

Simulation of simultaneous localization and mapping using point cloud data

Shuzlina Abdul-Rahman¹, Mohamad Soffi Abd Razak², Aliya Hasanah Binti Mohd Mushin³,
Raseeda Hamzah⁴, Nordin Abu Bakar⁵, Zalilah Abd Aziz⁶

^{1,6}Research Initiative Group of Intelligent Systems, Universiti Teknologi MARA, Malaysia
^{1,2,3,4,5,6}Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA, Malaysia

Article Info

Article history:

Received Jan 3, 2019

Revised Mar 7, 2019

Accepted May 25, 2019

Keywords:

Light detection and ranging
Localization
Mapping
Robot operating system
Simultaneous localization and mapping

ABSTRACT

This paper presents a simulation study on Simultaneous Localization and Mapping (SLAM) using point cloud data derived from the Light Detection and Ranging (LiDAR) technology. Methods like simulation are useful to simplify the process of learning algorithms, particularly when collecting and annotating large volumes of real data are both impractical and expensive. In this study, a map of a given environment was constructed using the Robotic Operating System (ROS) platform with Gazebo Simulator (GS). The paper begins by presenting the most currently popular algorithms that are widely used in SLAM namely the Extended Kalman Filter, Graph SLAM and Fast SLAM. The simulation of the Robot Operating System in MATLAB is also presented. The study performed the simulations by using standard SLAM with Turtlebot and Husky robots. Husky robot was further compared with the Adaptive Monte Carlo Localization (ACML) algorithm. The results showed that Hector SLAM could achieve the goal faster than ACML algorithm in a pre-defined map. Further studies in this field with other SLAM algorithms would certainly be beneficial to many parties due to the overwhelming demands of robotic applications.

Copyright © 2019 Institute of Advanced Engineering and Science.
All rights reserved.

Corresponding Author:

Shuzlina Abdul-Rahman,
Faculty of Computer and Mathematical Sciences,
Universiti Teknologi MARA,
Shah Alam, Selangor, Malaysia.
Email: shuzlina@tmsk.uitm.edu.my

1. INTRODUCTION

The last three decades have evidenced the growth of robotic applications in fields such as autonomous and unmanned aerial vehicles. This development is accompanied by an increasing challenge of developing the algorithms for the robots especially in dealing with real scenarios that are unsafe, expensive, or impractical in terms of data collection. Methods like simulation are useful to simplify the process of learning algorithms particularly when collecting and annotating large volumes of real data are both impractical and expensive [1]. In a simulated environment, the safety of systems and other hardware are generally not a concern. The simulation world can be procedurally constructed to specifications, allowing tests to be conducted especially under impractical and expensive conditions. Other past studies on simulation method can be found in [2-3].

Dealing with technology such as Light Detection and Ranging (LiDAR) deeply demands the application of the simulation method due to its sensitivity and cost. LiDAR has become one of the key technologies for remote sensing that is able to sample the entire environment and capture extremely accurate objects very quickly [4-5]. LiDAR is capable to collect more than one million points per second (pps) of high quality three-dimensional (3D) urban data [6]. The sensor is especially useful for accurate 3D of other man-made structures like road details, urban furniture or vegetation [7]. The most well-known manufacturer of

LiDAR is Velodyne [8-9]. The Velodyne High Definition LiDAR (HDL) unit provides a 360-degree azimuth field of view and a 26.5 degree elevation field of view, up to 15 Hz frame refresh rate, and rich point cloud populated at a rate of one million points per second [6]. LiDAR is also capable of recording every single data that bounces back to it every second. Undoubtedly, it has become a common fixture on autonomous vehicle run by companies like General Motors, Ford and Alphabet's Waymo [10].

Due to its robustness, this technology has encouraged many researchers to develop algorithms for scan matching, object detection and mapping [1, 5-7]. Since LiDAR applied 3D point cloud data, a good algorithm is needed to fully utilize all the data that has been produced. The algorithms must deal with the noise and large scale of data collected by LiDAR, thus developing the complex application would be greatly beneficial under simulation method. The objectives of this paper are to provide the simulation setup process, and to experiment a few standard SLAM algorithms for mapping and localization using Robotic Operating System (ROS). The remainder of this paper is organized as follows: Section 2 explains the Simultaneous Localization and Mapping (SLAM) and presents the commonly used algorithm in SLAM. Section 3 presents the simulation of ROS while Section 4 discusses the results of the simulation based on two robots. Finally, Section 5 concludes the paper.

