Penerbit Akademia Baru

Journal of Advanced Research Design

Journal homepage:
https://akademiabaru.com/submit/index.php/ard
ISSN: 2289-7984

Autonomous Robot Car Simulation and Control using Mobile Application

Luqman Hakim Zulkifli[1], Fitri Yakub[1,*]

[1] Department of Electronic Systems Engineering, Malaysia-Japan International Institute of Technology, Universiti Teknologi Malaysia, 54100 Kuala Lumpur, Malaysia

ARTICLE INFO

Article history:
Received 28 February 2020
Received in revised form 25 March 2020
Accepted 13 August 2020
Available online 27 August 2020

Keywords:

Autonomous cars; Robot Operating System (ROS); ROS-Mobile

ABSTRACT

Autonomous cars are becoming more and more relevant as time progresses. Companies are continuously developing better and smarter vehicles that act similarly to human driving behaviour. Many researchers have done projects regarding robot cars, autonomous or not, with different means of controlling them but not many are researching mobile applications as an alternative means to control autonomous vehicles. Mobile apps open to potential versatilities in terms of user interaction and accessibility with their vehicles. This paper proposes to simulate an autonomous robot car in a 3D simulation using the mobile app. In this paper, a 3D robot car was designed using the Robot Operating System (ROS) as the middleware for the robot simulation development, alongside with Gazebo simulator which provides the 3D environment for the simulation. Then, using mapping algorithm to train the robot car to map the simulation environment before testing the robot car's autonomous driving capabilities. The simulation was then tested through a phone application called ROS-Mobile by giving control commands and applying visual angles of the robot car from the mobile app. The mobile app was able to connect to the simulation *via* connecting IP addresses and users can control and visualize the robot car simulation from the mobile app.

1. Introduction

An ideal vision of what seems to be the future of modern transportation, autonomous vehicles have been the topic of intensive research by universities, car companies and research centres since the middle of the 1980s. Over the last decade, three competitions were organized by the Defence Advanced Research Projects Agency aimed to give rise to the technology of autonomous driving [1].

Nowadays, autonomous vehicles have become pertinent in a lot of real-life applications with uses ranging from domestic uses to military and industrial levels. Companies such as Tesla, Renault and many more have already released model vehicles that have an autopilot system where the vehicle enters full autonomous driving mode. As such companies continue to research and develop better and conventional approaches to improve autonomous driving technology, the world will slowly but surely transition to this new modern trend [4-8].

* *Corresponding author.*
E-mail address: mfitri.kl@utm.my

Smartphones on the other hand, are one of the most advanced pieces of technology to which almost everybody in the world has access. Today, many applications for smartphones have been developed, one of them is being able to act as remote controls for devices through integrated communications protocols, such as Wi-Fi or Bluetooth.

Present in an era where data can be stored and connected with one another, possibilities of technology integration are endless and open to various innovative applications. With the assistance of experimental simulations such as the 3D simulator GAZEBO, new applications can be tested with different environmental scenarios prior to real experimental testing to gain supple data and reduce risks and costs.

What makes autonomous vehicles apparent today is their ability to avoid obstacles from their current environment and they rely on their built-in sensors such as radars and cameras to detect obstacles and make decisions to navigate the vehicle accordingly. However, the technology is not perfect and often experiences setbacks. In May 2016, the first accident was caused by a Tesla vehicle during autopilot. Investigations found that the vehicles' sensor system failed to identify a moving object [2]. This caused the system unable to carry out any collision avoidance maneuver e.g. evading the object or braking.

Another tragedy happened in March 2018 when the 38-year-old lost his life after his Tesla car collided with a concrete barrier during autopilot [3]. The US National Transportation Safety Board (NTSB) reported that the driver was distracted by a smartphone video game and was too reliant on the autopilot system. They also reported that the Tesla's obstacle avoidance system was not programmed to sense the collision. This implies the importance of simulations to experiment and replay such scenarios to study and figure out the flaws of the sensor system.

