finally, using ROS, which offers a practical workspace for SLAM, sample SLAM applications are explained. This study is a tutorial for new researchers or students working in a similar field.

In fact, various introduction, tutorial and survey studies related to SLAM have been carried out so far. The tutorial work for SLAM by Durrant-Whyte and Bailey [34] has attracted a lot of attention from readers and researchers. First, the history of the SLAM problem is explained, then Bayes-based SLAM methods and finally the implementation of SLAM is discussed. Stachniss, et al. [13] briefly introduced the SLAM problem and explained the SLAM methods based on EKF, Particle Filter (PF) and graph optimization. In that study, SLAM problem was explained superficially, then the methods used in SLAM problem solution and various applications were mentioned. Sola [35] presented a study on the application of EKF-SLAM in MATLAB environment. First, a general and brief information about SLAM and EKF-SLAM was given, then coding was focused and finally, sample codes were shared. Grisetti, et al. [36] conducted a comprehensive tutorial on Graph-based SLAM. Huang, et al. [33] briefly introduced numerous recommended SLAM methods such as traditional SLAM, VSLAM, deep learning based SLAM. Studies by Taketomi, et al. [37] and Fuentes-Pacheco, et al. [38] are also survey studies that summarize the work done for VSLAM. Unlike the above-mentioned studies, our study provides a more comprehensive introduction and tutorial information for mobile robotic, filter-based SLAM and keyframe BA-based SLAM. This paper also gives detailed and easy-to-understand information about the recursive Bayesian filter. At the end of the study, this theoretical knowledge is strengthened by application in ROS environment. Therefore, this work differs from previous studies in terms of content and provides the reader with satisfactory information in the SLAM field.

2 SLAM

SLAM is the simultaneous estimation of a robot's location and environment. However, the simultaneous prediction in this easy definition makes SLAM a difficult topic. In order to understand the topic easily, certain terms must be known.

2.1 Related Terms

- **Mobile Robot**: Mobile robot is the device equipped with sensors and moves throughout the environment. This movement can be achieved by legs, wheels, wings, or something else.
- **Landmarks**: Landmarks are features that can be easily observed and distinguished from the environment.
- **State Estimation**: The state represents a variable that needs to be measured in relation to the application performed. Based on this study, the position of the robot or the location of a landmark can be considered as a state. The aim is to obtain an estimation related to these states. In fact, states can often be measured using various sensors. However, since a perfect measurement is not possible, the state is estimated. Because the data measured from the sensor always contain noise. Even if the noise is very small, it accumulates over time and generates huge errors. As long as the uncertainty or noise of consecutive measurements is not removed, the measurements taken become increasingly distant from the actual value. Therefore, after each measurement, an estimate should be made for the next measurement, taking into account the uncertainty. That is, a statistical estimate should be produced for each measured value. For this, recursive Bayes filter is used.
- **Localization**: Localization is actually part of the state estimation application for SLAM. Its purpose is to determine the location of the vehicle, robot or sensor. If only localization is available, the map of the environment is assumed to be fully known. Because the robot in an environment is localized according to the map.
- **Mapping**: Mapping is the prediction of the model of the environment by using the data received through the sensors on the mobile robot. Due to the noisy measurements, mapping is also a state prediction problem for SLAM. If an application only performs mapping task, it is assumed that localization is well known. For example, a robot whose position information is well known can measure the distance of close objects from its position using the laser range finder.
- **SLAM**: If localization will be performed and the environmental map is unknown or mapping will be performed and the position of the robot is unknown, this task is called SLAM. Both localization and mapping are very interdependent, if one is unknown the other is unpredictable. Therefore, in an application lacking both information, these two state estimates should be calculated simultaneously.

– **Navigation**: After a successful SLAM, navigation is a series of motion decisions made by a robot that knows its location and environmental map to go to the target point. Usually a successful SLAM provides better navigation.

2.2 Uncertainty in Robotic Applications

One of the factors that makes SLAM difficult is the uncertainty in the measurements. Although there are various sensors used for localization or mapping, no sensor provides an excellent measurement. In particular, in sequential measurements, the noises increase cumulatively and very high errors occur. Therefore, in real-time robotic applications in an unstructured environment, uncertainty is a problem that needs to be solved. Some situations causing this uncertainty are listed below.

– Sensors are not perfect.
– The robot cannot perform the expected movement.
– Events in the environment where the robot is located are not always predictable.
– It is not possible to model the robot path and environment perfectly.

Statistical-based methods are used to overcome these uncertainties. In these methods, the information available is expressed not by a single value, but by a probability distribution. Generally, Bayesian methods are used to make a statistical inference and decision making under uncertainty.

2.3 Definition of SLAM Problem

In fact, localization and mapping are separate problems. However, in autonomous mobile robot applications, it is often necessary to deal with both problems at the same time. Figure 3 shows the localization, mapping and the SLAM problem, which is a combination of the two.

Figure 3 shows the movement and perceptions of the robot at certain time intervals. The stars represent landmarks in the environment. In other words, the robot can recognize these landmarks in the environment and, accordingly, make an estimate about its location. The estimates made are shown in gray. Accordingly, in Fig. 3a, the robot tries to localize itself while it is moving. But in the second case, although the robot actually moved to the right, the robot thought it was moving to the left. The cause of this error may be wheel or ground conditions. Then, as a result of the observation made with the robot, it is measured that the robot is a certain distance from the landmark. However, considering the incorrectly measured robot position, a landmark at this distance is not encountered. At this point, it is understood that the position of the robot is measured incorrectly and a correction is made according to the position of the landmark. In other words, since the map (landmark locations) are

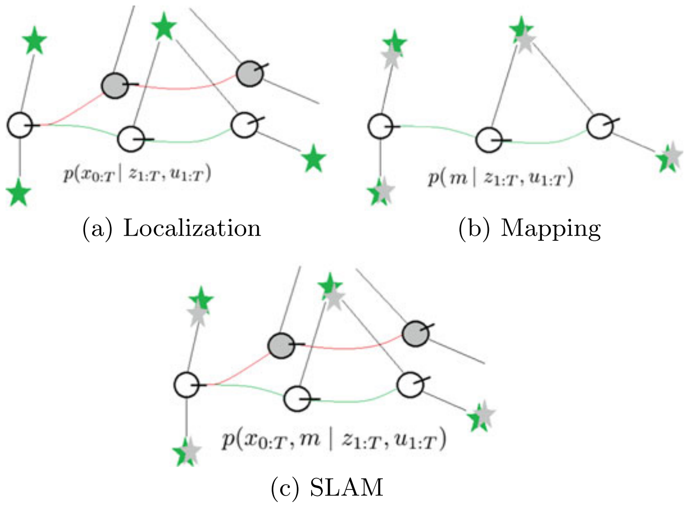
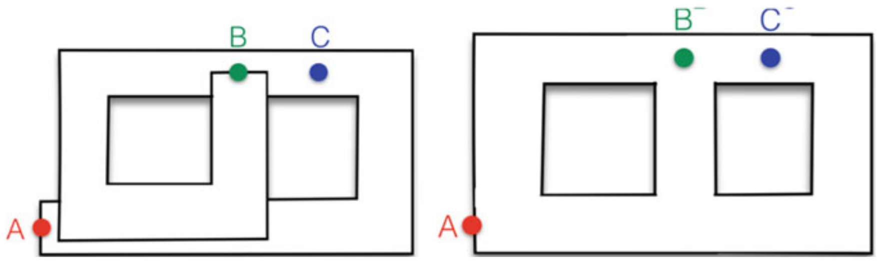

$p(x_{0:T} \mid z_{1:T}, u_{1:T})$
(a) Localization
$p(m \mid z_{1:T}, u_{1:T})$
(b) Mapping
$p(x_{0:T}, m \mid z_{1:T}, u_{1:T})$
(c) SLAM

Fig. 3 Structure of localization, mapping and SLAM problem

known, a more accurate localization estimation can be made by correcting the wrong measurement. The opposite of this is also true. That is, because the position of the robot is well known in the mapping problem shown in Fig. 3b, incorrectly measured landmark positions are corrected according to the robot position. Hence, localization and mapping are closely related. Therefore, the error in the measurement can be corrected by a known value associated with that measurement. However, both value is not known in SLAM, and another is needed to estimate one value. Therefore, as seen in Fig. 3c, the SLAM problem is more difficult and the error of the estimates is higher.

In the first applications related to localization and mapping, wheel encoders were used to determine the distance traveled. However, especially on uneven terrain or slippery surfaces, the position estimate obtained from the wheel odometry quickly deviates due to wheel slippage and this estimate becomes unusable after a few meters. Due to these disadvantages, other positioning strategies such as Inertial Measurement Units (IMU), GPS, laser odometry, Visual Odometry (VO) and SLAM have been proposed. VO is defined as the process of estimating the movement and orientation of the robot according to the reference frame by observing the images of its environment. VO is more advantageous than IMU and Laser in terms of cost. However, it may be subject to slipping problem in wheel odometry. In VO, the robot path is gradually (pose after pose) estimated based on the robot position. In SLAM, robot trajectory and map estimation are aimed. While the estimation in VO is local, there is a global estimation in SLAM. In this way, in SLAM, the loop can be closed by detecting that the robot has come back to a previously mapped place (loop closure detection, see Fig. 6). This information reduces the deviation in the estimates. This is one of the most important reasons for the development and importance of SLAM. With the



(a) Path of the robot created with VO

(b) Path of the robot created with SLAM

Fig. 4 VO and SLAM difference [39]

Table 1 Given and wanted for SLAM problem

Given	Wanted
The robot's control commands $\bullet u_{1:T} = \{u_1, u_2, u_3, \ldots, u_T\}$	Map of the environment $\bullet m$
Observations perceived by sensors $\bullet z_{1:T} = \{z_1, z_2, z_3, \ldots, z_T\}$	Locations of the robot $\bullet x_{0:T} = \{x_0, x_1, x_2, \ldots, x_T\}$

loop closure detection, the robot understands the real topology of the environment and can find shortcuts between different locations (e.g. B and C points in Fig. 4). An image showing the importance of loop closure is shown in Fig. 4. In Fig. 4, a robot started moving from point A and passed through point C and ended its movement at point B. Although B is close to C, according to VO, these two points are independent from each other. In SLAM, the intersection between two close points, B and C, is detected and these points are combined [39–41].

One of the most efficient solution methods for all three problems shown in Fig. 3 is to perform statistical based prediction. Because in order to represent uncertainty, the measured value must be expressed as probabilistic. The conditional probability equation under the images in Fig. 3 expresses this. The following questions should be asked to create these equations: "What do we have?" and "What do we want?" With the answer to these questions, conditional probabilistic equations are obtained. Then, various Bayes-based filters are used to solve this conditional probability. Considering the SLAM problem, the "given" and "wanted" values are shown in the Table 1 [42].

The u in Table 1 represents the control command. Control commands are motion or orientation information sent to the robot. For example, u_t can be commands like "go 1 m forward", "turn left", "stop". These commands can be sent remotely to the robot or can be obtained with information from the robot. In general, movement information is achieved with the information received from the robot. Because a command sent to the robot cannot be performed perfectly by the robot. For example, as a result of the command "go 2 m forward" sent to the robot remotely, the robot can move 1.95 m. However, a system that measures the number of revolutions of the wheel can more accurately calculate how far the robot is moving. This system is called odometry. Therefore, odometry information can be used as a command sent to the robot. z stands for measurements taken from sensors connected to the robot. This value is a given value because it is measured directly through sensors. For example, these z values can be obtained from a laser sensor that gives the distance information of the nearest obstacle. Or as a result of a camera used instead of a sensor, z values can visually represent the distance of the landmarks in a frame. As a result, z values represent values for measuring the location of landmarks. m represents the map of the environment. These maps are created according to objects such as landmarks, obstacles, etc. in the environment. So m represents the positions of these landmarks or obstacles. If all of these locations are found, a map of the environment is obtained based on the robot's location. Finally, x refers to the robot's coordinates at specific time intervals. The robot goes to these x locations with u commands and estimates the m values by making z measurements according to its location.

