Proceedings of the 2nd International Conference on Machine Learning and Automation
DOI: 10.54254/2755-2721/93/2024BJ0055

# Study of the characteristics and application scenarios of three SLAM algorithms based on comparative methods

**Qichen Wang**

The High School Affiliated to Beijing Normal University, Beijing, China

jkirlin10@liberty.edu

**Abstract.** This paper compares three Simultaneous Localization and Mapping (SLAM) algorithms. SLAM algorithms are the core technology for autonomous navigation and environmental perception of mobile robots. SLAM algorithms are used by mobile robots to perceive the surrounding environment, build up an environment map and position themselves in real-time in an unknown environment. This article first systematically reviews the basic principles of each algorithm based on experiments and studies that have been completed by previous researchers and illustrates their respective unique mechanisms for processing sensor data, map construction, and localization. Subsequently, this paper analyzes the performance differences and characteristics of the three algorithms in practical applications in terms of robustness in complex environments, consumption of computing resources, and accuracy for generated maps. Finally, based on the advantages and disadvantages of each analyzed algorithm, this article summarizes the most suitable and unsuitable usage scenarios of different algorithms in specific situations. Moreover, this article puts forward specific algorithm selection suggestions for different scenarios to help engineers and researchers make more appropriate decisions in actual projects.

**Keywords:** Hector SLAM, Gmapping, cartographer, algorithm.

## 1. Introduction

In recent decades, the robotics industry and its positioning technology have made great progress, but there are still many gaps waiting for research in this field. Simultaneous localization and mapping (SLAM) is one of the most popular but relatively poorly researched fields in today's society [1]. Simply, the SLAM algorithm uses various sensors to help the robot perceive the environment, create a 2D/3D map of the nearby area in a complex and dynamic environment, and simultaneously estimate the position of the robot in the area. The process of SLAM is critical for the navigation and obstacle avoidance functions of all kinds of robots, since the SLAM algorithm ensures the robot to clarify the position of itself and the target point to reach in an unknown and dynamic environment, preparing for path planning. Besides, SLAM algorithm also helps robots to obtain information about obstacles in the environment, preparing for taking actions to avoid obstacles. Because of the special functions of the SLAM algorithm, it is widely used in mobile robots, drones, autonomous driving and other fields [2].

SLAM algorithms can be classified according to different aspects. One way to classify different SLAM algorithms is based on the type of sensor the algorithms use. The types include vision-based SLAM, including monocular SLAM, stereo SLAM, and red-green-blue-depth (RGB-D) SLAM; lidar-

DOI: 10.54254/2755-2721/93/2024BJ0055

based SLAM, including 2D Lidar SLAM and 3D Lidar SLAM; Inertial Measurement Unit (IMU)-Aided SLAM, including visual-inertial SLAM and lidar-inertial SLAM; and multi-sensor fusion SLAM. Even though lidar sensors are expensive, they are widely used in robot SLAM algorithms due to their high speed and high accuracy when generating environment maps [3]. Because of its fastness, simplicity and the characteristic that does not consume a lot of memory of lidar-based SLAM, it has become the preferred method for indoor robots to perceive the surrounding environment [3]. Because of the wide application of lidar-based SLAM algorithms, this work selects this type of algorithm as the main research object.

Many scholars have completed high-quality summaries of vision-based SLAM algorithms, but there is less work on summarizing lidar-based SLAM algorithms [2]. Therefore, this article aims to select three different radar-based SLAM algorithms-Hector SLAM, Gmapping, and cartographer- and summarize and analyze their core principles, characteristics and usage scenarios. For these three algorithms, there have been many in-depth studies on their principles, so this article will not elaborate too much on the principles of various algorithms. The focus of this article is to analyze the characteristics and performance of these three algorithms and discuss and summarize their applicable scenarios. This study uses a comparative research method to compare the advantages and disadvantages of the three algorithms and explore in what circumstances they can achieve the best results and in what scenarios they are not the first choice. This study mainly summarizes the characteristics of different lidar-based SLAM algorithms developed by predecessors and provides effective suggestions and inspirations for the future application of these algorithms. This work hopes to provide reasonable suggestions for scholars to select appropriate SLAM algorithms in specific environments.

