Applying Graph-based SLAM Algorithm in a Simulated Environment

Nazrul Azril Badioze Zaman¹, Shuzlina Abdul-Rahman², Sofianita Mutalib³, Mohd Razif Shamsuddin⁴

¹²³⁴Research Initiative Group of Intelligent Systems, ¹²³⁴Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA, 40450 Shah Alam, Selangor, Malaysia
azril.nazrul@gmail.com, {shuzlina, sofi, razif}@fskm.uitm.edu.my

Abstract. This paper explores the capabilities of a graph optimization-based Simultaneous Localization and Mapping (SLAM) algorithm known as Cartographer in a simulated environment. SLAM refers to the problem in which an agent attempts to determine its location in the immediate environment as well as constructing the map(s) of its environment. SLAM is one of the most important aspects in the implementation of autonomous vehicle. In this paper, we explore the capabilities of the Cartographer algorithm which is based on the newer graph optimization approach in improving SLAM problems. A series of experiments were tested in order to discover its Cartographer capabilities in tackling SLAM problems. Then, we compare the results of Cartographer with Hector SLAM, another graph-based SLAM algorithm. We present the results from the experiments which show some promising findings based on the amount of computer resources used and the quality of the map(s) produced.

Keywords: Autonomous Vehicle, Graph, Optimization, Simultaneous Localization and Mapping

1. Introduction

Autonomous Vehicle (AV) is not a new topic in today’s technology nomenclature. Every now and then, news and articles that report on the topic of AV are published by a number of media streams. AVs, as the name implies, are vehicles that can perform driving or navigation tasks independently and move on its own with little to almost no input from a human driver. To achieve a full automation state, an AV must be able to process complex data with various attributes such as high dimensionality and unstructured data coming from various sources [1]. One of the most important criteria is for the vehicle to be able to receive data from its surroundings in order to scan the environment it is currently in. As an AV does not possess any senses such as vision, hearing, smell, taste and touch like humans do, most of the time, the problem with AV development is how the developers want to make the vehicle capable of sensing the environment it is currently situated in and its current position in real time [2]. In navigation, this problem is called Simultaneous Localization and Mapping (SLAM). As the name suggests, SLAM can be split into two parts namely the localization and mapping. Localization helps the agent ‘know’ and ‘aware’ of its own position in that particular environment it is currently in at any given time, while mapping helps the agent to construct and update the surrounding map(s) of the environment. Agent in this context includes robots, unmanned and autonomous vehicle which uses SLAM. Essentially, SLAM algorithms were used to help the agent make an estimation of the next step(s) it should take by employing and combining various types of calculations. Over the years, many
algorithms have been developed in order to tackle this problem with each one of them varying in the results they are able to produce.

This research made full use of the Simulation Method in order to determine the capabilities of the SLAM algorithm in solving SLAM problems. Simulation method was chosen in order to cope with a number of limitations faced during the execution of this research such as not having a real, working AV or a mobile robot as an AV at our disposal and not having the expertise needed in order to carry out this experiment in the actual environment. Given these limitations, simulation helps in reducing the risks of any unforeseen incidents that may happen if this experiment is carried out in a actual environment [3]. Furthermore, it is also cost-effective, needing only computers to run the simulations instead of using a real car or a mobile robot that is costly, while still be able to deliver the same, if not a better result. This paper presents a case study based on a SLAM algorithm that was experimented in a simulated environment in order to access its capabilities in improving SLAM problems. The case study examines the capabilities of Cartographer algorithm and compared with Hector SLAM algorithm. The remainder of this paper is organized in the following order: Section 2 describes the terminologies that are used in this experiment that is SLAM and Cartographer. Section 3 presents the methodology and the experiments setup. Section 4 presents the results and findings from the experiments undertaken, and Section 5 concludes the paper.

2. Background Study
   2.1. Simultaneous Localization and Mapping (SLAM)

Localization refers to the ability of an agent or vehicle to locate and keep track of its position in an unknown environment at any given time, while mapping refers to the process of constructing map(s) of the surroundings. SLAM refers to doing both of these tasks at the same time [4]. SLAM is important in autonomous navigation as it helps the agent to scan the environment it is currently in so that the agent can take the next steps in navigating such as the path-planning and decision-making [5][6]. As a result, SLAM enables the agent to move in an environment that it has no prior knowledge [7].