2. SIMULTANEOUS LOCALIZATION AND MAPPING

Simultaneous Localization and Mapping (SLAM) was first coined in 1995 and was presented at the 7th International Symposium on Robotics Research [10]. SLAM is defined as a process by which a robot can map out an environment and deduce its location both at the same time. It refers to the simultaneous estimation of the state of a robot equipped with on-board sensors, and the construction of a model or the map of the environment that the sensors are perceiving [12-13]. The sensors such as LiDAR, camera, odometer, and inertial sensor will collect the data of its surroundings as it moves and placed on the robot. Other properties of the robot such as velocity, sensor biases, and calibration parameters would also influence the pose or the position and the orientation of the robot. The map, on the other hand, is a representation of aspects of interest that includes position of landmarks and obstacles describing the environment in which the robot operates, while localization refers to the position of the robot given a map [12]. Although SLAM has the capability to learn the map and location simultaneously, its implementation is challenging due to other problems like loop-closure and data association. Furthermore, SLAM is like a chicken-or-egg analogy in which we need either localization or mapping or both to do one of the processes. Nevertheless, SLAM is useful in a huge range of applications where absolute position and precise map information are unobtainable [14-15]. The following sub-sections present the most commonly SLAM algorithms in the field of autonomous vehicle.

2.1. Extended Kalman Filter

The Extended Kalman Filter (EKF) is the derivation of Kalman Filter algorithm in which improvements have been made in Kalman Filter algorithm to handle non-linear problems. It is a well-known multi-sensor fusion method that integrates the information from multiple sensors for more accurate and credible information [15]. The method is a mathematical model which utilizes optimization for estimation [11]. Thus, the SLAM implementation with EKF is considerably difficult due to the approximation of real-time stochastic type system, and sensor noise known as Gaussian. This improper noise may cause instability to the entire system [16]. The implementation of EKF with a Laser Range Finder (LRF), also known as Laser Scanner Sensor (LSS) was carried out in which the accuracy of SLAM was improved by incorporating the derived output matrices through least-square techniques [17]. This approach was able to increase the mapping accuracy and scalability [11]. Moreover, the advantage of EKF is that it assumes that the process models and observations are locally linear and can therefore be linearized [18]. In addition, the advantages are straightforward application, large body of research to derive from, works reasonably well for a small number of features and distinct landmarks [19]. However, it also has certain disadvantages, which are quadratic complexities with a number of features, no guarantee of convergence in non-linear case, makes hard decisions about data associations, cannot correct for erroneous data associations, and need sufficiently distinct landmarks [19]. The widespread use of EKF are for filtering noisy signals, generating non-observable states (estimating velocity), and predicting future states fast.

2.2. Graph-SLAM

One intuitive way of formulating SLAM is to use a graph whose nodes correspond to the poses of the robot at different points in time, and whose edges represent constraints between the poses [20]. The key idea of Graph-SLAM is building a graph, which contains all the places the sensor has previously visited and connects them. It uses a graph to represent the problem and every node in the graph corresponds to a pose of

the sensor during mapping. Every edge between the two nodes corresponds to the spatial constraints between them. The goal of this algorithm is to find nodes configuration that minimizes the error introduced by the constraints. Graph-SLAM has shown its advantages in large scale mapping, since a graph can be easily scarified to reduce redundant information and increase the accuracy of the estimation [21]. It also increases the quality of the map over time due to error convergence by separating back and front-ends. It also helps to complete path and gets optimized when an error is found [22]. The main disadvantages of graph-based SLAM is that it requires high memory computations as it incorporates all the pose estimates during the calculation process unlike EKF and other filtering methods. Graph-SLAM is hard to implement and difficult in the optimization of parameters [21]. In addition, the error is converged by iteration, which makes the algorithm less deterministic [22].

2.3. Fast SLAM

FastSLAM splits the SLAM problem into a robot localization problem, and a collection of landmark estimation problems that are conditioned on the robot pose estimate. It uses a modified particle filter for estimating the posterior over robot paths. According to a study by [23], FastSLAM allows autonomous mobile robots an ability to learn a consistent model of its environment, which is its prerequisite. FastSLAM is efficient to be applied to environments mapping because it can process far larger data than could be handled by the EKF [24]. This algorithm is capable of building maps with orders of magnitude with more landmarks than Kalman Filter. It can also handle a small number of particles which works well regardless of the number of landmarks under certain conditions. In addition, it is able to recover from false data association and can pursue multiple data associations simultaneously [25]. Long term process is an inconsistent stochastic filter but, as a heuristic estimator, it can be both tractable and highly accurate, while in short-term process it is able to produce consistent results given enough particles [26]. The advantages of the fastSLAM is that it does not consider the measurement acquired at time; instead, the measurement is through resampling [27]. Its current form cannot produce consistent estimates in the long-term and may produce quite accurate results but the estimate of its accuracy soon becomes optimistic. Many researchers underestimate fastSLAM due to its uncertainties because of the higher landmarks, or precise sensors or more frequent observations but will improve accuracy and also speed up particle depletion [26]. Nowadays, robot simulators have robust physics engines, high-quality graphics, and convenient interfaces, affording researchers to substitute physical systems with their simulation models in order to pre-estimate the performance of theoretical findings before applying them to real robots.