## 2. Algorithm comparison

### *2.1. Hector SLAM*

*2.1.1. The principle of Hector SLAM.* Hector SLAM is a method of mapping and positioning in 2D space using data from lidar sensors. When the lidar sensor collects and transmits data about the surrounding environment to the central processor, an initial map blueprint is generated using Hector SLAM. More specifically, Hector SLAM processes the data collected by the lidar sensor about the distance between the walls or obstacles from the sensor. Then, Hector SLAM produces a preliminary orientation map of the surrounding walls and obstacles. Next, when the sensor moves to another location of the environment, the algorithm will modify the previous maps. The algorithm compares the current lidar scan data to previously generated maps [4]. Thereby estimating the pose (position and orientation) of the robot [4]. This process includes comparing current and previous data from lidar sensors and optimizing the match of them. By minimizing the error between the current data and the map, the estimated updated pose of the robot is obtained. The pose of the robot can be expressed as equation (1) [5].

$$\xi = (x, y, \psi)^T \tag{1}$$

It will continuously be updated based on scan-matching results. After each lidar scanning the environment, the pose change $\Delta\xi$ could be calculated [6]. Then, by adding a pose change $\Delta\xi$ to the previous pose, the algorithm will generate the new pose [6]. The update formula is equation (2).

$$\xi_k = \xi_{k-1} + \Delta\xi \tag{2}$$

After the new pose is recalculated, the environment map will be accurately revised, and the position of the robot will be calibrated. By repeating the above process, an accurate map of the environment can be drawn by Hector SLAM.

*2.1.2. The performance of Hector SLAM.* One of the biggest features of Hector SLAM is that its application does not require an odometer [6]. This feature greatly reduces the complexity of data fusion,

DOI: 10.54254/2755-2721/93/2024BJ0055

improves the update rate, and increases the accuracy of the generated maps [6]. The absence of odometer data means that the amount of data that should be processed by Hector SLAM is reduced. This process directly cuts back the computational burden on the algorithm. As a result, computing resources can be more focused on lidar data processing, further reducing latency and improving the overall speed and response time of the system to process and respond. Furthermore, the absence of an odometer can also increase the speed of the updated positioning of the system and enhance the real-time response capabilities of the system [6]. Besides, because Hector SLAM does not rely on an odometer, it will not cause calculation errors due to wheel slippage or uneven ground [4]. In this way, it can maintain better performance in some complex environments such as smooth floors. Since the Hector SLAM algorithm uses the Gauss-Newton method for matching, the amount of calculation and the pressure of computation are greatly reduced [3]. This feature further enhances Hector SLAM's ability to update the environment in real time and its processing speed. In addition, Hector SLAM also uses Grid Map as an environment representation method [6]. Because of this feature, robots that use Hector SLAM have better robustness to noise and various environmental changes. Moreover, when the sensor updates new data, Grid Map can help the algorithm effectively update the status to the relevant grid, maintaining the real-time nature of the map [7]. However, the use of Grid Map reduces the accuracy of the map to a certain extent. Grid Map needs to consume a lot of memory to store all raster information, especially for large and complex environments. In addition, there are some shortcomings in using Hector SLAM. The shortcomings include that Hector SLAM can only help the robot perceive the overall terrain and environment, and has no way to perceive small obstacles such as chairs and flower pots within the environment [3].