Over the years, a number of algorithms have been developed by researchers in order to solve SLAM problems based on the sensors that the agent used to scan the environment. At its core, SLAM can be categorized into two categories namely SLAM algorithm that uses particle filter and graph optimization. Particle usage has been proven essentially to solve various types of optimization, path finding and searching problems such as vehicle routing problem, vehicle assignment problem and others [8][9]. The use of particle filter for SLAM problems was first introduced in the 1980s. In the early days, SLAM algorithm that used particle filter estimated the position of an agent with landmarks by filtering out any noises in the environment [10]. With the advancement of technology, new methods were introduced to improve particle filter SLAM by adding features such as using the agent pose and data from the sensors to estimate the position of the agent [11][12]. Some examples of SLAM algorithm that is based on particle filter include Extended Kalman Filter (EKF), Unscented Kalman Filter (UKF), GMapping and FastSLAM [5][13].

Graph optimization is a newer approach to solve SLAM problems. First introduced in the 2000s, graph optimization consists of two processes namely graph construction and optimization [14]. The graph construction responsible for estimating the movement of an agent by constructing the image of the surroundings and detect changes in the image while the optimization process is used to optimize the agent’s pose in the current environment. Graph optimization has been improved in recent times by adding sensors such as cameras and lasers to improve the quality of SLAM problems. Examples of graph optimization-based SLAM algorithms include algorithms such as Real-Time Appearance-Based (RTAB-MAP), Stereo Parallel Tracking and Mapping (S-PTAM), ORB SLAM, ORB SLAM2, Hector SLAM, GraphSLAM, LSD-SLAM and Cartographer [15][13]. Figure 1 summarizes an overview of SLAM taxonomy compiled and adapted from various research articles that can be categorized into two main categories namely filter-based and vision-based. Due to the page limitation, this paper is unable to describe this taxonomy in greater detail.
2.2. Cartographer Algorithm
Cartographer is an algorithm developed by Google used to tackle SLAM problems [16]. Cartographer utilizes laser sensors such as the Light Detection and Ranging (LiDAR) to recognize the environment instead of cameras. As stated by the developers, Cartographer does not use particle filters, which allows it to use less resources compared to other algorithms. In order to overcome error accumulation from the lack of any filtering, Cartographer uses pose optimization approach. In this approach, the laser scans the environment, the data of which in turn will be inserted into a submap. When a submap is completed, in which no new scan by the laser will be inserted, the submap will be used as a consideration for a loop closure. The completed submap will be compared with the agent’s current pose. If there is a similar match between the agent’s current pose in the submap, the submap will be used as a loop closure constraint to the optimization problem. As the optimization happens at such a fast rate, the loop closure happens every time an area is revisited. Hence, the loop closing process has to happen quickly before a new scan is registered by the laser. The efficiency of Cartographer algorithm during the loop closing process makes it stand out from the other algorithms [16].

3. Methodology
3.1. Establishing Workspace
This research, in its entirety, was conducted using the Ubuntu Operating System (UOS). The reason for the use of the system is because this research fully utilizes the use of the Robotic Operating System (ROS) in which it is only available on the Ubuntu OS at the time of this research was carried out. With that being said, the first major step in this research development was installing the Ubuntu OS into the laptop. This research used the Ubuntu version 16.04 to ensure that it was compatible with the version of ROS that was used. Ubuntu was installed in a dual boot method to accommodate the lack of resources. Hence, Ubuntu resided in the same laptop as the Windows OS. After successfully installing Ubuntu, the next step was to install the ROS on Ubuntu. The version of ROS that was used in this research is called ‘Kinetic Kame’. Kinetic Kame is a version of ROS that received Long Term Support (LTS) from the ROS development team, which means it is much more stable and less buggy compared to other versions of ROS. Like any other applications on Ubuntu, ROS was installed by inserting the necessary commands into the Ubuntu terminal.

3.2. Installing Cartographer Algorithm
Cartographer was installed the same way as ROS, through the use of commands in the Ubuntu terminal. The tutorial for Cartographer installation can be found on the ROS wiki [17]. After successfully installing Cartographer, the Cartographer folder containing all the resources to run the algorithm should appear on the home directory of Ubuntu.