3. SIMULATION IN ROBOTIC OPERATING SYSTEM

Robotic Operating System (ROS) is a collection of software frameworks that support code reuse for robotics research and development. It was first developed by the Stanford AI Laboratory in 2007 for developing robotic applications and is currently maintained by The Open Source Robotics Foundation (OSRF) [28]. As mentioned by Quigley et al. [29], the ROS gives a structured communications layer above the host operating systems of a heterogeneous computer cluster unlike the traditional sense of process management and scheduling in operating systems. Since the main goal of ROS is to support the code reuse in robotics applications, it provides a built-in package system similar to any other services such as hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management [28]. Another good feature of ROS is that it does not have to be on the same system or architecture, and this promotes high flexibility and adaptability to user needs [30]. The paper is not intended to provide details about ROS, thus more details about ROS framework can be referred in Quigley et al. [31].

The list of softwares needed for this simulation includes Windows operating system, Ubuntu operating system, MATLAB, Robotic Operating System and the Gazebo Simulator. We need to have two different operating systems (OS) because the process will occur in two different environments. This is because the workstation with Windows OS will act as the host while the workstation with Linux OS will act as the simulator. The process will also proceed in the same workstation if the user has limitations with the number of computers. Setup of the simulation is by installing a virtual machine in the host workstation. Then, Ubuntu is installed in the Virtual machine. However, the performance is lesser since the workstation needs to work on two big tasks at the same time. The memory of the workstation will be fully loaded and slows the time for processing. In this setup, we used Windows 7 64 bit as the host, Ubuntu 18.04 LTS as the simulator, MATLAB R2018a and the Robot Operating System (ROS) with Gazebo simulator. We employed MATLAB R2018a [32] since it has ROS toolbox that will ease the process of controlling the robot in the gazebo. The gazebo simulator will run the virtual turtlebot as the robot. The virtual turtlebot is the same with the real one. TurtleBot is a low-cost, personal robot kit with open-source software. There are two types of

TurtleBots, real and virtual Turtlebots. The virtual TurtleBot can be used in the ROS and is suitable for mapping since it has LiDAR features.

3.1. Gazebo and Simulated TurtleBot Setup

This section presents the process of setting up the Gazebo simulator engine. It is a simulator that enables the user to perform the testing and experiment of physical scenarios. For this simulation, MATLAB is connected to Gazebo through the ROS interface. The simulator environment setup is performed in three steps as been described in [32]. The first step is to install a complete Linux OS and ROS with Gazebo in the other workstation or install the Virtual machine in the same workstation with the host. The second step is to download a virtual machine image that has been installed with Gazebo and ROS. This virtual machine is based on Ubuntu[®] Linux[®] OS which is pre-configured to ease the ROS examples in Robotics System Toolbox[™]. Once the installation process is completed, three Gazebo icons are created, namely "Gazebo Empty", "Gazebo Playground" and "Gazebo TurtleBot World". The first two Gazebo icons are used in the Gazebo examples while the last one is used in the TurtleBot[®] examples. For the ROS website, the suitable packages for TurtleBot are downloaded and the instruction in the website can be followed to get the Turtlebot running. The environment variables can be tested by pinging back and forth between the host and the Gazebo computer. The third step is to make/establish a connection between the host environments and the simulator environment. This step can be started by opening a new terminal in the Ubuntu virtual machine or the stand alone OS. The networking information for the virtual machine is shown by using ipconfig. As can be seen in Figure 1, under Under eth0, the inet addr displays the IP address for the virtual machine. The next step is to set up the network and this process requires two ROS environment variables: ROS_MASTER_URI and ROS_HOSTNAME. Once this process is performed, the environment variables are checked using echo \$ENV_VAR (the ENV_VAR can be replaced with the appropriate environment) while command of Close and Reopen is used to control the terminal for the effect to be seen. Figure 2 illustrates the correct environment variables assignments by using the fake IP addresses. On the host computer setup, the IP address of host computer on the network has to be identified. The command "ipconfig" can be used to display the network configuration for windows OS.

```

user@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:9e:0d:43
          inet addr:192.168.84.128  Bcast:192.168.84.255  Mask:255.255.255.0
          inet6 addr: fe80::c0e:29ff:fe9e:d43/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:7783253 errors:0 dropped:0 overruns:0 frame:0
          TX packets:15500746 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:471506606 (471.5 MB)  TX bytes:23421935293 (23.4 GB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16384  Metric:1
          RX packets:3473751 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3473751 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:541253053 (541.2 MB)  TX bytes:541253053 (541.2 MB)

user@ubuntu:~$