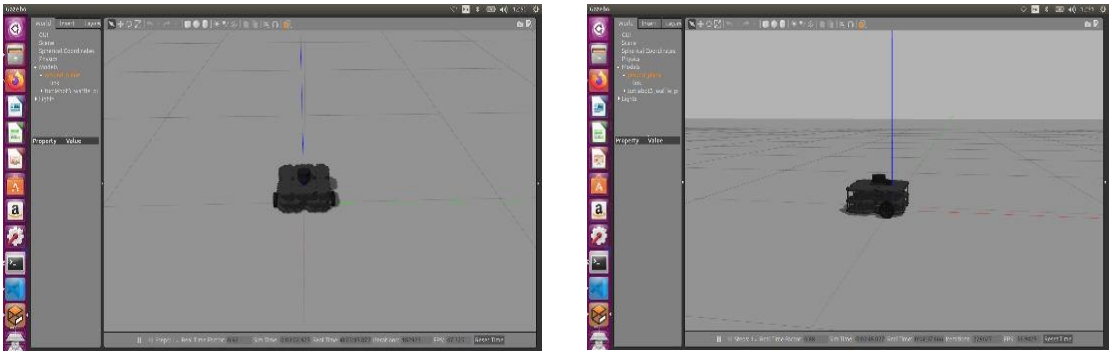
While autonomous cars are being actively researched and developed for the current time being, there are not many mobile applications for remote-controlled autonomous vehicles presently. Since previous existing studies focus on controlling robot simulations through the simulator software itself or a web browser application, this paper proposes a mobile application as a means to control and navigate an autonomous robot car in a simulation. The objectives of this research are to design a 3D robot car simulation and to control and navigate the 3D robot car simulation through a mobile application.

2. Methodology

2.1 Simulation Design

2.1.1 Robot car model

The mapping and navigation algorithm chosen to be applied to the robot car model (RCM) for this project followed the same algorithm as the TurtleBot simulation tutorial [9]. Due to not wanting to tamper with the default navigation and mapping stack parameter values set by the TurtleBot simulation tutorial, the RCM was designed with close reference to the TurtleBot3 Waffle Pi model such as wheel length, radius and sensor dimensions. Figure 1 (a) and (b) shows the TurtleBot3 Waffle Pi model and Table 1 shows its dimensions.



(a) (b)

Fig. 1. TurtleBot3 Waffle Pi model from (a) Top angle view and (b) Side angle view

Table 1
TurtleBot3 Waffle Pi model dimensions

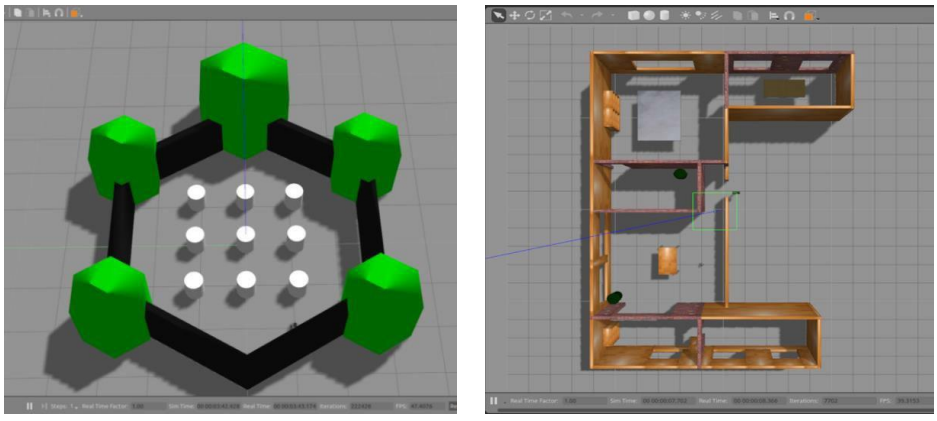
Component	Dimensions
Main Body	Length=26.6cm Width=26.6cm Height=9.4cm
Wheels	Length=1.8cm Radius=3.3cm
Top sensor (radar)	Length=3.15cm Radius=5.5cm
Front sensor (camera	Length=1.5cm Width=3cm Height=2.7cm

2.1.2 Simulation environment

Simulation environments act as routes for the RCM to test its mapping and navigation capabilities. Each simulation environment is different and comes with different sets of obstacles.