The reason for estimating the m and x values is that, as mentioned earlier, the u and z values contain uncertainty or noise. An accurate prediction is possible by reducing or eliminating this uncertainty. This uncertainty, which is the main problem for SLAM, is represented by statistical approaches. Figure 5 shows how the position of a robot is represented. In Fig. 5a uncertainty is neglected and the robot position is determined deterministically. Figure 5b, on the other hand, expresses uncertainty as probabilistic. Based on real-world measurements, all sensors contain errors. So no sensor can measure perfectly where the robot is. Even if the sensor error is too small to be negligible, this error accumulates over time. Therefore, the information received from the sensor should cover a range of values, not a single value. In this context, Fig. 5b provides a more accurate representation in real-world applications.

Probabilistic approaches should be taken into account in order to model uncertainty appropriately. For this reason, most robotic applications used for real-world problems such as localization and mapping contain probabilistic calculations. So the

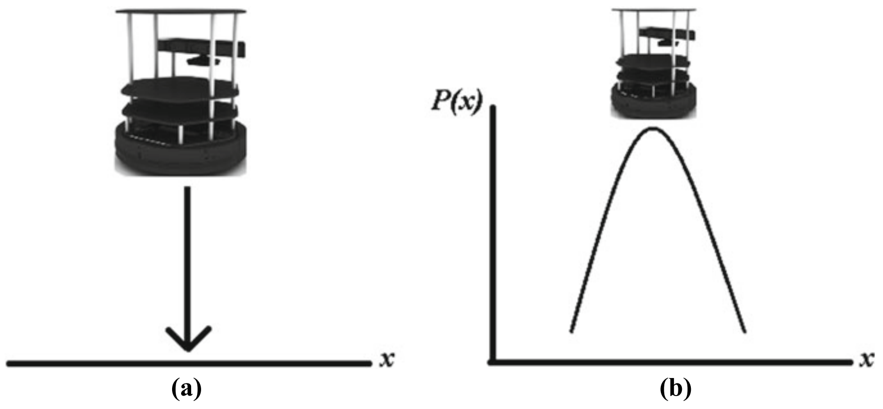
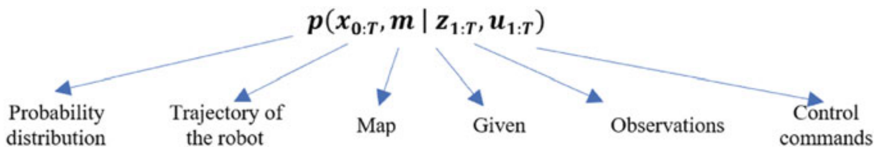


Fig. 5 Probabilistic and deterministic approach for a robot localization

problem should be expressed probabilistic. The problem for this study is the SLAM topic, and according to Table 1, the SLAM problem can be expressed as follows:

$$p(x_{0:T}, m \mid z_{1:T}, u_{1:T}) \tag{1}$$



Equation (1) is a probabilistic representation of the SLAM problem. In fact, it is the formulated version of Table 1. This Eq. (1) asks SLAM researchers a question: "What is the probabilistic distribution of the robot's location (x) and its environmental map (m), given the observation (z) and control commands (u)?" The purpose of all probabilistic SLAM methods is to solve this Eq. (1) efficiently.

In SLAM applications, the robot starts in an unknown environment and in an unknown location. Camera, laser, infrared, GPS, etc. sensors are used for the robot to detect its environment. The robot uses its sensors to observe the landmarks and, based on this information, estimates the location of the landmarks and calculates its position at the same time. For this, the robot must navigate the environment with control inputs. As the robot moves into the environment, the robot's changing observational perspective enables an incremental creation of a full landmarks map, which is used continuously to track the robot's current position relative to its initial position [43]. Simultaneous determination of both the map and the location is a state estimation problem. Therefore, the most commonly preferred methods for estimations made in SLAM applications are statistical based estimation methods. For this reason, Bayes-based methods are often used to successfully represent the probabilistic distribution in Eq. (1).

SLAM basically includes four modules: front-end, back-end optimization, loop closure detection and mapping. (see Fig. 6). In SLAM, firstly, measurement information is taken through a camera or sensors. The front-end includes feature extraction and position estimation, while the back-end has an optimization module. In the front-end step, the position of the robot is estimated using the robot's sensor data. However, the observed data contains different rates of noise in images, laser, etc. These noises cause cumulative errors in position estimation and these errors will increase over time. Thanks to filtering or optimization algorithms, the back-end can eliminate cumulative errors and improve localization and map accuracy. Loop closure detection is the ability of a robot to recognize a previously visited place. Loop closure solves the problem of predicted positions shifting over time [44–46].

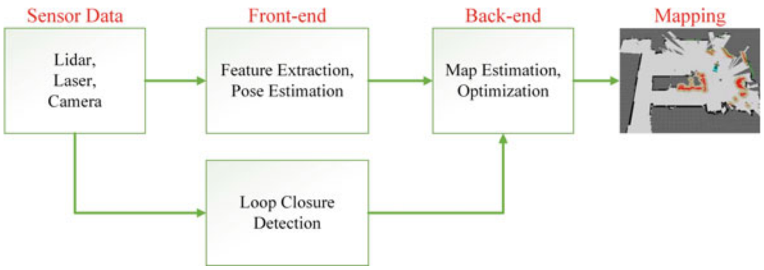


Fig. 6 Structure of SLAM

2.4 *Graphical Model of SLAM*

Generally, the relationships between states are shown by graphical modeling besides mathematical representation. The graphical modeling better explains the overall structure of the system when there is a time relationship between measured or predicted states.

Figure 7 shows the relationship between observed (measured) and unknown situations in the SLAM problem. Each node shows states and arrows also show interaction between states. For example, the robot's previous pose X_{t-1} and the current control input u_t or odometry information directly affect the robot's new pose X_t. Therefore, if the current position of the robot and the command given are known, the next position of the robot can be successfully estimated. This is called the motion model or state transition matrix. Also, when Fig. 7 is examined, the current observation Z_t depends on the current situation X_t and the map point m. That is, if the current status of the robot and the corresponding map point are known, the current measurement information can be successfully estimated. This is called the sensor model or measurement model.

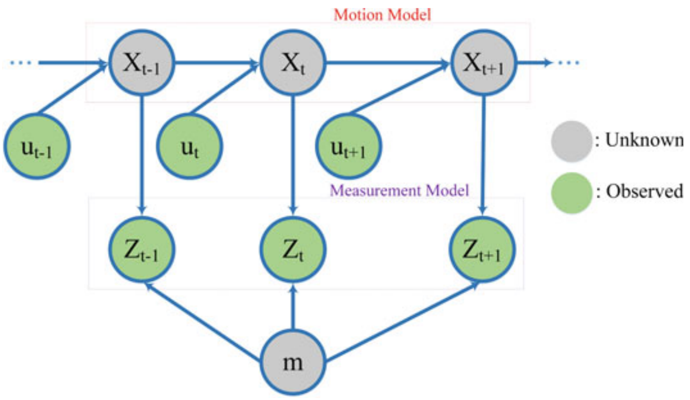


Fig. 7 Graphical model of SLAM problem

2.5 Full Versus Online SLAM

The purpose of SLAM is to estimate the robot's path and the map. However, this is possible in two ways: full SLAM and Online SLAM. If current and all past poses are taken into account, this is full SLAM. So the robot's entire path is predicted. This is shown in Eq. (2).

$$FullSLAM \rightarrow p(x_{0:T}, m|z_{1:T}, u_{1:T}) \tag{2}$$

If only the current pose is estimated and past poses are not taken into account, online SLAM is performed. As shown in Eq. (3), only the current position of the robot is estimated.

$$OnlineSLAM \rightarrow p(x_t, m|z_{1:T}, u_{1:T}) \tag{3}$$

Since robotic SLAM applications are often used in real-time application, Online SLAM is preferred more frequently. As the robot moves, the additional number of poses with respect to the past increases the calculation cost. Filter-based SLAM approaches marginalize previous situations. While this provides an advantage for processing time, it also brings inconsistency and linearization errors.

3 Bayesian Filter Based SLAM

Data recorded by cameras or sensors always have noise. For sensitive and efficient applications, it is necessary to make a state estimation by taking these noises into account. Not only the measuring devices cause noise, but objects can also be exposed to noise. External factors prevent the movement of the object and cause noise. Statistical methods estimate the state by taking these measurement and model uncertainties (noise) into account. For this, they take the current state and measurement values, then estimate the next state variable. In other words, the goal of a continuous state is to obtain the probability density function (pdf). Generally, the accepted method for such problems is to use the recursive Bayes filter method that solves the problem in two steps. These steps are the prediction and correction (update) step. Bayes-based state estimation can efficiently predict object movement by combining object state and observations.

3.1 Bayes Theorem

Bayes'theorem has an important place in mathematical statistics. As a formula, it is based on the relationship between conditional probabilities and marginal probabilities. This theorem aims to produce results using universal truths and observations in modeling any state. In other words, it produces a solution for state estimation. Estimating uncertain information using observations and prior information is the most important feature that distinguishes the Bayesian approach from classical methods. This method is also known as filtering method since the best estimation value is obtained from the noisy data with the Bayes method. Bayesian filter techniques are a powerful statistical method that is often preferred to overcome measurement uncertainty and to solve multiple sensor fusion problems.

Equation (4) shows the components of the Bayes formula, and Eq. (5) shows the Bayes formula. Bayes theorem is a rule that emerges as a result of conditional probability.

$$P(X|Z) = \frac{P(X, Z)}{P(Z)} \quad \text{or} \quad P(Z|X) = \frac{P(X, Z)}{P(X)} \tag{4}$$

$$P(X|Z) = \frac{P(Z|X)P(X)}{P(Z)}, \, P(Z) \neq 0 \tag{5}$$

In the Bayes theorem (Eq. 5), the first term in the numerator is called likelihood or sensor measurement information. The second term is called prior information, and the denominator is called evidence. The left side of Eq. (5) ($P(X|Z)$) is called belief (bel). In the Bayesian formula, if the denominator is expressed according to the total probability rule over the variable X, Eq. (6) is obtained. Equation (6) is valid for discrete cases. If belief covers a certain range rather than certain values, that is, if there is continuity, then Eq. (7) should be used. If there are more variables affecting the belief (e.g. control command (u)), then Eq. (8) is obtained.

$$P(X|Z) = \frac{P(Z|X)P(X)}{\sum_{i=0}^{n} P(Z|X)P(X)} \tag{6}$$

$$P(X|Z) = \frac{P(Z|X)P(X)}{\int P(Z|X)P(X)dX} \tag{7}$$

$$P(X|Z, U) = \frac{P(X, Z, U)}{P(Z, U)} = \frac{P(X)P(Z|X)P(U|X, Z)}{P(Z)P(U|Z)} \tag{8}$$

$P(Z)$ in the Bayesian equation in Eq. (5) does not affect the posterior distribution, it only ensures that the total value of the posterior distribution is equal to one. Since the $P(X|Z)$ (bel) distribution is the main target for Bayesian problems, its amplitude is not much concerned. Therefore, Eq. (5) can be expressed as follows.

$$P(X|Z) \quad \propto \quad P(Z|X)P(X) \tag{9}$$

As can be seen from Eq. (9), the posterior distribution is directly proportional ($\propto$) to the product of the likelihood and a priori distribution. That is, the posterior distribution is formed by using the available data and measuring the states with sensors. Obtaining the posterior distribution is mostly used in sequential computational applications. There is a continuous calculation. The prior information is first obtained or randomly assigned, then each posterior distribution is used as prior information for the next calculation. This situation can be thought of as an interconnected chain.