```

Figure 1. IP address for Ubuntu terminal


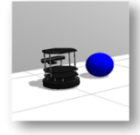
Host (MATLAB)	Gazebo Simulator
IP Address: 192.168.1.100	IP Address: 192.168.1.200
	
ROS_IP = 192.168.1.100	ROS_IP = 192.168.1.200
ROS_MASTER_URI = 192.168.1.200	ROS_MASTER_URI = 192.168.1.200

Figure 2. Environment variable

3.2. TurtleBot with Teleoperation Control

This section discusses the mechanism in controlling the TurtleBot using the keyboard as mentioned in [32]. The instructions are sent from the host environment to the simulator environment. These instructions are coded in MATLAB and are sent to the Gazebo simulator through ROS interface. The following instructions describe how to set up the object and how to start the keyboard control. The process begins by downloading a hardware support package for TurtleBot. This package allows users to collect sensor data and send control commands without explicitly calling ROS commands. It also allows users to communicate transparently with a simulated robot in Gazebo or with a physical TurtleBot. To install this package, the following command which can be found at the MATLAB Home tab by selecting "TurtleBot-Based Robots: "open Add-Ons > Get Hardware Support Packages". An alternative approach is by using the roboticsAddons command. Once the installation is completed, the connection to the TurtleBot can be established by initializing ROS. At this stage, the TurtleBot can be connected with ROS by replacing the sample IP address (192.168.1.1) with the IP address of the TurtleBot. The TurtleBot should run either in

simulation through Gazebo and a Simulated TurtleBot or get started with a real TurtleBot for the startup procedure. As stated in [32], Gazebo TurtleBot World would be a good choice under simulation. Then, the odometry and laser scan topics are subscribed in order to ensure the messages are received accordingly.

The next process is to create a publisher for controlling robot velocity. In controlling the robot, the exampleHelperTurtleBotKeyboardControl function was activated that permits the user to control the TurtleBot using the keyboard. Figure 3 shows a sample of a few command windows, and the Gazebo world after some keyboard teleoperation by the user. The TurtleBot needs to be moved slowly to produce a good mapping result. This is because the plotting process require some/sufficient time, thus if the movement of the TurtleBot is too fast this would result in a messy obstacle plotting due to imprecision in the odometry topic that runs at high speeds. Figure 4 shows an example of a messy world plot. Once the simulation is done, the bot from the Gazebo is disconnected to avoid any crash on the system. This can be done by pressing the “q” function, while the publishers and subscribers on the host can be cleared using the command “clear” before the ROS is shutdown. It is recommended to use “roshutdown” once the simulation is completed by shutting down the global node and to disconnect it from the TurtleBot.

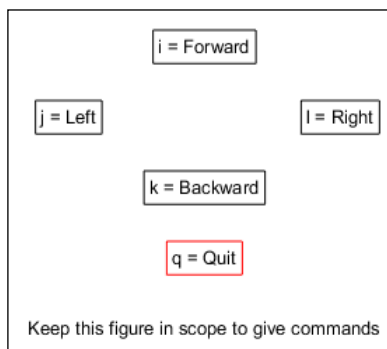


Figure 3. Samples of the command window

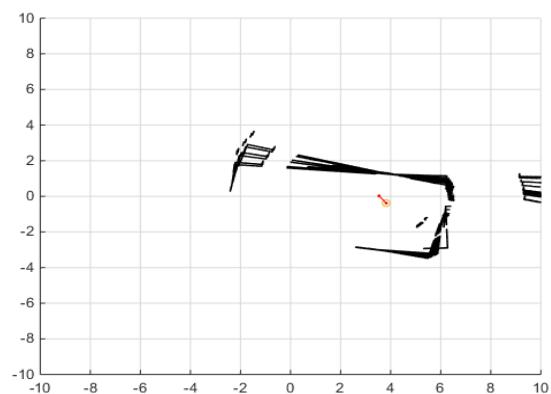


Figure 4. Samples of the messy world plot

4. RESULTS

In this section, simulation results with TurtleBot and Husky Robot are presented. Simulation with Husky robot utilizes the Hector SLAM and AMCL algorithms to compare time taken in reaching the goal.