*2.1.3. Application scenarios of Hector SLAM.* In general, Hector SLAM has the following advantages. Hector SLAM has a fast-processing speed and short delay. Hector SLAM has strong real-time performance, which allows the robot to quickly update the map and environment in a short time. Moreover, it can efficiently perceive the environment and construct maps in smaller environments. In addition, it also allows the robot to work well in special circumstances, such as smooth floors and noisy environments. However, Hector SLAM also has some flaws when utilizing. The shortcomings of Hector SLAM include its inability to draw detailed environments and maps in vast environments. Hector SLAM cannot sense smaller obstacles either. Based on the advantages and disadvantages of Hector SLAM, Hector SLAM is most suitable for small-scale indoor environments and is not suitable for large-scale and complex outdoor environments. More specifically, indoor environments include home environments, offices, shopping malls, supermarkets, or larger warehouse environments or specific factories. In the small-scale indoor environment similar to the circumstances described above, Hector SLAM can play its strengths. Hector SLAM can perceive the entire environment efficiently and meticulously. For example, at home, where the rooms are relatively small and have a fixed layout, Hector SLAM can quickly build an environment map and navigate. In an indoor environment, it can take advantage of the benefits of Hector SLAM without an odometer, such as adapting to a variety of complex environments (smooth and noisy). Smooth floors and noisy environments, for example, are a true simulation of an office environment. Moreover, the indoor environment is more changeable and complex than the outdoor environment, which can give full play to the real-time performance of Hector SLAM. By applying Hector SLAM, people coming and going in the supermarket, for example, can be well perceived. However, in outdoor scenes, such as parks, roads, etc., using Hector SLAM is not a good choice. Due to the vast outdoor environment, Hector SLAM cannot perceive the environment in detail, hindering the robot from working effectively.

### *2.2. Gmapping*

*2.2.1. The principle of Gmapping.* Gmapping is a SLAM algorithm based on Rao-Blackwellized Particle Filtering (RBPF) [1]. Gmapping uses a grid map to represent the surrounding environment [7]. Gmapping estimates the status of the robot and senses the surrounding environment based on the data that scanned by the lidar sensor. Specifically, Gmapping will first calculate the pose and status of the

DOI: 10.54254/2755-2721/93/2024BJ0055

robot based on the data from the lidar sensor. The status and the pose of the the robot could be represented by equation (3) [8].

$$\left\{x_0, y_0, \theta_{0,\ldots}, x_k, y_k, \theta_k\right\} \tag{3}$$

Then, Gmapping calculates the posterior probability distribution. This step, as the core of the Gmapping algorithm, involves the process to detect the state of the robot and to build up a map through lidar sensor measurement and control input [1]. The posterior probability distribution calculated by Gmapping can be expressed by equation (4) [1].

$$p\left(x_{1:t}, m \middle| z_{1:t}, u_{1:t}\right) = p\left(m \middle| z_{1:t}, x_{1:t}\right) \cdot p\left(x_{1:t} \middle| z_{1:t}, u_{1:t}, x_0\right) \tag{4}$$

The meaning of the equation is to calculate the joint posterior distribution of robot status $x_{1:t}$ (the positions and poses from time 1 to t) and the current map $m$ when given the measurements of lidar sensor $z_{1:t}$ (distance information and data about the environment from time 1 to t that are measured by lidar sensors) and control input $u_{1:t}$ (robot motion instructions, including the robot's movement distance and rotation angle from time 1 to t). Gmapping utilizes particle filtering to generate maps [1]. After calculating the posterior probability distribution, Gmapping then updates the weight of each particle based on this distribution in order to gradually approach the real environment around the robot and the real state of the robot. Thus, Gmapping updates the probability of each grid being occupied by obstacles. Based on these probabilities, Gmapping can more accurately update the occupancy status of each grid unit, thereby constructing an accurate dynamic environment map. Finally, when the robot returns to an already familiar area, Gmapping will compare the sensor data with the existing map. In this way, Gmapping can identify and correct inconsistencies between the map and lidar data and adjust the map in time [8].