3.2 Recursive Calculation of State Distribution Using the Bayes Rule

The main purpose of the Bayesian approach is to estimate the current state of the system using observations up to the present state. Usually, observations obtain sequentially (recursive) in time. The posterior probability distribution ($Bel(x_t)$) measured in each time interval also includes uncertainty. Bayesian filters determine these belief values sequentially, using state-space equations. This situation can be shown as in Eq. (10). In Eq. (10), x represents the state variables, z represents the sensor data or observation variables [47].

$$Bel(x_t) = p(x_t|z_1, z_2, ..., z_t) \tag{10}$$

Equation (10) can be interpreted as follows: "If the history of sensor measurement data is $z_1, z_2, ..., z_t$, what is the probability that the object is at x position at time t?". As can be seen, the dependence on measurement data increases over time. Because in each time interval, a new measurement data is added to the process and complexity increases. Therefore, belief becomes increasingly incalculable. This will be seen in more detail when the recursive Bayes equation is obtained. To reduce this process complexity, Markov assumption is used in Bayesian filter applications. According to the Markov assumption, the current state x_t, contains all relevant information. Sensor measurements depend only on the current position of an object, and the x_t state of an object depends only on its x_{t--1} state and u_t control command. States before x_{t--1} do not provide additional information. The graphical model of Markov assumptions is shown in Fig. 8. In addition, the mathematical expression of the Markov assumption is given in Eqs. (11) and (12).

$$P(z_t|x_{1:t}, z_{1:t}, u_{1:t}) = P(z_t|x_t) \tag{11}$$

$$P(x_t|x_{1:t-1}, z_{1:t}, u_{1:t}) = P(x_t|x_{t-1}, u_t) \tag{12}$$

Recursive Bayesian filter is used to estimate sequential measurements and states. Markov assumptions and the total probability rule are used to recursively calculate the posterior probability distribution. Equations (13)–(19) shows the steps of creating a recursive Bayesian filter equation for sequential state prediction. The rules applied for the creation of Eqs. (14)– (18) are Bayes rule-Markov assumption-total probability rule-Markov assumption-Markov assumption [48, 49].

$$Bel(x_t) = P(x_t|u_{1:t}, z_{1:t}) \tag{13}$$

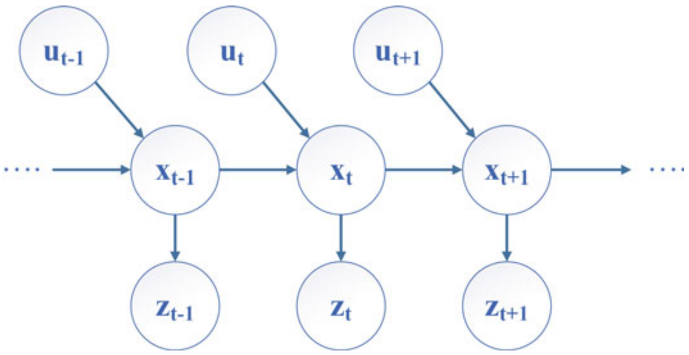


Fig. 8 Graphical model of Markov assumption

$$= \alpha P(z_t|x_t, z_{1:t}, u_{1:t}) P(x_t|u_{1:t}, z_{1:t-1}) \tag{14}$$

$$= \alpha P(z_t|x_t) P(x_t|u_{1:t}, z_{1:t-1}) \tag{15}$$

$$= \alpha P(z_t|x_t) \int P(x_t|u_{1:t}, x_{t-1}, z_{1:t-1}) P(x_{t-1}|u_{1:t}, z_{1:t-1}) dx_{t-1} \tag{16}$$

$$= \alpha P(z_t|x_t) \int P(x_t|u_t, x_{t-1}) P(x_{t-1}|u_{1:t}, z_{1:t-1}) dx_{t-1} \tag{17}$$

$$= \alpha P(z_t|x_t) \int P(x_t|u_t, x_{t-1}) P(x_{t-1}|u_{1:t-1}, z_{1:t-1}) dx_{t-1} \tag{18}$$

$$= \alpha P(z_t|x_t) \int P(x_t|u_t, x_{t-1}) Bel(x_{t-1}) dx_{t-1} \tag{19}$$

In Eq. (19):

- α: Normalization coefficient
- $P(z_t|x_t)$: Sensor Model
- $P(x_t|u_t, x_{t-1})$: Motion Model
- $Bel(x_{t-1})$: Previous State Distribution

Equation (19) is the equation of the recursive Bayesian filter. The α in the equations represents the normalization factor. $Bel(x_{t-1})$ represents the belief value for the previous case, i.e. the recursive term. $Bel(x_{t-1})$ is also called prior information. In short, it shows that the current state depends on the previous state. Other terms in the equation; $P(z_t|x_t)$ is called measurement model update or sensor model update, and $P(x_t|u_t, x_{t-1})$ is motion model update. As a result, Eq. (19) shows that if the previous state distribution is known, and also a control command has been applied and sensor observation has been obtained, the new state can be statistically estimated. Multiplying the previous belief with motion model in Eq. (19) forms the *prediction* step. Multiplying the value obtained in the prediction step with the sensor model and normalization factor is the *correction (update)* step. Recursive Bayesian prediction occurs as a result of the *prediction* and *correction* steps performed iteratively. The *prediction* and *correction* steps are shown in Eqs. (20) and (21), respectively.

$$\overline{Bel}(x_t) = \int P(x_t|u_t, x_{t-1}) Bel(x_{t-1}) dx_{t-1} \tag{20}$$

$$Bel(x_t) = \alpha P(z_t|x_t)\overline{Bel}(x_t) \tag{21}$$

The prediction step makes an estimate of the current state using the previous state and the motion model, and because it is an estimate, it includes an overline. Since there is an estimate based on motion information, noise increases at this step. In the correction step, since the sensor data is added to the estimate, the noise is reduced and a lower variance distribution is obtained than the previous state distribution. The concept is the same in Bayes-based state estimation methods, that is, the prediction-correction steps are repeated iteratively.

As mentioned in Chap. 2, the solution of the SLAM problem requires a recursive estimate of the robot's pose and the map of the environment. Therefore, SLAM is a state estimation problem. The predicted state x_t in the recursive Bayes formula (see Eq. 19) represents only one state. Since there are two different state estimations in SLAM, as seen in Eq. (3), the m is also used. In SLAM applications, x_t represents the position of the robot and m represents the location of the landmarks. The Bayes filter is a framework for recursive state estimation. It does not specify which technique should be used to solve the integral in the equations. There are different methods for solving this Eq. (19). For the solution, Kalman Filter (KF) and PF based methods are mostly used. To determine the method to be used, it should be checked whether the distribution of x_t is Gaussian, whether the motion and sensor model is linear. These factors indicate which type of Bayes filter will be more effective. For example, if models are linear and state distribution is Gaussian, the use of KF-based Bayes methods provides a more optimal estimation than the methods used for the general distribution and model. However, in reality nothing is perfectly Gaussian and nothing is perfectly linear. So, in this case, the KF may not the optimal solution to address the SLAM problem. For such cases, KF-based (e.g. EKF based-SLAM) and PF-based (e.g. FastSLAM) methods are available in the literature that can also produce solutions for nonlinear situations.

4 Keyframe BA Based SLAM

Apart from deep learning-based studies proposed recently, monocular SLAM solutions are either filter-based or keyframe BA-based. As explained in Chap. 3, probability distributions on features and camera pose parameters are updated in filter-based methods, then measurements taken from all images are combined sequentially

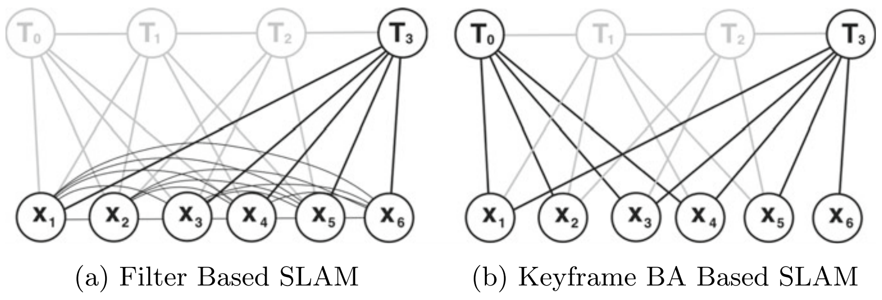


Fig. 9 Filter based versus Keyframe BA based SLAM [23]

(marginalization) (see Fig. 9a). All landmarks on the map are combined with the camera pose, so localization and mapping are intertwined. On the other hand, in keyframe BA systems, not all data is marginalized as in filters, localization and mapping are done in two different steps. Localization is performed using normal frames without keyframes (gray nodes in Fig. 9b), and mapping is performed using keyframes (black nodes in Fig. 9b) [32]. Figure 9 compares the two methods clearly. T_0, T_1, T_2, T_3 represent the time sequential poses of the camera, X_0, X_1, X_2, ... represent the position of the feature points on the map. As can be seen from the figure, filter-based SLAM combines previous poses on the last pose, so it is uncomplicated. However, as a result of marginalizing all the information on a single value, the graph quickly becomes interconnected, so inconsistency and linearization errors occur. In the BA based SLAM, an optimization approach is adopted. The camera pose is optimized according to the measured values. Unlike the filter, it stores a small subset (keyframes) of past poses. When the change in map points in consecutive frames reaches a certain threshold value, the relevant frame is assigned as a keyframe. Compared to filtering, keyframe BA based SLAM contains more elements as some of the past poses are retained. However, the absence of marginalization, preserving some of the past poses and optimizing the pose values make keyframe BA very efficient. If there is no keyframe selection, BA will perform optimization for each frame. Because with BA, the visual reconstruction of an environment based on a series of 3D points obtained from different camera angles and parameters is improved. However, with keyframe selection, this optimization process is applied to a limited number of frames, which is important for real-time SLAM implementation [23]. Due to this advantage, today's modern SLAM approaches, ORB-SLAM [29]and LSD-SLAM [30], are based on keyframe BA. The application of ORB-SLAM is presented in Chap. 5.

5 ROS Based SLAM Applications

Real-time robotic applications involving various sensors such as camera, odometry, ultrasonic sensor, etc. are difficult to develop. Sensors, actuators used on robots are not ideal and as a result, the system contains a large amount of uncertainty. The state of the robot and the environment are unpredictably variable. As a result, real information about the state of the robot or the environment can never be reached. This random situation makes it difficult to work with real robots. Each of the sensors used serves a different purpose, and the uncertainties in the behavior of these sensors differ from each other. There are also many algorithms for evaluating data on hardware elements and eliminating noise. As a result of all this complexity, a problem of uniformity arises. Moreover, writing this software repeatedly in different languages is also a waste of time. A platform called ROS has been thought to alleviate the burden of all this and lead to rapid developments in the field of robotics. This section includes a brief introduction to ROS and sample SLAM applications using ROS.