4.1. Simulation Results with TurtleBot

In simulation with TurtleBot, scripts are done in MATLAB and the simulation is done on the Gazebo in ROS. The simulation using a standard SLAM and the algorithm is provided by the ROS toolbox in the MATLAB. The movement of the bot is controlled by the keyboard. Figure 5 shows the results before the TurtleBot moves and Figure 6 shows the results after the TurtleBot moves forward after pressing “i”. We can clearly see the graph on the right hand side of the image. As the bot moves, it maps the environment. In this study, the simulation uses standard SLAM only. The virtual TurtleBot, ROS and MATLAB are very useful and can be used in simulating and testing the algorithms.

4.2. Simulation Results with Husky Robot

This section presents the simulation result of Husky robot with Hector SLAM and AMCL algorithms. Hector SLAM requires low computational resources as it is available as open source based on ROS. It relies on scan matching and does not require loop closure as it is sufficiently accurate [33]. The scan matching uses a Gauss-Newton approach as it solves non-linear least squares problems based on occupancy grid maps [34]. This approach optimizes the alignment of beam endpoints with the current map [35]. Hector SLAM is a part of the linear square optimization-based SLAM [36]. Adaptive Monte Carlo Localization (AMCL) use resampling scheme that is both beneficial to a line-based sensor model, and which minimizes the error between real and sampled particle using Kullback Leibler Distance (KLD) [37]. Figures 7 and 8 show the localization of the mobile robot while creating the map of the environment in ROS environment. As one can see, the mobile robot could locate itself while creating the map of the unknown environment.

Table 1 presents the results of the five experiments between Hector SLAM and AMCL algorithms with different pairs of points. As can be seen, the Hector SLAM consistently requires less time in reaching the goal for all the points. In general, it took approximately 122.53 seconds on average to reach the goal while AMCL algorithm took 199.94 seconds. This is probably because Hector SLAM does not require loop closure and only depends on scan matching as compared to AMCL algorithm.

Table 1. Modelling Time between Hector SLAM and AMCL Algorithm

Experiments	Points	Time Taken to Reach Goal	
		Hector SLAM	AMCL algorithm
1	A to B	25.32 seconds	32.49 seconds
2	C to D	24.18 seconds	43.56 seconds
3	E to F	20.86 seconds	39.45 seconds
4	G to H	31.73 seconds	37.66 seconds
5	I to J	20.44 seconds	46.78 seconds
	Total	122.53 seconds	199.94 seconds