2.1.2.1 Provided simulation environments

The TurtleBot simulation package already provided two world environments in a launch file [9]. The world environments are TurtleBot3 World and TurtleBot3 House World as shown in Figure 2 (a) and (b) respectively.



(a) (b)

Fig. 2. (a) The TurtleBot3 World and (b) the Turtlebot3 House World [9]

2.1.2.2 Designed simulation environments

Other than the provided simulation environment, Gazebo also has a world editor feature and a building editor feature where users can create and design their world environment by using and manipulating the 3D shapes and objects provided in Gazebo. Two new simulation environments, KSJ-room World and KSJ-corridor World were created. These environments were designed replicas based on Kolej Siswa Jaya (KSJ), Universiti Teknologi Malaysia hostel room and corridors, respectively. These environments were to test the RCM mapping and navigation abilities if done in real-world scenarios. Figures 3 (a) and (b) show photos of the KSJ hostel room and Figures 4 (a) and (b) are photos of the KSJ corridor as a reference for the environment design.

(a) (b)

Fig. 3. Photos of Kolej Siswa Jaya room from (a) Entrance door view and (b) Bathroom door view

(a) (b)

Fig. 4. Photos of Kolej Siswa Jaya corridors from (a) View of the elevator at the center of the floor and (b) View of room corridor

2.2 Tele-operation and Mapping

The main step to train the robot to drive around its environment autonomously was to introduce the robot to every space or course it potentially needs to manuever to. This can be done by controlling the RCM *via* tele-operation to map the entire environment course creating a 2D map, from sensor and location data collected by the RCM. This process is called Gmapping. The Gmapping package provides laser-based Simultaneous Localization and Mapping (SLAM) as a ROS node named slam_gmapping node. Widely used in robotics, SLAM's algorithm generates a map of the environment the RCM picked up during mapping and simultaneously locates its position using applied sensors. This purpose of SLAM is the main reason why it is a great mapping method in the field of autonomous driving [10]. Figure 5 shows all SLAM classifications based on the algorithm and target generation used.

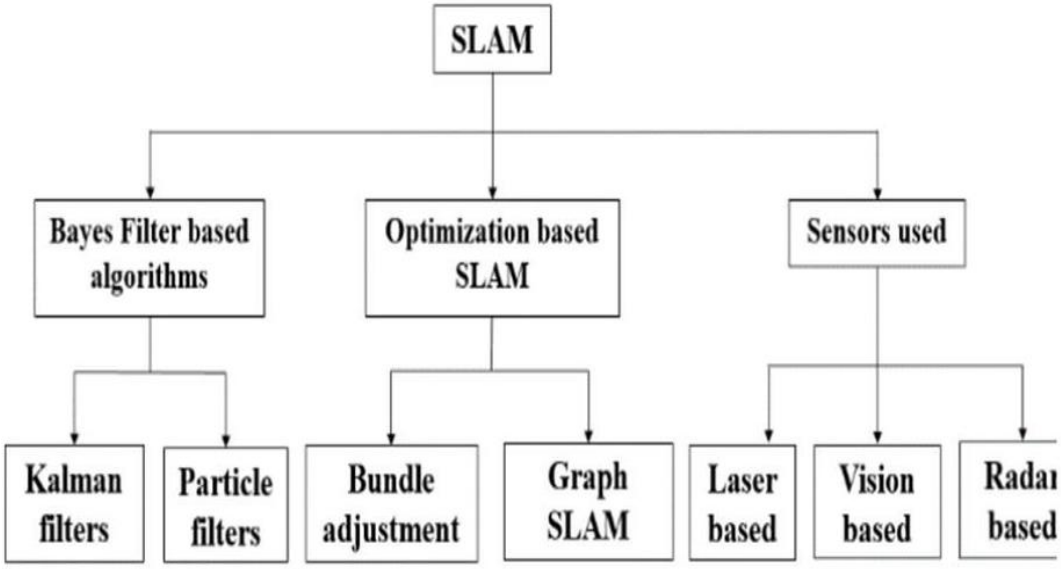


Fig. 5. Simultaneous localization and mapping (SLAM) classifications [10]