5.1 What Is ROS?

ROS is a free operating system for the robots that works on other operating systems and provides the services you would expect from an operating system. It can be configured on Ubuntu, Debian, and ROS 2 on Windows. ROS includes hardware simulators, low-level device control, message transmission between processes and functions, and package management. In addition, ROS includes tools and libraries for compiling, writing, and executing code on multiple computers. This description is accurate and clearly states that ROS is not a substitute for other operating systems but works alongside them. Below are the benefits of developing robotic software in ROS:

- Distributed computation: Many robots today are based on software that runs on multiple computers. For example, each sensor is controlled by a different machine and ROS provides simple and powerful mechanisms to handle it.
- Reusability of codes: The rapid development of robotics is due to the collection of good algorithms for the usual functions of robots such as navigation, routing and mapping, and so on. In fact, the existence of such algorithms will be useful when there is a way to implement them with the new concept, without the need to implement any algorithm for each new system. To prevent this problem, ROS provides different concepts such as standard packages and messages.
- Quick testing: One of the reasons why the development of robotic software is more challenging than the expansion of other parts is that it is time-consuming to test and find errors. A real robot may not always be available, and when it is available, the processing is very slow and tedious. ROS provides two effective tools for this problem; simulation environments and the ability to store and retrieve sensor data and messages in ROS bags.
- Stored and real data: The important thing is that the difference between the stored information and the actual sensor is negligible. Because the real robot, the simulator, and the re-run of the bag files, all three create the same kind of communication data. Therefore, codes don't have to work in different modes and we don't even have to specify whether it's connected to a real robot or stored data.

Although ROS is not the only platform that provides these features but what sets ROS apart is the vast robotic community that uses and supports it. This broad support makes ROS logically robust, expandable and improving in the future.

5.2 Sample SLAM Applications

SLAM of the mobile robot is a very important and necessary part of the overall problem of fully automating these robots. Common methods of Bayes-based SLAM are:

- Kalman Filter: It is an estimator that uses the previous state estimation and the current observation to calculate the current state estimation and is a very powerful tool for combining information in the presence of uncertainties.
- Particle Filter: This method is based on the Monte Carlo algorithm and uses particles to show the state distributions. It was developed to solve nonlinear problems that Kalman filters struggled to solve.

Different kinds of sensors can be used to implement a SLAM on ROS such as LIDAR, Ultrasonic sensor and cameras. This section presents an EKF-RGBD-SLAM package for 2D mapping along with a tutorial on using two common and open source SLAM packages available in ROS; GMapping[1] and Hector Slam.[2] Moreover, a step by step tutorial on using famous ORBSLAM2 is presented.

For the rest of this chapter, a working standard installation[3] of ROS Kinetic[4] and Gazebo simulation package[5] on Ubuntu Xenial Xerus (16.04)[6] is assumed. Moreover, successful running of the packages requires a working catkin build system. That is, necessary packages are required to be cloned to (or copied to) the *catkin/src* directory following by successful execution of below commands:

```
$ cp -r PACKAGE_PATH/PACKAGE_NAME CATKIN_PATH/catkin_ws/src
$ cd CATKIN_PATH/catkin_ws
$ catkin_make
$ source devel/setup.bash
```

[1] http://wiki.ros.org/gmapping.

[2] http://wiki.ros.org/hector_slam.

[3] http://wiki.ros.org/kinetic/Installation/Ubuntu.

[4] http://wiki.ros.org/kinetic.

[5] http://gazebosim.org/tutorials?tut=ros_overview.

[6] https://releases.ubuntu.com/16.04/.

5.2.1 EKF-RGBD-SLAM

A simple EKF-only based SLAM is implemented in this section which uses the RGBD sensor to obtain a 2D environment map. Unlike most EKF-only SLAMs implemented in ROS and simulation based on landmarks that locate the location of landmarks [50], EKF-RGBD-SLAM attempts to gain a 2D map of the environment. The package has been implemented in python and tested on Turtlebot in ROS and Gazebo simulation environment. This EKF-RGBD-SLAM is based on two separate sensors. The first is the encoders of the robot's wheels, which will give us details on the robot's motion. The second is the RGBD sensor, which helps the robot to detect the environment (walls and objects) and map the environment. It performs a gradual, discrete-time state estimation using EKF on the wheel odometry and RGBD measurements.
Subscribed Topics:

– odom: used for motion model and motion estimation
– scan: used for feature/obstacle detection and mapping

Published Topics:

– map: The map data to create 2D map
– entropy: Distribution probability of map related to robot pose

In this section, a step by step tutorial on using the EKF-RGBD-SLAM package is provided. The Gazebo simulation environment is used to gather robot and laser information. In order to start the EKF-RGBD-SLAM package, a single *ekf_rgbd_slam.launch* file needs to be executed in the command line. However, before executing the command let's take a look at the content of the main launch and python files. The content of the *ekf_rgbd_slam.launch* file is simple:

```
1  <launch>
2    <include file="$(find ekf_rgbd_slam)/launch/simulation.launch"/>
3    <node pkg="rviz" type="rviz" name="rviz" args="-d $(find ekf_rgbd_slam)/rviz/custom.rviz" output="screen"/>
4    <node pkg="ekf_rgbd_slam" type="ekf-rgbd-slam.py" name="ekf_rgbd_slam_node" output="screen"/>
5  </launch>
```

This launch file consists of three main parts. First part executes another launch file to start the Gazebo simulation environment (2). Second part opens the custom Rviz configuration file (3) and the third part starts the ekf_rgbd_slam node which is responsible for the input sensor data topic subscription, EKF sensor fusion and map generation, and output topic publishes (4). This node will be discussed in the next subsection. First let’s look at the content of the simulation.launch file that opens the robot model and test environment in Gazebo simulation:

```
1   <launch>
2     <arg name=“stacks” value=“$(optenv TURTLEBOT_STACKS circles)”/>
3     <arg name=“base” value=“$(optenv TURTLEBOT_BASE kobuki)”/>
4     <arg name=“3d_sensor” value=“$(optenv TURTLEBOT_3D_SENSOR kinect)”/ >
5     <include file=v$(find gazebo_ros)/launch/empty_world.launch”>
6       <arg name=“debug” value=“false”/>
7       <arg name=“world_name” value=“$(find ekf_rgbd_slam)/worlds/test.world”/>
8       <arg name=“use_sim_time” value=“true”/>
9     </include>
10    <include file=“$(find turtlebot_gazebo)/launch/includes/$(arg base).launch.xml”>
11      <arg name=“base” value=“$(arg base)”/>
12      <arg name=“stacks” value=“$(arg stacks)”/>
13      <arg name=“3d_sensor” value=“$(arg 3d_sensor)”/><arg name=“3d_sensor”
    value=“$(arg 3d_sensor)”/>
14    </include>
15    <node pkg=“robot_state_publisher” type=“robot_state_publisher”
    name=“robot_state_publisher”>
16      <param name=“publish_frequency” type=“double” value=“30.0” />
17    </node>
18    <node pkg=“nodelet” type=“nodelet” name=“laserscan_nodelet_manager”
    args=“manager”/>
19    <node pkg=“nodelet” type=“nodelet” name=“depthimage_to_laserscan” args=“load
    depthimage_to_laserscan/DepthImageToLaserScanNodelet laserscan_nodelet_manager”>
20      <param name=“scan_height” value=“10”/>
21      <param name=“output_frame_id” value=“/camera_depth_frame”/>
22      <param name=“range_min” value=“0.45”/>
23      <remap from=“image” to=“/camera/depth/image_raw”/>
24      <remap from=“scan” to=“/scan”/>
25    </node>
26  </launch>
```

The goal of this launch file is to define and open the test world with the Turtlebot robot in the Gazebo simulation environment. It defines arguments (2–4), opens the *test.world* file (5–9) and adds a Turtlebot robot with 3D RGBD sensor on it (10–25). The simulated Turtlebot could be customized by modifiyng the defined argument such as: TURTLEBOT_STACKS= hexagons, TURTLEBOT_3D_SENSOR=asus_xtion_pro and etc. Through modifying the world name to your own custom world file in line 8. In order to change the test environment we need to modifiy the *test.world* name to our custom world in line (8).
After launching the Gazebo simulation and Rviz, the ekf_rgbd_slam.launch file executes the core node *ekf-rgbd-slam.py* file. The slightly shortened and modified version of the core functions of this file is shown below:

```
class rgbdSlamNode(object):
    def __init__(self, x0 = 0, y0 = 0, theta0 = 0, odom_linear_noise = 0.025, odom_angular_noise = np.deg2rad(2), measurement_linear_noise = 0.2, measurement_angular_noise = np.deg2rad(10)):
        self.robotToSensor = np.array([x, y, theta])
        self.publish_maps = rospy.Publisher(”lines”, Marker)
        self.publish_uncertainity = rospy.Publisher(”uncertainity”, Marker)
        self.subscribe_scan = rospy.Subscriber(”scan”, LaserScan, self.laser_callback)
        self.subscribe_odom = rospy.Subscriber(”odom”, Odometry, self.odom_callback)
        self.last_odom = None
        self.odom = None
        self.odom_new = False
        self.laser_new = False
        self.pub = False
        self.x0 = np.array([x0,y0,theta0])
        self.ekf = ekf_rgbd_slam(x0, y0, theta0, odom_linear_noise, odom_angular_noise, measurement_linear_noise, measurement_angular_noise)
    def odom_callback(self, msg):
        self.time = msg.header.stamp
        trans = (msg.pose.pose.position.x, msg.pose.pose.position.y, msg.pose.pose.position.z)
        rot = (msg.pose.pose.orientation.x, msg.pose.pose.orientation.y, msg.pose.pose.orientation.z, msg.pose.pose.orientation.w)
        if self.last_odom is not None and not self.odom_new:
            delta_x = msg.pose.pose.position.x - self.last_odom.pose.pose.position.x
            delta_y = msg.pose.pose.position.y - self.last_odom.pose.pose.position.y
            yaw = yaw_from_quaternion(msg.pose.pose.orientation)
            lyaw = yaw_from_quaternion(self.last_odom.pose.pose.orientation)
            self.uk = np.array([delta_x * np.cos(lyaw) + delta_y * np.sin(lyaw), - delta_x * np.sin(lyaw) + delta_y * np.cos(lyaw), angle_wrap(yaw - lyaw)])
            self.odom_new = True
            self.last_odom = msg
        if self.last_odom is None:
            self.last_odom = msg
    def laser_callback(self, msg):
        self.time = msg.header.stamp
        rng = np.array(msg.ranges)
        ang = np.linspace(msg.angle_min, msg.angle_max, len(msg.ranges))
        points = np.vstack((rng * np.cos(ang), rng * np.sin(ang)))
        points = points[:, rng < msg.range_max]
        self.lines = splitandmerge(points, split_thres=0.1, inter_thres=0.3, min_points=6, dist_thres=0.12, ang_thres=np.deg2rad(10))
        if self.lines is not None:
            for i in range(0, self.lines.shape[0]):
                point1S = np.append(self.lines[i,0:2], 0)
                point2S = np.append(self.lines[i,2:4], 0)
                point1R = comp(self.robotToSensor, point1S)
                point2R = comp(self.robotToSensor, point2S)
                self.lines[i,:] = np.append(point1R[0:2], point2R[0:2])
            publish_lines(self.lines, self.publish_maps, frame='/robot', time=msg.header.stamp, ns='scan_lines_robot', color=(0,0,1))
            self.laser_new = True
    def iterate(self):
        if self.odom_new:
            self.ekf.predict(self.uk.copy())
            self.odom_new = False
            self.pub = True
        if self.laser_new:
            Innovk_List, H_k_List, S_f_List, Rk_List,idx_not_associated = self.ekf.data_association(self.lines.copy())
            self.ekf.update_position(Innovk_List, H_k_List, S_f_List, Rk_List)
            self.laser_new = False
            self.pub = True
        if self.pub:
            self.publish_results()
            self.pub = False
    def publish_results(self):
        msg_odom, msg_ellipse, trans, rot, env_map = get_ekf_msgs(self.ekf)
        publish_lines(env_map, self.publish_maps, frame='/world', ns='map', color=(0,0,1))
if __name__ == '__main__':
    rospy.init_node('rgbdSlam')
    node = rgbdSlamNode(x0=0, y0=0, theta0=0, odom_linear_noise = 0.025, odom_angular_noise = np.deg2rad(2), measurement_linear_noise = 0.5,0.2, measurement_angular_noise = np.deg2rad(10))
    r = rospy.Rate(10)
    while not rospy.is_shutdown():
        node.iterate()
        r.sleep()
```

The main function of the node starts at line (60). After initializing the node with initial variable values (61–62), we define the ros sensor reading rate (63) and let the program start by executing node.iterate function until the user manually exits the ros program (64–65). The iterate function is the main loop of the filter which performs the prediction and update functionalities of the EKF filter defined in *ekf_rgbd_slam.py* file (44–56). This file is explained later in this section. The iterate function reads new odom and RGBD sensor data received from two corresponding functions, odom_callback and laser_callback, respectively (15–43). The odom data are extracted and updated with the odom_callback function (15–28) to be used later by the iterate function. The laser_callback function receives the point cloud data from the RGBD sensor and extracts the map as an array of lines (29–43). After all the required data are gathered the publish_results function publishes the results (57–59). The auxiliary functions used to process point clouds, extract lines from them and publish the resulting topics are in *functions.py* file.