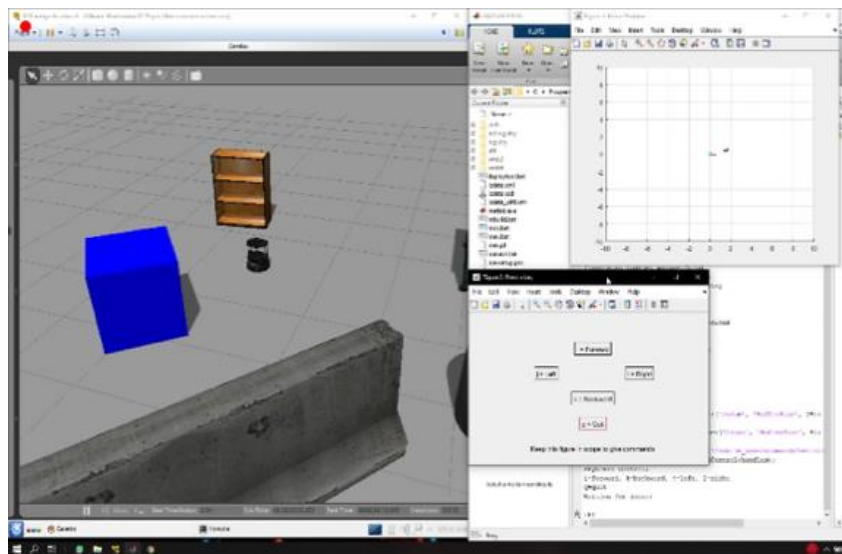


Figure 5. Result before the TurtleBot moves

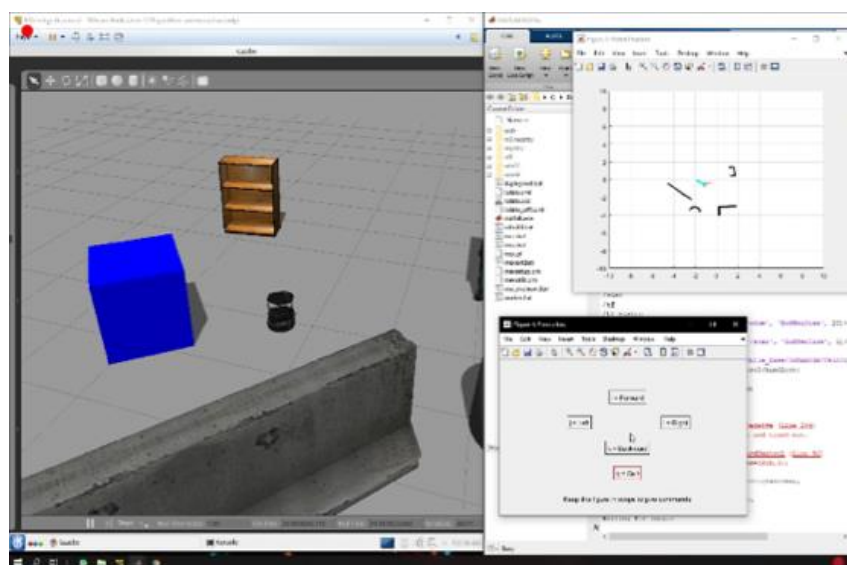


Figure 6. Result after the TurtleBot moves

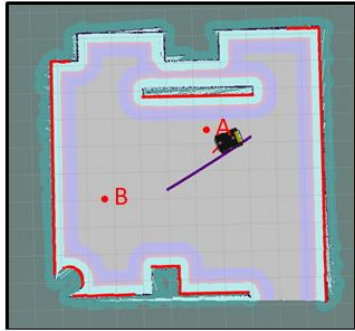


Figure 7. Experiment 1 with Hector SLAM from point A to B

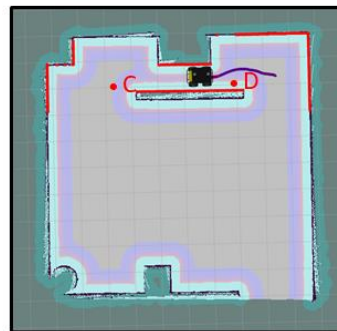


Figure 8. Experiment 2 with Hector SLAM from point C to D

5. CONCLUSIONS

This paper demonstrated the simulation study of SLAM using point cloud data derived from LiDAR technology. The map of a given environment was constructed with Gazebo Simulator in ROS. The ROS simulation setup in MATLAB was described as a guidance to other interested researchers. The study also presented the simulation experiments with Turtlebot which map the movement of the bot and the environment. Further, simulation experiment with Husky robot utilizes the Hector SLAM and AMCL algorithms are presented to compare time taken in reaching the goal. Results showed that Husky with Hector SLAM take less duration to complete the given task. Further studies in this field with other SLAM algorithms would certainly be beneficial to many parties due to the demands of robotic application.

ACKNOWLEDGEMENTS

Acknowledgements. The authors would like to thank the Research Management Centre (RMC) under the grant of TRGS (600-IRMI/KPT 5/3/TR (001/2017)-2) of Universiti Teknologi MARA (UiTM) and the Institute of Quality & Knowledge Advancement (InQKA) for the publication support.

REFERENCES

- [1] Markom, Marni Azira, et al. "A mapping mobile robot using RP Lidar scanner." 2015 *IEEE International Symposium on Robotics and Intelligent Sensors (IRIS)*. IEEE, 2015.
- [2] Chitumodhu, B., Loka, R. (2019), An open source tool for reliability evaluation of distribution system using Monte Carlo simulation, 14(3), *Indonesian Journal of Electrical Engineering and Computer Science (IJECS)*.
- [3] Al-Rawashdeh, A (2019), Simulation and analysis of the possibilities of traction electric motor, 14(1), *Indonesian Journal of Electrical Engineering and Computer Science (IJECS)*.
- [4] Murphy, Kevin E. "Light detection and ranging (LIDAR) mapping system." U.S. Patent No. 6,711,475. 23 Mar. 2004.
- [5] Teichmann, Marek, and Bud Mishra. "Probabilistic algorithms for efficient grasping and fixturing." *Algorithmica* 26.3-4 (2000): 345-363.
- [6] Hernández-García, Denis-Eduardo, et al. "3D city models: Mapping approach using lidar technology." CONIELECOMP 2011, 21st International Conference on Electrical Communications and Computers. IEEE, 2011.
- [7] Haala, Norbert, et al. "Mobile LiDAR mapping for 3D point cloud collection in urban areas — A performance test." *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci* 37 (2008): 1119-1127.
- [8] A. Hussey, "Velodyne Slashes the Price in Half of Its Most Popular LiDAR Sensor," Businesswire, 2018. [Online]. Available: <https://www.businesswire.com/news/home/20180101005041/en/Velodyne-Slashes-Price-Popular-LiDAR-Sensor>. [Accessed: 25-May-2018].
- [9] Velodyne LiDAR, April, 2019 [Online]. Available: <https://velodynelidar.com/products.html>. [Accessed April. 19, 2019].
- [10] Lee, T. B, "How 10 leading companies are trying to make powerful, low-cost lidar", Jan, 2019 [Online]. Available: <https://arstechnica.com/cars/2019/02/the-ars-technica-guide-to-the-lidar-industry/>. [Accessed April. 19, 2019].
- [11] Durrant-Whyte, Hugh, and Tim Bailey. "Simultaneous localization and mapping: part I." *IEEE robotics & automation magazine* 13.2 (2006): 99-110.
- [12] Kuzmin, Maxim. "Review, Classification and Comparison of the Existing SLAM Methods for Groups of Robots." In Proceedings of the 22st Conference of Open Innovations Association FRUCT, p. 16. FRUCT Oy, 2018.
- [13] Cadena, Cesar, et al. "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age." *IEEE Transactions on robotics* 32.6 (2016): 1309-1332.