The SLAM algorithm method used for this project is the Extended Kalman Filter (EKF) under the Kalman Filters (KF) class based on Bayes Filter. The KF consists of two parts: the prediction stage and the updating stage. The prediction stage uses past values and iterations to guess the current state's state. The update phase uses this guessed state and merges its current data collected from the sensor nodes, known as the posterior [10]. The EKF was developed on top of the KF to generally eliminate the Non-Linearity problem of the pose model [11]. In the EKF method, a first-order Taylor expansion was used to estimate the current. In most cases, the guessed states were quite near to ground truth.

2.2.1 Mapping parameters

SLAM_gmapping node comes with a long list of parameters [12] so that users can customize and adjust the mapping parameters suitable for their robot's specifications to produce more accurate or more detailed mapping results. In this project, the SLAM_gmapping parameters focused are listed in Table 2.

Table 2
Focused mapping parameters [12]

Parameter	Description
map_update_interval (Map update)	Time (in seconds) taken for every laser scan updates of the generated map. The less time taken, the more often the map is updated but takes greater loading computation
maxUrange (Laser range)	The maximum range of the laser scan distance
lskip	The number of laser beams to skip or turned off
linearUpdate (Linear update)	The range for the robot to travel for a scan process to occur
angularUpdate (Angular update)	The angle for the robot to turn for a scan process to occur
particles	Number of filter particles during scans
xmin	Minimum map generated size in x-axis
xmax	Maximum map generated size in x-axis
ymin	Minimum map generated size in y-axis
ymax	Maximum map generated size in y-axis

2.3 Autonomous Drive Testing

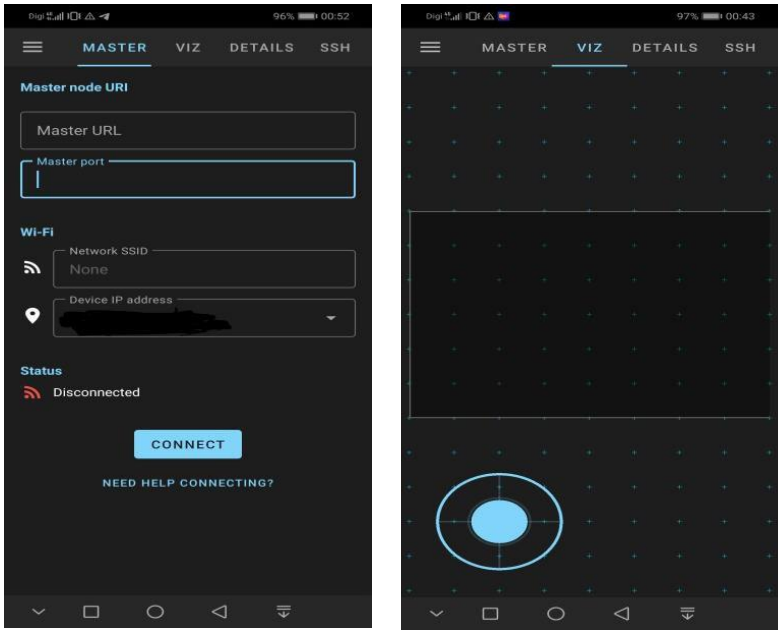
ROS navigation package is a 2D navigation stack that takes in information from the robot's sensors, the robot's odometry, and a goal pose and outputs safe velocity commands that are sent to a mobile base, in this case, the RCM. The goal of this navigation stack was to move the RCM from the initial or starting position to the goal position without colliding with any obstacle within the simulation environment.