In order to better understand the EKF filtering part of the package we can look at the content of *ekf_rgbd_slam.py* file below. Please note that due to the long length of content, some of the auxiliary functions available in the original file are omitted and only main functions of EKF are included here. More details could be find in full package on Github.

```
class ekf_rgbd_slam(object):
  def __init__(self, x0, y0, theta0, odom_linear_noise, odom_angular_noise, measurement_linear_noise, measurement_angular_noise):
    self.odom_linear_noise = odom_linear_noise
    self.odom_angular_noise = odom_angular_noise
    self.measurement_linear_noise = measurement_linear_noise
    self.measurement_angular_noise = measurement_angular_noise
    self.chi_thres = 0.1026
    self.Qk = np.array([[ self.odom_linear_noise**2, 0, 0], [0, self.odom_linear_noise**2, 0 ], [ 0, 0, self.odom_angular_noise**2]])
    self.Rk=np.eye(2)
    self.Rk[0,0] = self.measurement_linear_noise
    self.Rk[1,1] = self.measurement_angular_noise
    self.x_B_1 = np.array([x0,y0,theta0])
    self.P_B_1 = np.zeros((3,3))
    self.featureObservedN = np.array([])
    self.min_observations = 0
  def predict(self, uk):
    n = self.get_number_of_features_in_map()
```

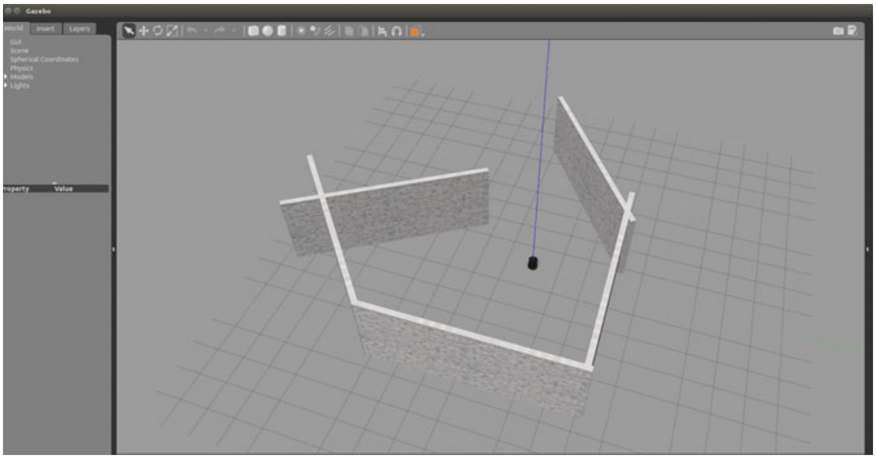


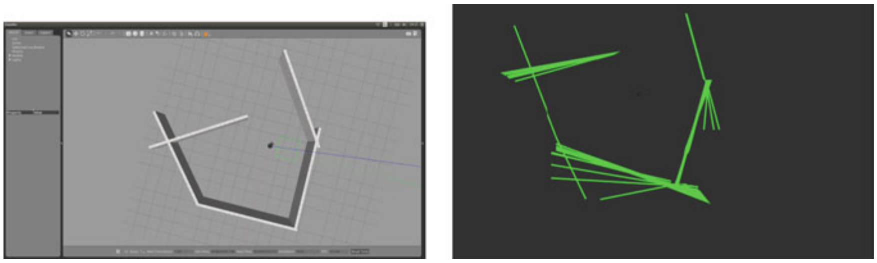
Fig. 10 Turtlebot 2 in Gazebo simulation environment

The class starts with initializing the state variables, linear and angular noise of odometry, and linear and angular noise of the RGBD measurement sensor and transforming them to matrices (2–15). The state of the robot according to previous state and odometry measurements along with its corresponding uncertainty is predicted in predict function (16–25). Inside this function, the number of observed features is calculated with get_number_of_features_in_map function (17). In order to obtain the uncertainty P_B_1, the jacobian matrices F_k and G_k of compositions are computed with respect to robot A_k and odometry W_k (18–21, 25). According to received odometry data u_k and its measurement matrix Q_k the odometry state is updated (22–24). After the prediction step, the update_position function updates the position of the robot according to the given position and the data association parameters computed with data_association function (26–35). The main part of this function is the computation of the kalman gain Kk (30–31) and updating the pose and uncertainty x_B_1 and P_B_1, respectively (32–35). To execute and test the package we can use a single command:

```
$ roslaunch ekf_rgb_slam ekf_rgbd_slam.launch
```

As mentioned earlier, this command launches the test environment and Turtlebot robot in a Gazebo simulation window as Fig. 10 and the mapping operation in Rviz window.

To view the published topics of the *ekf-rgbd-slam.py* node “rostopic list” command can be used. Here, entropy and map topics define the uncertainty and real-time map respectively. To view the contents of the topics “rostopic echo /TOPIC_NAME” command could be used. Since this is a very basic command to execute in terminal,



(a) Test environment in Gazebo (b) Resulting map in Rviz

Fig. 11 Test environment in Gazebo versus result in Rviz

the execution and viewing the output of this command is remained for the reader to do.

```
$ rostopic list
...
/entropy
...
/map
...
```

To navigate the Turtlebot in the Gazebo use the following command:

```
$ roslaunch turtlebot_teleop keyboard_teleop.launch
Control Your Turtlebot!
---------------------------
Moving around:
u i o
j k l
m ,

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%
space key, k : force stop
anything else : stop smoothly
```

Figure 11 shows the test environment in Gazebo and its resulting map in Rviz as markers after navigation of the robot for some time.

The test environment that we used the package to obtain its map is shown in Fig. 11. (a) and its corresopnding 2D map in (b). The green lines show the obstacles (walls) sensed be RGBD sensor. The amount of the lines and their length can be configures by the threshold variables defined by splitandmerge function in the *ekf-rgbd-slam.py* file. The argument of this function are split_thres, inter_thres, min_points, dist_thres and ang_thres.

– split_thres: Distance threshold to trigger a split

– inter_thres: Max distance between consecutive points in a line
– min_points: Min number of points in a line
– dist_thres: Max distance to merge lines
– ang_thres: Max angle to merge lines

5.2.2 GMapping

This package is used to obtain a 2D grid map of the environment based on the laser data and comes built-in if we install the ROS using the recommended desktop-full method. The subscribed and published topics of this package are as below: Subscribed Topics:

– tf: Used to properly connect robot base, encoder odometry and laser data to obtain the 2d grid-map.
– scan: Used to create the map.

Published topics:

– map_metadata: Information about the created map
– map: The map data to create 2d grid map
– ∼entropy: Distribution probability of map related to robot pose(The higher value is the higher uncertainty)

More information about the Gmapping package is available at the official webpage in ROS Wiki.[7]

In this section, a step by step tutorial on using the Gmapping package is provided. The Gazebo simulation environment is used to gather robot and laser information. The first step is to launch the robot and laser sensors. To do this in the Gazebo the following command is used in the command line.

```
$ roslaunch turtlebot_gazebo turtlebot_world.launch
```

Once again as in previous section, the custom modification could be applied in the environment such as:

```
1 $ export TURTLEBOT_STACKS=circles
2 $ export TURTLEBOT_BASE=create
3 $ export TURTLEBOT_3D_SENSOR=asus_xtion_pro
4 $ roslaunch turtlebot_gazebo turtlebot_world.launch world_file := worlds/willowgarage.world
```

Through lines (1–3) the custimizations to the Turtlebot can be applied and through the line (4) you can specify your custom world file. Fig. 12 shows the window opened after executing “roslaunch turtlebot_gazebo turtlebot_world.launch” command.

[7]ROS Wiki [Online]. Available: http://wiki.ros.org/..

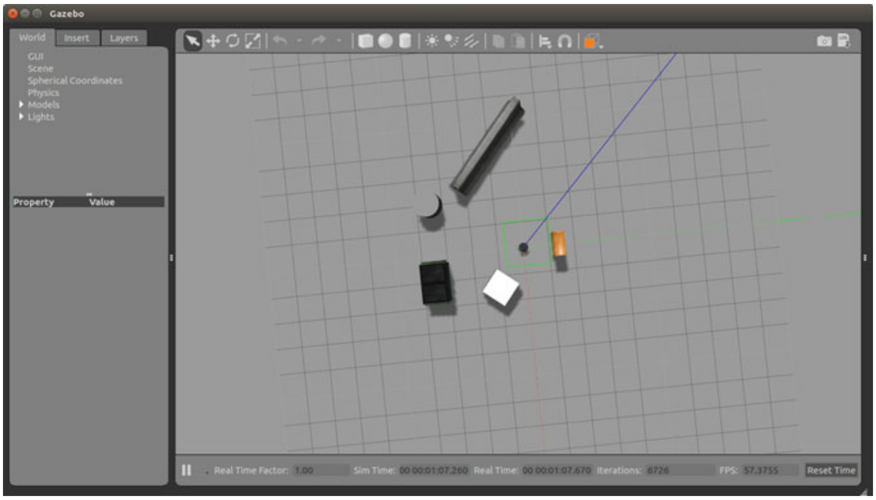


Fig. 12 Turtlebot 2 in Gazebe simulation environment

To view the published topics use the following command in a command line tab/window:

```
$ rostopic list
/camera/depth/camera_info
/camera/depth/image_raw
/odom
/rosout
/rosout_agg
/scan
/tf
/tf_static
```

Here the /camera/... topics define the information for the camera of RGBD sensor. the /odom topic defines the robot location in reference to the initial position and the /scan topic defines the depth sensor information of RGBD camera. The /tf/... topics define the transformation among the robot and different sensors installed on it. In order to view the content of the topics use the following commands:

```
$ rostopic echo /scan
header:
seq: 863
stamp:
secs: 498
nsecs: 380000000
frame_id: “/camera_depth_frame”
angle_min: −0.521567881107
angle_max: 0.524276316166
angle_increment: 0.00163668883033
time_increment: 0.0
scan_time: 0.0329999998212
range_min: 0.449999988079
range_max: 10.0
ranges: [1.325965166091919, 1.325973629951477, 1.3259814975874,
1.3259918689727783, 1.3260047435760498, 1.32603800296
1.32608151435
nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
nan, nan, nan, nan, nan, nan, nan]
intensities: []
—
```

```
$ rostopic echo /tf
transforms:
–
header:
seq: 0
stamp:
secs: 1782
nsecs: 930000000
frame_id:“odom”
child_frame_id: “base_footprint”
transform:
translation:
x: 2.53603142551e-06
y: -2.55864843431e-07
z: -0.322665376962
rotation:
x: 0.0
y: 0.0
z: -0.322665376962
w: 0.94651310319
—
```