3.3. Installing Hector SLAM
The Hector SLAM package used in this project was retrieved from github instead of using the official package hosted on ROS wiki. The github version of the package has been modified to include integration with an agent that was used in the simulation without needing to install any additional dependencies. Hector SLAM was installed the same way as Cartographer, by inputting command into the terminal.
3.4. Running Simulation
After successfully installing the Cartographer algorithm, the next step taken was running the simulation to test the algorithm. Running these algorithms required calling a launch file, which contains all of the information of the processes that will be used during the simulation process.

4. Results and Findings
4.1. Computer Resource Usage
The computer resource usage was checked by using the system monitor program in the Ubuntu. Similar to the Task Manager in Windows OS, system monitor shows the processes of the programs and applications running. System monitor also shows the Central Processing Unit (CPU) usage and Random-Access Memory (RAM) usage of the processes running. In this experiment, the CPU and RAM usage were recorded while the Cartographer and Hector SLAM were simulated in the maze environment. No other processes were started during this experiment to get better readings on the computer resource usage of the Cartographer and Hector SLAM algorithms. This experiment was repeated ten times to get the average results for those two algorithms. Figures 2 and 3 show the results for both Cartographer and Hector SLAM on the first run.

![Figure 2. CPU and RAM usage while running Cartographer](image)
![Figure 3. CPU and RAM usage while running Hector](image)

Table 1 shows the overall results from the experiment. From the results table, Cartographer uses an average of 89.18% of CPU resources while Hector SLAM uses 91.82%. As for the RAM usage, Cartographer recorded an average of 55.2% while Hector SLAM uses 83.6%. Although there are no significant improvements between Cartographer and Hector SLAM in the CPU usage area i.e. a difference of 2.64%, it can be clearly seen that Cartographer uses a lot less RAM compared to Hector SLAM i.e. a difference of 28.4%. These findings directly correlate with the observation that Cartographer uses less resources overall as it utilizes pose graph optimization instead of particle filter.

| Experiment | Cartographer | Hector |
|------------|--------------|--------|
|            | CPU (%) | RAM (%) | CPU (%) | RAM (%) |
| 1          | 88.45   | 51.2    | 92.55   | 82.0    |
| 2          | 89.22   | 55.1    | 92.20   | 84.3    |
4.2. Map Accuracy
The next experiment was conducted to test mapping capabilities of both algorithms. This experiment was conducted in a playpen environment instead of the maze and the suburb, where it is much easier to map the whole environment, which has less obstacles compared to the other two environments. For this research, the maps produced were displayed in RViz, a 3D visualization tool that comes together with a full ROS desktop installation. The generated maps can be seen by the greyed-out area of the environment in RViz. Figure 4, 5 and 6 shows the environment, layout of the environment as mapped by Cartographer algorithm and layout of the environment as mapped by Hector SLAM respectively. In this experiment, both algorithms show that they were able to capture and map the outline of the environment perfectly. This can be seen clearly by observing the environment in Gazebo and the map shown in RViz.

Figure 4. Environment used in this experiment
It is worth to note that the map generated by Hector SLAM shows some anomalies in which the greyed-out area passes through what seems to be a static object and walls. In contrast, Cartographer did not encounter this problem and was able to produce a much sharper and clearer map of these two environments. As shown in Figure 7, the image on the left was adapted from the map produced by Cartographer, while the image on the right was from Hector SLAM. These problems (anomalies) may be caused by the lack of loop closure in Hector SLAM or various other reasons that are unidentifiable by the time this research was completed.

5. Conclusion
This paper shows the capabilities of graph optimization-based SLAM algorithm to work in a simulated environment. SLAM algorithms known as Cartographer and Hector were tested to uncover their capabilities in handling SLAM problems. Cartographer can be said to be more efficient when it comes to the resources used to run it. This can be observed during the first experiment where Cartographer recorded a significantly lesser computer resources used, mainly the CPU and RAM usage, compared to Hector SLAM. Cartographer was also able to produce a clearer and sharper map compared to its counterpart in this research the Hector SLAM. This is very important as the feature of SLAM as discussed earlier is the mapping part. In the future, a more in-depth work can be done with the Hector SLAM algorithm in order to identify what causes the anomalies during the mapping experiment. The Cartographer algorithm also can be compared with other SLAM algorithm with additional parameters in order to evaluate its capabilities in handling SLAM problem. Lastly, more research will be needed to experiment with other types of SLAM algorithms and to include more obstacles of various forms.