The ROS navigation package comes with an implementation of several navigation-related algorithms that aid in implementing autonomous navigation properties in mobile robots. Users only need to give the goal destination of their robot and their robot odometry information from sensors such as wheel encoders, IMU and GPS, alongside other sensor data streams. The navigation package's output will be the velocity and movement commands that will move the robot to the goal destination.

The navigation process for this project used Adaptive Monte Carlo Localization (AMCL). AMCL is a localization system for a robot moving in 2D. This system applies the adaptive Monte Carlo localization approach that uses an adaptive particle filter to trace the ground robot's pose, which changes continuously as the ground robot navigates in the area [13]. Ideally, by applying the created 2D map from the Gmapping step, the AMCL algorithm will try to match the laser scans from the robot car to the map and thus localizing the robot to its current environment based on the map.

2.4 Mobile Application Connection and Simulation Control

The ROS-Mobile app can be downloaded on Android phones for free at the Google Play Store. The user interface consists of four tabs. The application must first establish a connection with the simulation computer's IP address at the "MASTER" tab. Figure 6 (a) shows the ROS-Mobile app at the "MASTER" tab where the phone is currently not connected to any devices. After establishing a connection, users can assign widgets at the "DETAILS" tab such as a joystick widget to remote control the robot car simulation, a camera widget to view the robot from the camera plugin angle, and a visualizer widget to visualize the mapping of the robot during the mapping process. Finally, users can use and visualize their selected widgets at the "VIZ" tab. Figure 6 (b) shows the "VIZ" tab with already assigned camera and joystick widgets but since the application is not connected to the simulation computer IP address, no images were shown at the camera widget.



(a) (b)

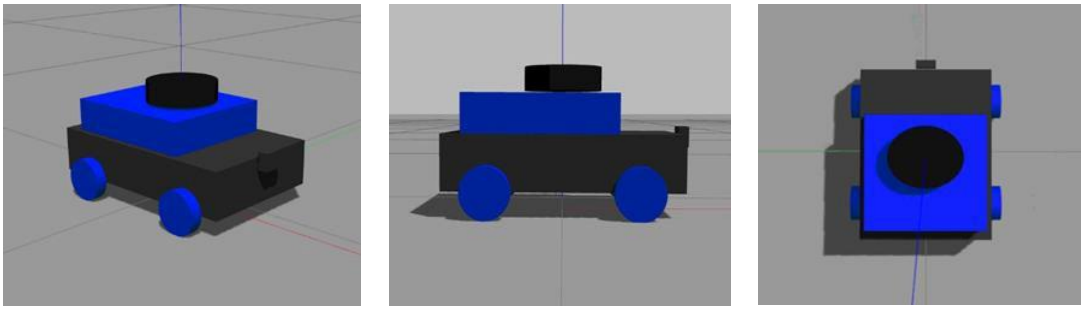
Fig. 6. ROS-Mobile application at (a) MASTER tab and (b) VIZ tab

3. Results and Discussion

3.1 Robot Car Simulation Design

3.1.1 Robot car model

Since the mapping stack and navigation stack algorithm used in this project was designed for the Turtlebot3 model, most of the RCM's design was influenced by the dimensions of the Turtlebot3 model such as the wheel length and diameter and the top sensor (radar)'s height from the ground. Figure 7 (a), (b) and (c) shows the 45-degree angle view, the side view and the top view of the finished RCM designed, respectively. The specifications and dimensions of the designed 3D RCM for this project are described in Table 3.



(a) (b) (c)

Fig. 7. The finished RCM design from (a) Angle view, (b) Side view and (c) Top view

Table 3
Dimensions of the 3D RCM

Component	Dimensions
Main body	Length=32cm
	Width=20cm
	Height=7cm
Top chassis	Length=21cm
	Width=18cm
	Height=5cm
Wheels	Length=1.8cm
	Radius=3.3cm
Top sensor (Radar)	Length=3.15cm
	Radius=5.5cm
Front sensor (Camera)	Length=1.5cm
	Width=3cm
	Height=2.7cm

3.1.2 Simulation environment

Four different simulation environments were used in this project for testing the mapping and navigation of RCM. Two of them were provided by the tutorial website [9] which were Turtlebot3 World and Turtlebot3 House World. The remaining two simulation environments were created and designed new environments based on the Kolej Siswa Jaya (KSJ) Hostel which are KSJ-room and KSJ-corridor world.