Now the Gmapping could be started by following command in a new window/tab:

```
$ roslaunch turtlebot_navigation gmapping_demo.launch
... logging to /home/user/.ros/log/64848-60f26251/h-user-63.log
Checking log directory for disk usage. This may take a while
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB
...
```

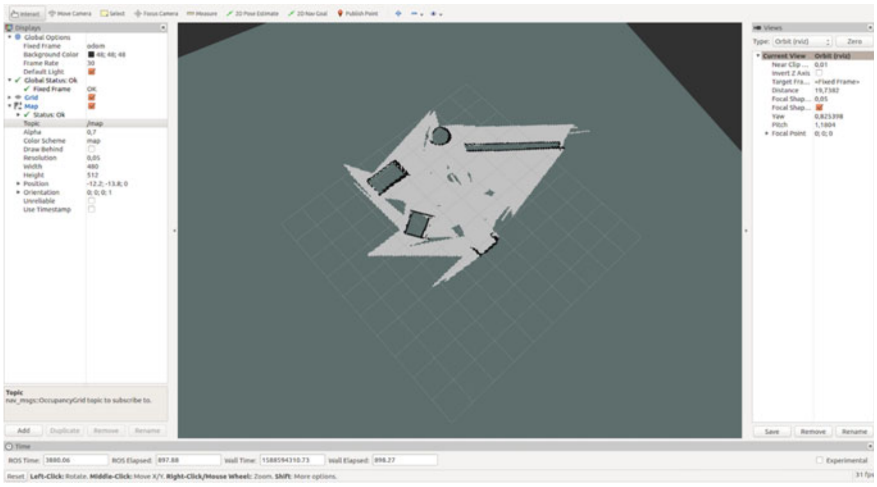
This command will publish three more topics as below. The content of them can be viewed using the echo command as above.

```
$ rostopic list
...
/map
/map_metadata
...
/slam_gmapping/entropy
...
```

The /map topic defines a 2D grid map, in which each cell represents the probability of occupancy in range [0–100] and defining the unknown as -1. The /map_metadata topic defines the basic information about the characterists of the map such as time, resolution, width, height and the origin of the map. Finally, the /slam_gmapping/entropy defines the uncertainty of the robot pose in a floating point number. To navigate the Turtlebot in the Gazebo you can use the teleoperation tool:

```
$ roslaunch turtlebot_teleop keyboard_teleop.launch
...
Control Your Turtlebot!
---------------------------
Moving around:
u i o
j k l
m ,

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%
space key, k : force stop
anything else : stop smoothly
```

As shown in the table above, the Turtlebot motion orientation and velocity could be controlled through the different keyboard characters. Moreover, you can use other command sources such as PS3 or XBOX360 joysticks by executing *ps3_teleop.launch* or *xbox360_teleop.launch*, respectively, instead of *key-*



```
Global Options
  Fixed Frame          odom
  Background Color     48; 48; 48
  Frame Rate           30
  Default Light        ✓
Global Status: Ok
  Fixed Frame          OK
Grid                   ✓
Map                    ✓
  Status: Ok
  Topic                /map
  Alpha                0,7
  Color Scheme         map
  Draw Behind          ☐
  Resolution           0,05
  Width                480
  Height               512
  Position             -12.2; -13.8; 0
  Orientation          0; 0; 0; 1
  Unreliable           ☐
  Use Timestamp        ☐

Topic
nav_msgs::OccupancyGrid topic to subscribe to.

Type: Orbit (rviz)
Current View     Orbit (rviz)
  Near Clip ...    0,01
  Invert Z Axis    ☐
  Target Fra...    <Fixed Frame>
  Distance         19,7382
  Focal Shap...    0,05
  Focal Shap...    ✓
  Yaw              0,825398
  Pitch            1,1804
  Focal Point      0; 0; 0

ROS Time: 3880.06   ROS Elapsed: 897.88   Wall Time: 1588594310.73   Wall Elapsed: 898.27   ☐ Experimental
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel: Zoom. Shift: More options.
```

Fig. 13 GMapping in Rviz

board_teleop.launch. In order to view the 2D grid map in the Rviz environment use be below command with the given configuration as presented in Fig. 13:

```
$ rosrun rviz rviz
[ INFO] [1588593411.985889716]: rviz version 1.12.17
[ INFO] [1588593411.985969124]: compiled against Qt version 5.5.1
[ INFO] [1588593411.985999037]: compiled against OGRE version 1.9.0
[ INFO] [1588593412.146953582]: Stereo is NOT SUPPORTED
[ INFO] [1588593412.147066937]: OpenGl version: 3 (GLSL 1.3)
[ INFO] [1588593430.357540892, 3000.130000000]: Creating 1 swatches
```

The custom topics and sensor information could be added and displayed using the Add button in the left buttom of the window. After navigating the robot using teleoperation tool the map is displayed in through the map topic in the Rviz. Above "rosrun rviz rviz" command will open a new window as shown in Fig. 13.

5.2.3 Hector Mapping

In this section the hector_mapping package is used to obtain a 2D grid map of the environment. This package also comes with the desktop-full installation of the ROS and uses LIDAR laser scan to create an accurate 2D map. The difference between this Gmapping and hector_mapping package is that the later one can perform SLAM operation without odometry information. Therefore, hector_mapping could be an alternative solution to gmapping in cases where odometry data is unavailable.

Subscribed Topics:

– scan: LIDAR scan data used to create the map.

– syscommand: String to control the mapping state such as resetting.

Published topics:

– map_metadata: Information about the created map
– map: The map data to create 2D grid map
– slam_out_pose: The estimated pose of LIDAR
– poseupdate: The estimated pose of LIDAR with distribution probability

More information about the hector_mapping package is available at the official web-page in ROS Wiki.[8]

In this section, the hector mapping algorithm is used to obtain the 2D grid map. Since hector mapping requires LIDAR and due to unavailability of LIDAR in Turtlebot version 2 we will use Turtlebot version 3 in Gazebo. In order to do that, start Gazebo Turtlebot version 3 world using below command:

```
$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

TurtleBot 3 world is a Gazebo simulation world consisting of basic items that form the shape of the TurtleBot 3 inside a test environment. TurtleBot 3 world is primarily used for testing purposes such as SLAM and Navigation. Once again it is possible to customize the robot we are using or the open a different world using command such as:

```
1 $ export TURTLEBOT3_MODEL=${TB3_MODEL}
2 $ roslaunch turtlebot3_gazebo turtlebot3_house.launch
```

The type of the Turtlebot 3 model such as burger, waffle, waffle pi could be defined by ${TB3_MODEL} where TB3_MODEL is the model of the turtlebot. In addition, the custom world file could be defined by command in line 2, where *turtlebot3_house.launch* is another predefined house environment for Turtlebot 3. You can define your own custom world instead of this file. The above "roslaunch turtlebot3_gazebo turtlebot3_world.launch" command will open a new Gazebo window as shown in Fig. 14.

[8]ROS Wiki [Online]. Available: http://wiki.ros.org/..

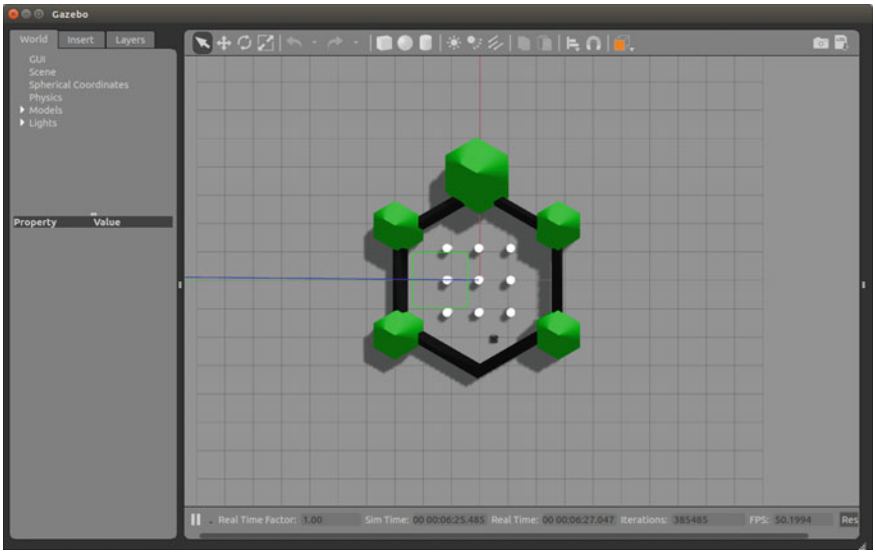


Fig. 14 Turtlebot 3 in Gazebo simulation environment

The navigation through the environment is possible by teleoperation tool for Turtlebot 3. This tool is same as the one described in previous section for Turtlebot 2 in Gmapping. Hence, not much details are given here.

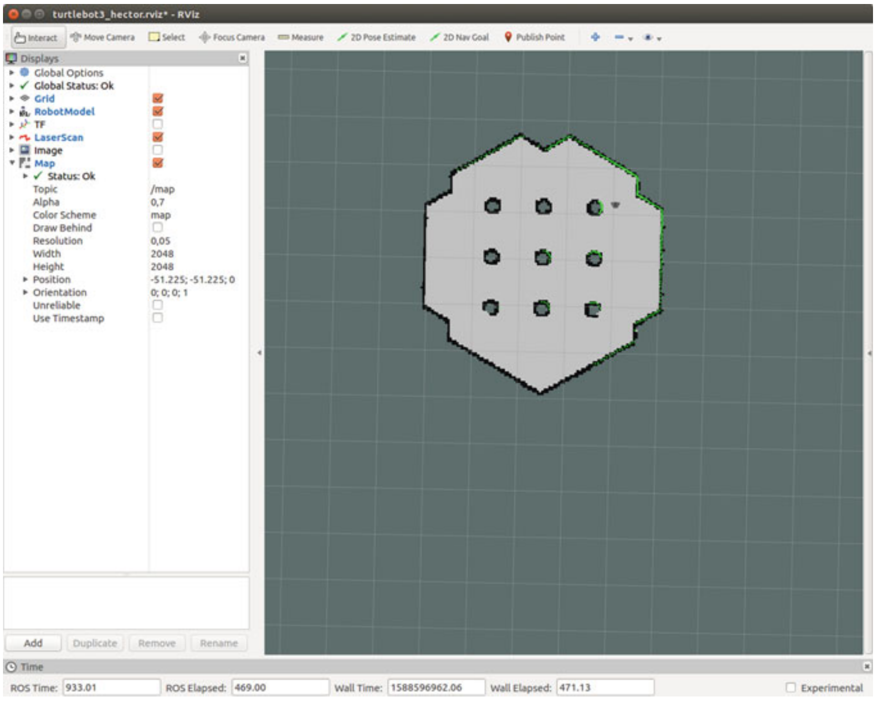
```
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

In order to obtain the 2D grid map using hector slam method below command can be used.

```
$ roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=hector
```

This command will start the hector mapping operation. There are other option such as Gmapping for Turtlebot 3 which you can define by specifying *slam_methods:= gmapping* instead of *slam_methods:=hector*. This command supports Gmapping, Cartographer, Hector, and Karto among various SLAM methods.