- [14] Medina, Sergio, et al. "Localization and mapping approximation for autonomous ground platforms, implementing SLAM algorithms." 2014 III *International Congress of Engineering Mechatronics and Automation (CIIMA)*. IEEE, 2014.
- [15] Do, Cong Hung, Huei-Yung Lin, and Yi-Chun Huang. "Simultaneous localization and mapping with neuro-fuzzy assisted extended Kalman filtering." 2017 *IEEE/SICE International Symposium on System Integration (SII)*. IEEE, 2017.
- [16] Huang, Shoudong, and Gamini Dissanayake. "Convergence and consistency analysis for extended Kalman filter based SLAM." *IEEE Transactions on robotics* 23.5 (2007): 1036-1049.
- [17] Klančar, Gregor, Luka Teslić, and Igor Škrjanc. "Mobile-robot pose estimation and environment mapping using an extended Kalman filter." *International Journal of Systems Science* 45.12 (2014): 2603-2618.
- [18] Dissanayake, MWM Gamini, et al. "An experimental and theoretical investigation into simultaneous localisation and map building." *Experimental robotics VI*. Springer, London, 2000. 265-274.
- [19] Collier, Jack. "SLAM Techniques and Algorithms." (2009).
- [20] Grisetti, Giorgio, et al. "A tutorial on graph-based SLAM." *IEEE Intelligent Transportation Systems Magazine* 2.4 (2010): 31-43.
- [21] Oh, Taekjun, et al. "Graph-based SLAM (Simultaneous Localization And Mapping) for Bridge Inspection Using UAV (Unmanned Aerial Vehicle)." World Congress on Advances in Structural Engineering and Mechanics (ASEM). IASEM Conferences, 2017.
- [22] Appel, Robin, et al. "Analysis, optimization, and design of a SLAM solution for an implementation on
- [23] Hahnel, Dirk, et al. "An efficient FastSLAM algorithm for generating maps of large-scale cyclic environments from raw laser range measurements." Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453). Vol. 1. IEEE, 2003.
- [24] Montemerlo, Michael, and Sebastian Thrun. "Simultaneous localization and mapping with unknown data association using FastSLAM." 2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422). Vol. 2. IEEE, 2003.
- [25] Montemerlo, Michael, et al. "FastSLAM: A factored solution to the simultaneous localization and mapping problem." *Aaai/iaai* 593598 (2002).
- [26] Bailey, Tim, et al. "Consistency of the EKF-SLAM algorithm." 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2006.
- [27] Montemerlo, Michael, et al. "FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges." *IJCAI*. 2003.
- [28] Tawil, Yahya "An Overview of How ROS Works", June, 2017 [Online]. <https://www.allaboutcircuits.com/technical-articles/an-introduction-to-robot-operating-system-ros/>
- [29] Quigley, Morgan, et al. "ROS: an open-source Robot Operating System." *ICRA workshop on open source software*. Vol. 3. No. 3.2. 2009.
- [30] Robohub, "ROS 101: Intro to the Robot Operating System," Clearpath Robotics, 2014. [Online]. Available: <http://robhub.org/ros-101-intro-to-the-robot-operating-system/>. [Accessed: 24-May-2018].
- [31] Quigley, Morgan, Brian Gerkey, and William D. Smart. Programming Robots with ROS: a practical introduction to the Robot Operating System. "O'Reilly Media, Inc.", 2015.
- [32] Robotics System Toolbox, April, 2019 [Online]. Available: <https://www.mathworks.com/help/robotics/>[Accessed April. 19, 2019].
- [33] López, Elena, et al. "A multi-sensorial simultaneous localization and mapping (SLAM) system for low-cost micro aerial vehicles in GPS-denied environments." *Sensors* 17.4 (2017): 802.
- [34] Turnage, Doris M. "Simulation results for localization and mapping algorithms." 2016 Winter Simulation Conference (WSC). IEEE, 2016.
- [35] Balçılar, Muhammet, et al. "An architecture for multi-robot hector mapping." 2016 *International Symposium on Innovations in Intelligent Systems and Applications (INISTA)*. IEEE, 2016.
- [36] Ronzhin, Andrey, Gerhard Rigoll, and Roman Meshcheryakov, eds. Interactive Collaborative Robotics: *Third International Conference, ICR 2018, Leipzig, Germany, September 18-22, 2018*, Proceedings. Vol. 11097. Springer, 2018.
- [37] Hanten, Richard, et al. "Vector-amcl: Vector based adaptive monte carlo localization for indoor maps." International Conference on Intelligent Autonomous Systems. Springer, Cham, 2016.