Acknowledgment
The authors would like to thank the Research Management Centre (RMC) under the grant of FRGS 600-IRMI/FRGS 5/3 (461/2019) of Universiti Teknologi MARA (UiTM) and the Faculty of Computer and Mathematical Sciences for all the given support.
References

[1] S. Abdul-Rahman, A. A. Bakar, and Z. A. Mohamed-Hussein, “An intelligent data preprocessing of complex datasets,” *Intell. Data Anal.*, vol. 16, no. 2, pp. 305–325, 2012.

[2] T. T. O. Takleh, N. A. Bakar, S. A. Rahman, R. Hamzah, and Z. A. Aziz, “A brief survey on SLAM methods in autonomous vehicle,” *Int. J. Eng. Technol.*, vol. 7, no. 4, pp. 38–43, 2018.

[3] S. Abdul-Rahman, M. S. Abd Razak, A. H. Mohd MushiN, R. Hamzah, N. Abu Bakar, and Z. Abd Aziz, “Simulation Of Simultaneous Localization And Mapping Using Point Cloud Data,” *Indones. J. Electr. Eng. Comput. Sci.*, vol. 14, no. 2, pp. 1–10, 2019.

[4] O. Agunbiade and T. Zuva, “A Review: Simultaneous Localization And Mapping In Application To Autonomous Robot Agunbiade OY and Zuva T,” no. May, 2018.

[5] Z. Kong and Q. Lu, “A brief review of simultaneous localization and mapping,” *Proc. IECON 2017 - 43rd Annu. Conf. IEEE Ind. Electron. Soc.*, vol. 2017-Janua, pp. 5517–5522, 2017.

[6] C. Cadena *et al.*, “Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age,” *IEEE Trans. Robot.*, vol. 32, no. 6, pp. 1309–1332, 2016.

[7] A. R. Khairuddin, M. S. Talib, and H. Haron, “Review on simultaneous localization and mapping (SLAM),” *Proc. - 5th IEEE Int. Conf. Control Syst. Comput. Eng. ICCSCE 2015*, no. November, pp. 85–90, 2016.

[8] M. Yusoff, J. Ariffin, and A. Mohamed, “Solving vehicle assignment problem using evolutionary computation,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6145 LNCS, no. PART 1, pp. 523–532, 2010.

[9] S. Masrom, S. Z. Z. Abidin, N. Omar, A. S. A. Rahman, and Z. I. Rizman, “Dynamic parameterizations of particle swarm optimization and genetic algorithm for facility layout problem,” *ARPN J. Eng. Appl. Sci.*, vol. 12, no. 10, pp. 3195–3201, 2017.

[10] R. Smith, M. Self, and P. Cheeseman, “Estimating uncertain spatial relationships in robotics,” in *Proceedings. 1987 IEEE International Conference on Robotics and Automation*, vol. 4, pp. 850–850.

[11] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, “FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem Michael,” pp. 593–598, 2002.

[12] G. Grisetti, C. Stachniss, and W. Burgard, “Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters,” *IEEE Trans. Robot.*, vol. 23, no. 1, pp. 34–46, 2007.

[13] M. Filipenko and I. Afanasyev, “Comparison of Various SLAM Systems for Mobile Robot in an Indoor Environment,” *2018 Int. Conf. Intell. Syst.*, no. September, pp. 400–407, 2018.

[14] N. Sünderhauf and P. Protzel, “Towards a robust back-end for pose graph SLAM,” *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 1254–1261, 2012.

[15] B. M. F. Da Silva, R. S. Xavier, T. P. Do Nascimento, and L. M. G. Gonsalves, “Experimental evaluation of ROS compatible SLAM algorithms for RGB-D sensors,” *Proc. - 2017 LARS 14th Lat. Am. Robot. Symp. 2017 5th SBR Brazilian Symp. Robot. LARS-SBR 2017 - Part Robot. Conf. 2017*, vol. 2017-Decem, pp. 1–6, 2017.

[16] Hess W, Kohler D, Rapp H, Andor D. Real-time loop closure in 2D LIDAR SLAM. In2016 IEEE International Conference on Robotics and Automation (ICRA) 2016 May 16 (pp. 1271-1278). IEEE.