Figure 15 shows the result of the “roslaunch turtlebot3_slam turtlebot3_slam. launch slam_methods:=hector” command in Rviz tool after navigating the Turtlebot 3 using teleoperation tool. Like the Rviz tool mentioned in previous section, the displayable topics are totally customizable.



Global Options
Global Status: Ok
Grid ✓
RobotModel ✓
TF ☐
LaserScan ✓
Image ☐
Map ✓
Status: Ok
Topic /map
Alpha 0,7
Color Scheme map
Draw Behind ☐
Resolution 0,05
Width 2048
Height 2048
Position -51.225; -51.225; 0
Orientation 0; 0; 0; 1
Unreliable ☐
Use Timestamp ☐

ROS Time: 933.01 ROS Elapsed: 469.00 Wall Time: 1588596962.06 Wall Elapsed: 471.13 ☐ Experimental

Fig. 15 Hector mapping in Rviz

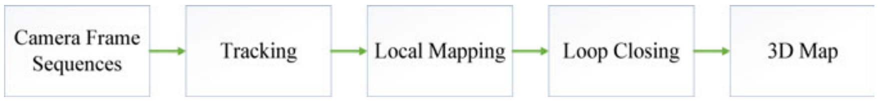


Fig. 16 ORB-SLAM2 data flow

5.2.4 ORB-SLAM2

ORB-SLAM2[51] is a 3D map construction VSLAM method that uses a camera as the data gathering sensor. This method was first proposed by Raul Mur-Artal and et al. for monocular cameras in 2015 and later improved and added the support for RGBD and stereo camera in 2017. ORBSLAM2 treats the SLAM problem as three different threads that work in parallel to produce a 3D map in real-time. The data flow of this method is shown in Fig. 16.

The tracking thread locates the camera and specifies when a new keyframe should be added. It matches current frame FAST features with the previous frame and optimizes the pose with motion bundle adjustment. It uses ORB as the feature descriptor. In case the tracking is lost, the place recognition module starts and attempts to relocalize the camera. The local mapping thread uses the concept of covisibility graph of keyframes to obtain a local visible map. The ORB features are triangulated and matched in connected keyframes in the covisibility graph. In case the matched map point is found in more than 25and is observed by at least three keyframes, it will be added to the local map. The loop closing thread uses a bag of words principle to identify possible loops within the system and to adapt the global optimization. It searches the bag of words [52] in the covisibitlity graph of the current keyframe and its vicinity. If three clear loop candidates are successively found, this loop is known to be a serious candidate. Afterwards, a series of optimizations and a RANSAC are applied to these loop candidates to remove the noise and accept one if necessary. In order to handle the scale and final optimization the pose and map points of current keyframe and its neighbours are corrected and fused respectively.

In this section, we present a step by step SLAM using ORS-SLAM2 on the KITTI [53] dataset. In order to do that, the prerequisite libraries are:

- C++11/C++0x compiler
- OpenCV (for image manipulation and feature extraction)
- Eigen3 (for performing mathematical operations on the Matrices)
- Pangolin (for visualization and user interface)
- DBoW2 (for indexing and converting images into a bag-of-word representation)

First, we need to clone the ORB-SLAM repository from Github into the working environment:

```
$ git clone https://github.com/raulmur/ORB_SLAM2.git
```

Second, build the library using following commands:

```
$ cd ORB_SLAM2
$ chmod +x build.sh
$ ./build.sh
```

Third, download the KITTI or other dataset into a folder and execute the following command. Edit the *KITTIX.yaml* and DATASET_PATH as per your folder structure.

```
$ ./Examples/Monocular/mono_kitti Vocabulary/ORBvoc.txt
      Examples/Monocular/KITTIX.yaml
      DATASET_PATH/dataset/sequences/SEQUENCE_NUMBER
```

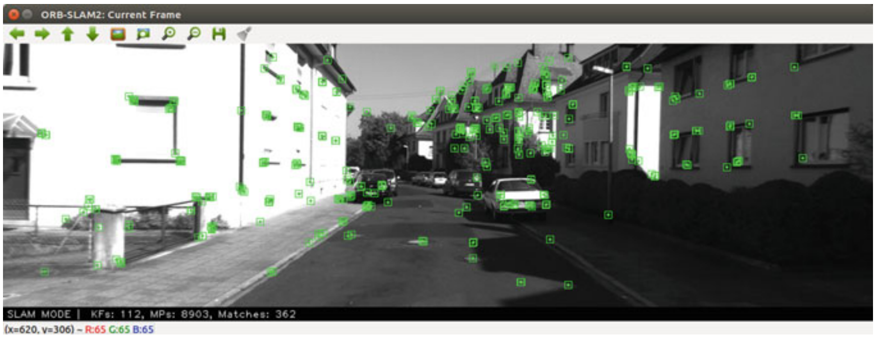
In this command *./Examples/Monocular/mono_kitti* points to the path of executable created after building the library using *./build.sh* command. It should be noted that the command should be run from inside the ORB_SLAM2 main folder or otherwise the file paths need to be configured accordingly.

The *Vocabulary/ORBvoc.txt* file represents the vocabulary file. This file is used by DBoW2 to fast recognition and loop closure in case of losing tracking features. The same vocabulary file can be used every time since it has been collected from a huge set of data and works well.

The *Examples/Monocular/KITTIX.yaml* file points to a yaml setting file. This file contains the ORB and camera calibration parameters and rectification information in case the frames are not pre-rectified. Below is a sample yaml file settings for camera used by KITTI dataset. The intrinsic calibration matrix could contain the calibration and distortion parameters of the camera. Other parameters such as width, height, fps depend on the resolution of camera.

```
# Camera/distortion parameters
Camera.k1: 0.0
Camera.k2: 0.0
Camera.p1: 0.0
Camera.p2: 0.0

Camera.fx: 718.856
Camera.fy: 718.856
Camera.cx: 607.1928
Camera.cy: 185.2157

# Camera frames per second
Camera.fps: 10.0

# Color order of the images (0: BGR, 1: RGB. It is ignored if images are grayscale)
Camera.RGB: 1

# ORB Extractor: Number of features per image
ORBextractor.nFeatures: 2000

# ORB Extractor: Scale factor between levels in the scale pyramid
ORBextractor.scaleFactor: 1.2

# ORB Extractor: Number of levels in the scale pyramid
ORBextractor.nLevels: 8

# ORB Extractor: Fast threshold
ORBextractor.iniThFAST: 20
ORBextractor.minThFAST: 7

# Viewer Parameters
Viewer.KeyFrameSize: 0.1
Viewer.KeyFrameLineWidth: 1
Viewer.GraphLineWidth: 1
Viewer.PointSize:2
Viewer.CameraSize: 0.15
Viewer.CameraLineWidth: 2
Viewer.ViewpointX: 0
Viewer.ViewpointY: -10
Viewer.ViewpointZ: -0.1
Viewer.ViewpointF: 2000
```

Finally, the *PATH_TO_DATASET_FOLDER/dataset/sequences/SEQUENCE_NUMBER* argument denotes the folder path containing the dataset sequences. The folder should contain all the images that needs to be used in the ORB_SLAM. Figures 17 and 18 illustrate the results after executing above command that demonstrates a real-time SLAM process.



SLAM MODE | KFs: 112, MPs: 8903, Matches: 362

(x=620, y=306) ~ R:65 G:65 B:65

Fig. 17 KITTI dataset seq-00 frame

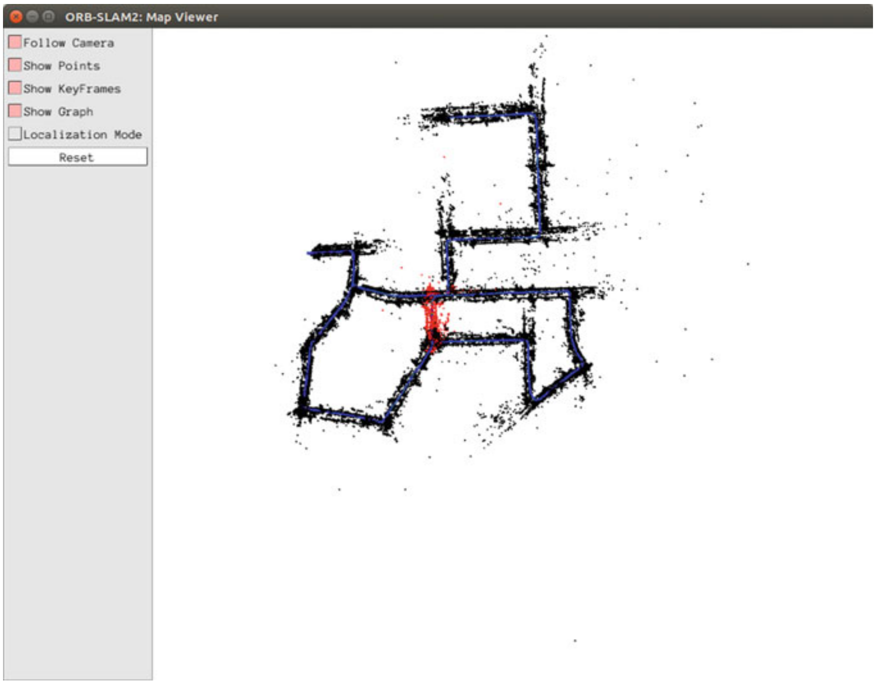


Fig. 18 ORB-SLAM2 on KITTI dataset seq-00

6 Conclusion

In this study, the SLAM tutorial is presented for mobile robotic. For this, general-to-specific approach has been adopted. First, a general information about mobile robotics is given and the importance of SLAM for an autonomous mobile robot is emphasized. Later, the SLAM problem is introduced and information about the solution of the SLAM problem is given. After providing the reader the theoretical basis of SLAM, the Bayes filter used for the SLAM solution is introduced and the relationship between recursive Bayesian filter and SLAM has been shown both theoretically and mathematically. Afterward, keyframe BA based SLAM, which includes state-of-the-art methods, is mentioned. Finally, based on this theoretical information, the ROS environment is introduced for the reader to practice and, four examples of SLAM studies are explained step by step. With this study, SLAM problem and Bayes-based SLAM and keyframe BA-based SLAM solution techniques are presented to the reader in a detailed and easy-to-understand manner. Also, thanks to Chap. 5, the reader will have a basic level of knowledge to perform SLAM applications in the ROS environment.

Acknowledgements Authors are thankful to RAC-LAB (www.rac-lab.com) for providing the data.Conflict of Interest: The authors declare that they have no conflict of interest.