*2.2.2. The performance of Gmapping.* Like Hector SLAM, Gmapping utilizes a grid-based map to represent the environment. This approach provides a simple, intuitive and efficient way to represent and update maps [7]. However, since the use of grid maps comes with substantial memory storage and computational requirements, Gmapping is best suited for building two-dimensional maps [7]. For constructing three-dimensional maps, in contrast, Gmapping cannot perform well [7]. In addition, the closed-loop detection process enables Gmapping to perform well under high-resolution and high-frequency lidar scanners [8]. In contrast, the closed-loop performance of Gmapping is relatively poor when utilizing low-resolution and low-frequency lidar scanners [8]. In this case, Gmapping will face the problem of particle poverty, resulting in relatively high inconsistency in the constructed map. Gmapping also has another significant feature, that is, this algorithm utilizes both odometer and liar sensors [7]. The both use of odometry and lidar makes the data from the two complementary, allowing Gmapping to accurately sense the environment and estimate the position of robot. Even when lidar data acquisition is slow, the information provided by the odometry can help the algorithm update the robot's position in a short time. Moreover, using odometry and lidar at the same time can improve the robustness of the system and reduce errors caused by the use of a single sensor. However, the price of these advantages is the increase in computational complexity. Fusing multi-sensor data requires more complex calculations, which causes the algorithm to run slower. More specifically, in experiments by Mustafa Eliwa et al., the performance of Fast SLAM (Gmapping) and Hector SLAM in sensing the same environment was compared [9]. The data shows that the response time of drawing the map of the environment by Fast SLAM is 4 seconds, and the error of the map drawn by Fast SLAM is 0.7 centimeters [9]. And the response time for drawing the map of the environment by Hector SLAM is 2 seconds, and the error of the map drawn by Hector SLAM is 1.2 centimeters [9]. It can be seen that compared to the Hector SLAM algorithm, Gmapping has a slower response time, but it can draw a more accurate map than Hector SLAM. This shows that the computational complexity of Gmapping is higher than that of Hector SLAM, and the map drawn by Gmapping is more accurate.

DOI: 10.54254/2755-2721/93/2024BJ0055

*2.2.3. Application scenarios of Gmapping.* In general, the advantages of using Gmapping include that the maps generated by Gmapping are relatively accurate, and the robustness of the system that utilizes Gmapping is relatively high. However, Gmapping has a heavy computational and memory-stored burden, resulting in long latency and response time. Besides, the performance of Gmapping depends largely on the performance of the sensor. Based on the characteristics and the advantages and disadvantages of Gmapping, this algorithm is most suitable for two-dimensional scenes that require high resolution. Specifically, Gmapping can perform well in circumstances like autonomous driving cars, precision industrial robots, indoor navigation robots, home service robots, etc. For example, with the support of high-precision lidar, Gmapping can be used to generate accurate maps to assist vehicles in driving and avoiding obstacles. For another example, in buildings or factory workshops, a two-dimensional map is sufficient to meet the functions of a robot. Robots can utilize Gmapping to draw accurate two-dimensional maps for navigation and path planning. Beyond that, in situations like home service robots, low latency and immediate respond is not necessarily required. In this case, even if there is a delay, it will not have serious consequences. Therefore, the disadvantages of Gmapping's huge computational burden can be well offset in this situation. However, Gmapping cannot perform well in low-resolution 3D scenes. Specifically, using the Gmapping algorithm in scenarios like underground mines or field exploration, drones, or low-cost robots is not a good choice. For example, if the budget of the robot is limited, the performance of the sensor cannot be well guaranteed, which greatly reduces the performance of the Gmapping algorithm and decreases the accuracy of the generated map. For another example, drones require high real-time performance. The response delay of Gmapping cannot satisfy the application of drones in dynamic environments. Moreover, whether for field exploration or mine environment, three-dimensional map representation is required. However, Gmapping has poor performance in three-dimensional map representation. Therefore, in the above situations, Gmapping is not the best choice.

### *2.3. Cartographer*

*2.3.1. The principle of Cartographer.* Cartographer is a certain SLAM algorithm that can handle both 2D and 3D data [7]. It builds up a map and positions itself using the motion data from the IMU and the environmental data from the lidar sensor. Cartographer's fundamental components include Front-End Processing, Loop Closure, and Back-End Optimization. The main task of the Front-End Processing is to process the data from various sensors and generate local subgraphs [10]. Specifically, lidar sensors scan the surrounding environment and generates point cloud data. Then, combined with the data from the IMU, the pose of the robot is initially estimated, based on the equation (5), where $x_t$ is the estimate pose at time $t$, $f$ is kinematic model, $u_t$ is control input, and $w_t$ is process noise [10].

$$x_t = f\left(x_{t-1}, u_t\right) + w_t \tag{5}$$

The main purpose of Loop Closure is to identify and correct the pose error of the robot in the area. This process is realized through the loopback detection formula, as shown in equation (6), where $z_t^{loop}$ is the measurement result of loopback detection, $h$ is the measurement model, $x_{t-k}$ and $x_t$ are the estimates of current pose and past pose, and $v_t$ is the measurement noise [10].