BIOGRAPHIES OF AUTHORS



Shuzlina Abdul Rahman received her Bachelor in Computer Science degree from Universiti Sains Malaysia (USM) in 1996, Master of Science in Information Technology from Universiti Utara Malaysia (UUM) in 2000 and PhD in Science and System Management from Universiti Kebangsaan Malaysia (UKM) in 2012. She is currently working as an Associate Professor at the Faculty of Computer & Mathematical Sciences (FSKM), Universiti Teknologi MARA (UiTM), Shah Alam, Selangor, Malaysia. Her research interests include computational intelligence, machine learning and data analytics & optimization.



Mohamad Soffi Bin Abd Razak received his Bachelor in Information Technology degree from Universiti Teknologi MARA (UiTM), Shah Alam, Selangor, Malaysia majoring in Intelligent System Engineering in 2015 and Master in Information Technology degree from the same university in 2019. He is currently working as a Research Assistant at the Research Initiative Group of Intelligent Systems (RIGIS), UiTM. His research interests include intelligent system, image processing and environment mapping for autonomous navigation system.



Aliya Hasanah Mohd Mushin received her Diploma in Computer Science from UiTM Machang, Kelantan in 2016. She is currently doing her Bachelor in Intelligent Systems Engineering degree at UiTM Shah Alam, Selangor. She is in her last semester of her bachelor's degree and she is currently doing her internship at Datawiz Consultancy Services Sdn. Bhd. in Petaling Jaya, Selangor. The company is an established Information and Communication Technologies (ICT) service provider and software developer.



Raseeda Hamzah is currently working as a Senior Lecturer at the Faculty of Computer and Mathematical Sciences (FSKM), Universiti Teknologi MARA (UiTM), Shah Alam, Selangor, Malaysia. She received her bachelor and master degrees from Universiti Teknikal Malaysia (UteM), Melaka in 2007 and University of Malaya (UM) in 2010. She later received her PhD (Computational Linguistics) from UiTM Shah Alam in 2016. She is actively doing research in signal processing, pattern recognition and feature analysis for various areas and different types of data. Her research work has been published in annual publications and journals. She is a member of the Research Initiative Group (RIG) for Digital Image and Speech Technology (DIASST). She is also a group member of the FRGS and Lestari grant for ongoing research.



Nordin Abu Bakar is an Associate Professor and researcher at the Faculty of Computer and Mathematical Sciences (FSKM), UiTM, Shah Alam, Selangor, Malaysia. He received his Bachelor in Computer Science degree from Wichita State University (WSU), USA in 1986 and MSc in Computer Science degree from Bradley University, USA in 1988. He joined UiTM in 1990 and was assigned to UiTM Terengganu in Dungun to strengthen the fledgling Diploma in Computer Science program there. He was later given a scholarship to pursue his PhD in Computer Science at the University of Essex in the UK. His research interests include Artificial Intelligence, Machine Learning, Business Informatics and Finance. The on-going project that he is currently heading is the social innovation project involving the Orang Asli (Aborigines) community in Lojing, Gua Musang, Kelantan sponsored by NBOS, KPT and UiTM.



Zalilah Abdul Aziz is a senior lecturer at the Faculty of Computer and Mathematical Sciences (FSKM), Universiti Teknologi MARA (UiTM), Shah Alam, Selangor, Malaysia. After completing her Diploma and Bachelor Degree in Computer Science in UiTM, she worked as a Computer Programmer and System Analyst in numerous government agencies such as the Department of Statistics Malaysia, and the Malaysian Administrative Modernisation and Management Planning Unit (MAMPU). She joined UiTM as a lecturer in 1996, and in 2003 she received her Master in Software Engineering degree from Universiti Putra Malaysia (UPM). She obtained her PhD in Computer Science from University of Nottingham, United Kingdom in 2013. Her research interests include Artificial Intelligence, Computational Intelligence and Optimization, Software Engineering and System Development.