References

1. J.J. Craig, Introduction to robotics: mechanics and control, 3/E (Pearson Education India, Chennai, 2009)
2. R. Siegwart, I.R. Nourbakhsh, D. Scaramuzza, Introduction to autonomous mobile robots (MIT press, Cambridge, 2011)
3. B. Mu, J. Chen, Y. Shi, Y. Chang, Design and implementation of nonuniform sampling cooperative control on a group of two-wheeled mobile robots. IEEE Trans. Ind. Electron. **64**(6), 5035–5044 (2016)
4. F. Ruggiero, V. Lippiello, A. Ollero, Aerial manipulation: a literature review. IEEE Robot. Autom. Lett. **3**(3), 1957–1964 (2018)
5. M. Hutter et al., ANYmal-toward legged robots for harsh environments. Adv. Robot. **31**(17), 918–931 (2017)
6. Y. Tang, L. Qin, X. Li, C. Chew, J. Zhu, A frog-inspired swimming robot based on dielectric elastomer actuators, in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2017), pp. 2403–2408
7. D. Di Paola, A. Milella, G. Cicirelli, A. Distante, An autonomous mobile robotic system for surveillance of indoor environments. Int. J. Adv. Robot. Syst. **7**(1), 8 (2010)
8. Y. Sasaki, J. Nitta, Long-term demonstration experiment of autonomous mobile robot in a science museum, in *2017 IEEE International Symposium on Robotics and Intelligent Sensors (IRIS)* (IEEE, Ottawa, 2017), pp. 304–310
9. M.F. Aslan, A. Durdu, K. Sabanci, Shopping robot that make real time color tracking using image processing techniques. Int. J. Appl. Math. Electron. Comput. **5**(3), 62–66 (2017)
10. Y.-G. Kim, H.-K. Kim, S.-G. Lee, K.-D. Lee, Ubiquitous home security robot based on sensor network (2006)

11. I. Palunko, P. Cruz, R. Fierro, Agile load transportation: safe and efficient load manipulation with aerial robots. IEEE Robot. & Autom. Mag. **19**(3), 69–79 (2012)
12. F. Rubio, F. Valero, C. Llopis-Albert, A review of mobile robots: concepts, methods, theoretical framework, and applications. Int. J. Adv. Robot. Syst. **16**(2), 1729881419839596 (2019)
13. C. Stachniss, J.J. Leonard, S. Thrun, Simultaneous localization and mapping, in *Springer Handbook of Robotics* (Springer, Berlin, 2016), pp. 1153–1176
14. E. Trabes, M.A. Jordan, A node-based method for SLAM navigation in self-similar underwater environments: a case study. Robotics **6**(4), 29 (2017)
15. Y. Chen et al., *Possibility of Applying SLAM-Aided LiDAR in Deep Space Exploration* (Springer International Publishing, Cham, 2017), pp. 239–248
16. Z. Ren, L. Wang, L. Bi, Robust GICP-based 3D LiDAR SLAM for underground mining environment. Sensors **19**(13), 2915 (2019)
17. J.-C. Piao, S.-D. Kim, Adaptive monocular visual-inertial SLAM for real-time augmented reality applications in mobile devices. Sensors **17**(11), 2567 (2017)
18. J. Zhang, Y. Ou, G. Jiang, Y. Zhou, An approach to restaurant service robot SLAM, in *IEEE International Conference on Robotics and Biomimetics (ROBIO)* (2016), pp. 2122–2127
19. J.-C. Trujillo, R. Munguia, E. Guerra, A. Grau, Cooperative monocular-based SLAM for multi-UAV systems in GPS-denied environments. Sensors **18**(5), 1351 (2018)
20. A. Al-Kaff, D. Martin, F. Garcia, A. de la Escalera, J.M. Armingol, Survey of computer vision algorithms and applications for unmanned aerial vehicles. Expert. Syst. Appl. **92**, 447–463 (2018)
21. M. Bueno, H. González-Jorge, J. Martínez-Sánchez, L. Díaz-Vilariño, P. Arias, Evaluation of point cloud registration using Monte Carlo method. Measurement **92**, pp. 264–270 (2016)
22. R. Giubilato, S. Chiodini, M. Pertile, S. Debei, An evaluation of ROS-compatible stereo visual SLAM methods on a nVidia Jetson TX2. Measurement **140**, 161–170 (2019)
23. H. Strasdat, J.M. Montiel, A.J. Davison, Visual SLAM: why filter? Image Vis. Comput. **30**(2), 65–77 (2012)
24. A. Chatterjee, O. Ray, A. Chatterjee, A. Rakshit, Development of a real-life EKF based SLAM system for mobile robots employing vision sensing. Expert. Syst. Appl. **38**(7), 8266–8274 (2011)
25. A. Chatterjee, F. Matsuno, A Geese PSO tuned fuzzy supervisor for EKF based solutions of simultaneous localization and mapping (SLAM) problems in mobile robots. Expert. Syst. Appl. **37**(8), 5542–5548 (2010)
26. M. Quan, S. Piao, M. Tan, S.-S. Huang, Accurate monocular visual-inertial SLAM using a map-assisted EKF approach. IEEE Access **7**, 34289–34300 (2019)
27. G. Klein, D. Murray, Parallel tracking and mapping for small AR workspaces, in *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality* (IEEE, Japan, 2007), pp. 225–234
28. H. Strasdat, J. Montiel, A.J. Davison, Scale drift-aware large scale monocular SLAM. Robotics: Science and Systems VI, **2**(3), 7 (2010)
29. R. Mur-Artal, J.M.M. Montiel, J.D. Tardos, ORB-SLAM: a versatile and accurate monocular SLAM system. IEEE Trans. Robot. **31**(5), 1147–1163 (2015)
30. J. Engel, T. Schops, D. Cremers, LSD-SLAM: large-scale direct monocular SLAM (Springer International Publishing, Cham, 2014), pp. 834–849
31. C. Forster, M. Pizzoli, D. Scaramuzza, SVO: fast semi-direct monocular visual odometry, in *2014 IEEE international conference on robotics and automation (ICRA)* (IEEE, Hong Kong, 2014), pp. 15–22
32. G. Younes, D. Asmar, E. Shammas, J. Zelek, Keyframe-based monocular SLAM: design, survey, and future directions. Robot. Auton. Syst. **98**, 67–88 (2017)
33. B. Huang, J. Zhao, J. Liu, A survey of simultaneous localization and mapping (2019), arXiv:1909.05214
34. H. Durrant-Whyte, T. Bailey, Simultaneous localization and mapping: part I. IEEE Robot. & Autom. Mag. **13**(2), 99–110 (2006)

35. J. Sola, Simulataneous localization and mapping with the extended Kalman filter, Avery quick guide with MATLAB code (2013)
36. G. Grisetti, R. Kummerle, C. Stachniss, W. Burgard, A tutorial on graph-based SLAM. IEEE Intell. Transp. Syst. Mag. **2**(4), 31–43 (2010)
37. T. Taketomi, H. Uchiyama, S. Ikeda, Visual SLAM algorithms: a survey from 2010 to 2016. IPSJ Trans. Comput. Vis. Appl. **9**(1), 16 (2017)
38. J. Fuentes-Pacheco, J. Ruiz-Ascencio, J.M. Rendón-Mancha, Visual simultaneous localization and mapping: a survey. Artif. Intell. Rev. **43**(1), 55–81 (2015)
39. C. Cadena et al., Past, present, and future of simultaneous localization and mapping: toward the robust-perception age. IEEE Trans. Robot. **32**(6), 1309–1332 (2016)
40. A. Durdu, M. Korkmaz, A novel map merging technique for occupancy grid-based maps using multi-robot: a semantic approach. Turk. J. Electr. Eng. Comput. Sci. **27**(5), 3980–3993. https://doi.org/10.3906/elk-1807-335
41. D. Scaramuzza, F. Fraundorfer, Visual odometry [tutorial]. IEEE Robot. & Autom. Mag. **18**(4), 80–92 (2011)
42. C. Stachniss, Introduction to robot mapping, http://ais.informatik.uni-freiburg.de/teaching/ws12/mapping/pdf/slam03-ekf.pdf
43. D.J. Spero, R.A. Jarvis, A review of robotic SLAM (2007)
44. A. Li, X. Ruan, J. Huang, X. Zhu, F. Wang, Review of vision-based simultaneous localization and mapping, in *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)* (IEEE, China, 2019), pp. 117–123
45. G. Jiang, L. Yin, S. Jin, C. Tian, X. Ma, Y. Ou, A simultaneous localization and mapping (SLAM) framework for 2.5 D map building based on low-cost LiDAR and vision fusion. Appl. Sci. **9**(10), 2105 (2019)
46. Y. Xia, J. Li, L. Qi, H. Fan, Loop closure detection for visual SLAM using PCANet features, in *2016 international joint conference on neural networks (IJCNN)* (IEEE, Canada, 2016), pp. 2274–2281
47. V. Fox, J. Hightower, L. Liao, D. Schulz, G. Borriello, Bayesian filtering for location estimation. IEEE Pervasive Comput. **2**(3), 24–33 (2003)
48. S. Thrun, Probabilistic robotics. Commun. ACM **45**(3), 52–57 (2002)
49. C. Stachniss, A short introduction to the bayes filter and related models, http://ais.informatik.uni-freiburg.de/teaching/ws12/mapping/pdf/slam03-ekf.pdf
50. Randall Smith, Matthew Self, Peter Cheeseman, *Estimating uncertain spatial relationships in robotics*. Autonomous robot vehicles (Springer, New York, NY, 1990), pp. 167–193
51. R. Mur-Artal, J.D. Tardós, Orb-slam2: an open-source slam system for monocular, stereo, and rgb-d cameras. IEEE Trans. Robot. **33**(5), 1255–1262 (2017)
52. Dorian Galvez-Lopez, Juan D. Tardos, Bags of binary words for fast place recognition in image sequences. IEEE Trans. Robot. **28**(5), 1188–1197 (2012)
53. A. Geiger, P. Lenz, R. Urtasun, Are we ready for autonomous driving? the kitti vision benchmark suite, in *2012 IEEE Conference on Computer Vision and Pattern Recognition* (IEEE, Providence, RI, 2012), pp. 3354–3361

Author Biographies

M. Fatih Aslan is a research assistant at Karamanoglu Mehmetbey University (KMU), Karaman, Turkey. After completing his BSc with a high degree in Selçuk University (SU), Konya, Turkey in 2016 he started to work in Karamanoglu Mehmetbey University in 2017. In 2018, he completed his master's degree at Selcuk University. He is currently a PhD student in Electrical and Electronic Engineering at Konya Technical University. His research interests include robotics, image processing, machine learning and object tracking.

Akif Durdu Akif Durdu has been an associate professor with the Electrical-Electronics Engineering Department at the Konya Technical University (KTUN) since 2013. He earned a PhD degree in Eletrical-Electronics engineering from the Middle East Technical University (METU), Ankara, Turkey in 2012. He received his B.Sc. degree in Eletrical-Electronics Engineering in 2001 at the Selcuk University, Konya, Turkey. His research interests include intelligent control systems, autonomous robotics systems, search & rescue robotics, human-robot interaction, multi-robots networks and sensor networks. Dr. Durdu is teaching courses in control engineering, robotics and mechatronic systems.

Abdullah Yusefi received his B.Sc (2011) in Computer Science from Kabul Univ. in Kabul, Afghanistan and the M.E. (2014) in Computer Engineering from Osmania Univ., Hyderabad, India. He is currently working on his Ph.D. in Computer Engineering at Konya Technical Univ., Konya, Turkey. His research interests lie in the general area of autonomous systems, particularly in localization, sensor fusion and probabilistic state estimation models, as well as their applications in decision making, autonomous navigation, SLAM and multi-agent systems.

Kadir Sabanci was born in 1978. He received his B.S. and M.S. degrees in Electrical and Electronics Engineering (EEE) from Selcuk University, Turkey, in 2001, 2005 respectively. In 2013, he received his Ph.D. degree in Agricultural Machineries from Selcuk University, Turkey. He has been working as Assistant Professor in the Department of EEE at Karamanoglu Mehmetbey University. His current research interests include image processing, data mining, artificial intelligent, embedded systems and precision agricultural technology.

Cemil Sungur has been a full professor with the Electrical-Electronics Engineering Department at the Konya Technical University (KTUN) since 2017. He earned a PhD degree in Eletrical-Electronics engineering from the Selcuk University (SU), Konya, Turkey in 2002. He received his B.Sc. degree in Electric Education in 1979 at the Gazi University, Ankara, Turkey. Prof. Dr. Sungur is teaching courses in industrial electric-electronics, advanced automation systems and industrial automation and advanced PLC programming.