$$z_t^{loop} = h(x_{t-k}, x_t) + v_t \tag{6}$$

Finally, the purpose of the Back-End Optimization is to integrate multiple local subgraphs into a global map. This function can be met by the equation (7), where $x$ is the set of all poses, $C$ is the set of all pose pairs, $z_{ij}$ is the measurement of poses from $i$ to $j$, and $\sum ij$is the covariance matrix of the measurement noise [10].

$$\min \sum_{(i,j)\in C} \left\| z_{ij} - h(x_i, x_j) \right\|^2 \sum ij \tag{7}$$

DOI: 10.54254/2755-2721/93/2024BJ0055

*2.3.2. The performance of Cartographer.* First of all, because Cartographer uses a method of constructing a local map and then integrating the local map into a global map, the computational complexity of the algorithm is reduced, allowing the algorithm could be effectively applied to large-scale environments [10]. In addition, compared with filter-based SLAM methods such as Gmapping, errors do not accumulate over time [2]. This is mainly because the graph optimization method used by Cartographer can globally optimize trajectories and reduce cumulative errors, performing better in large environments. Although graph optimization algorithms ensure the overall consistency of the map, graph optimization is very computationally intensive. This process requires processing large amounts of data, which may result in increased processing time and decreased frame rates [2]. Besides, the loop detection process helps reduce errors and improve map consistency. However, in highly similar environments, this may lead to incorrect identification of loopback points, increasing the inaccuracies of the map. Like other algorithms, Cartographer uses grid maps to represent the environment [11]. The advantages and disadvantages of using grid maps have been mentioned above. In addition, Cartographer also has the ability to generate 2D and 3D maps [7]. Because Cartographer can utilize and integrate multiple sensors, Cartographer can process 2D lidar data, 3D lidar data, and IMU data to help generate 2D and 3D maps. Besides, the approach of generating multiple submaps and optimizing them individually by the Cartographer allows the system to process both 2D and 3D information to produce maps.

*2.3.3. Application scenarios of Cartographer.* Overall, Cartographer offers some advantages, including the high consistency and accuracy of maps, the ability to create 2D and 3D maps, and the procedure of global optimization and reducing error accumulation. However, the computing load of Cartographer is relatively heavy. As a result, Cartographer has relatively long processing time and low frame rates. Besides, in highly similar environments, the algorithm will be inaccurate. In addition, because of the use of grid maps, memory consumption and resolution issues cannot be ignored. Based on these features of Cartographer, Cartographer is suitable for large-scale indoor and outdoor scenes. For example, in large warehouses, Cartographer can generate highly accurate 2D and 3D maps to help guide robots as they move between complex racks. In this case, the height of the shelves must be considered, so the 3D map is very important, making Cartographer a good choice. For another example, Cartographer is also very suitable for drones. Specifically, the 3D maps generated by Cartographer can help drones perform tasks such as structural measurements and post-disaster assessments. In these cases, the ability to perceive the large environment could be fully utilized and also takes advantage of building 3D maps, which other algorithms have not. However, Cartographer is not suitable for low-priced robots or environments that include narrow and highly similar corridors. For example, if the robot has limited computing resources, such as some low-cost home cleaning robots, Cartographer may not be a good choice because graph optimization requires high computing power to ensure map accuracy. Using Cartographer may cause robots to take a long time to make a movement, reducing real-time performance and user's experience. For another example, in hotel or office building environment, there are many similar long corridors. Therefore, Cartographer's loop detection may cause miscalculations, resulting in the generation of erroneous maps and reducing the accuracy of the map.

## 3. Conclusion

Through a comparative study of three algorithms-Hector SLAM, Gmapping, and Cartographer- the paper finds that Hector SLAM runs fast but has relatively low accuracy; Gmapping can generate accurate 2D maps and has high robustness, but has long latency and relies on sensor performance; and Cartographer can generate accurate 2D and 3D maps, but the algorithm has a large amount of computation burden and low map accuracy under certain conditions. Based on the characteristics of these three algorithms, the paper finds that Hector SLAM is most suitable for small-scale indoor environments and is not suitable for large-scale complex outdoor environments. Gmapping is suitable for high-precision-required 2D scenes but not for complex 3D scenes. Cartographer is suitable for large-scale indoor or outdoor environments but not for small and repetitive indoor environments. Hence, the article suggests that for robots used in outdoor wide scenes, Cartographer is the first SLAM algorithm

DOI: 10.54254/2755-2721/93/2024BJ0055

to be considered. For robots used in indoor environments, if the real-time requirements of the robot are high, Hector SLAM is suggested to be selected. For robots that do not require high real-time performance but require high accuracy, Gmapping is the best choice.

The contribution of the article is mainly to summarize the basic principles of the three algorithms-Hector SLAM, Gmapping, and Cartographer-, analyze the characteristics of the three algorithms, and discuss the application scenarios of the three algorithms. The article not only offers direction and suggestions for the selection of SLAM algorithms for different robots but also provides insights into the performance of different algorithms in different environments. This is helpful for researchers to select appropriate SLAM algorithms based on specific tasks, thereby improving the accuracy of the map generated and the efficiency of localization. Although summarizing and comparing three algorithms, the paper does not examine the performance differences of each algorithm in specific scenarios in depth. And the recommendations from the article might not always be appropriate in some unique situations. More in-depth application scenarios can be summarized in future studies by refining particular situations and comparing and analyzing the performance of various algorithms.

## References

[1] Teame, W. G., Zhongmin, W., & Yu, Y. (2020). Optimization of SLAM Gmapping based on Simulation. *Int. J. Eng. Res. Technol*, *9*, 74-81.

[2] Zhou, Z., Cao, J., & Di, S. (2023). A review of 3D LiDAR SLAM algorithms. *Chinese Journal of Scientific Instrument*, *(9)*, 13-27.

[3] Hakim, H., Alhakeem, Z. M., & Fadhil, A. (2021). Asus Xtion Pro Camera Performance in Constructing a 2D Map Using Hector SLAM Method. *IRAQI JOURNAL OF COMPUTERS, COMMUNICATIONS, CONTROL AND SYSTEMS ENGINEERING*, *21*(3), 1-11.

[4] Can, A., Price, J., & Montazeri, A. (2022). A Nonlinear Discrete-Time Sliding Mode Controller for Autonomous Navigation of an Aerial Vehicle Using Hector SLAM. *IFAC-PapersOnLine*, *55*(10), 2653-2658.

[5] Sim, R., & Roy, N. (2005, April). Global a-optimal robot exploration in slam. In *Proceedings of the 2005 IEEE international conference on robotics and automation* (pp. 661-666). IEEE.

[6] Vanicek, P., & Beran, L. (2018). Navigation of robotics platform in unknown spaces using LIDAR, Raspberry PI and hector slam. *Journal of Fundamental and Applied Sciences*, *10*(3S), 494-506.

[7] Li, L., Schulze, L., & Kalavadia, K. (2024). Promising SLAM Methods for Automated Guided Vehicles and Autonomous Mobile Robots. *Procedia Computer Science*, *232*, 2867-2874.

[8] Wang, P., Chen, Z., Zhang, Q., & Sun, J. (2016). A loop closure improvement method of Gmapping for low cost and resolution laser scanner. *IFAC-PapersOnLine*, *49*(12), 168-173.

[9] Eliwa, M., Adham, A., Sami, I., & Eldeeb, M. (2017). A critical comparison between Fast and Hector SLAM algorithms. *REST Journal on Emerging trends in Modelling and Manufacturing*, *3*(2), 44-49.

[10] Xu, J., Wang, D., Liao, M., & Shen, W. (2020, November). Research of cartographer graph optimization algorithm based on indoor mobile robot. In *Journal of Physics: Conference Series* (Vol. 1651, No. 1, p. 012120). IOP Publishing.

[11] Dwijotomo, A., Abdul Rahman, M. A., Mohammed Ariff, M. H., Zamzuri, H., & Wan Azree, W. M. H. (2020). Cartographer slam method for optimization with an adaptive multi-distance scan scheduler. *Applied Sciences*, *10*(1), 347.