3.1.2.1 Provided simulation environments

Turtlebot3 World and Turtlebot3 House World are simulation environments provided along with the tutorial package of Turtlebot3 simulation at Robotis Emanual website [9]. Figure 8 shows the Turtlebot3 World environment where the obstacles are only simple pillars enclosed in a hexagon wall. Figure 9 shows the Turtlebot3 House World environment. Figure 10 shows a detailed view of the environment with labelled furniture and rooms. The Turtlebot3 House World environment furniture consists of tables, cupboards, trash cans and has 6 rooms namely Room 1, Room 2, Room 3, Room 4, Room 5 and Room 6. The approximate dimensions of each room in the TurtleBot3 House World environment are shown in Table 4.

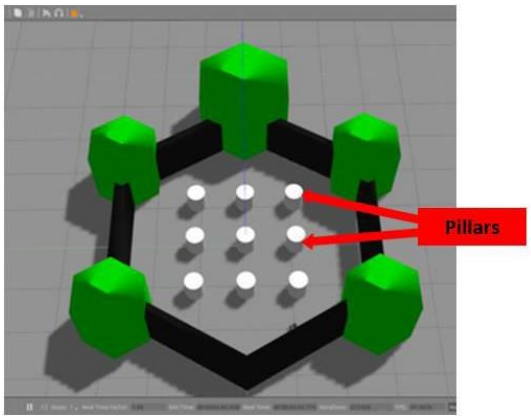


Fig. 8. Turtlebot3 World

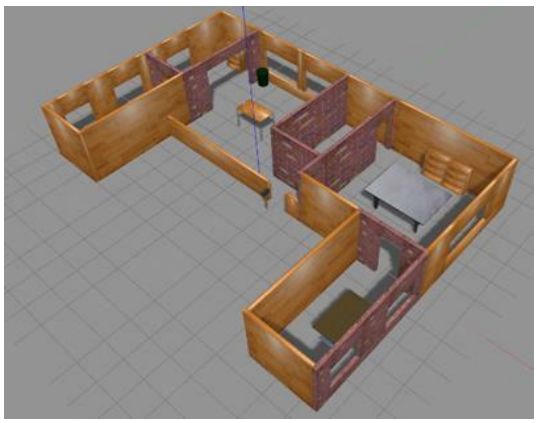


Fig. 9. Perspective view of Turtlebot3 House World

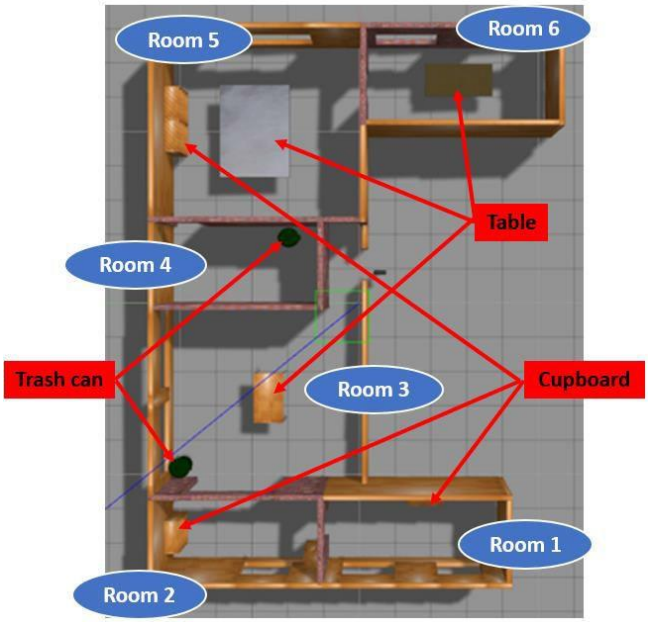


Fig. 10. Detailed top view of Turtlebot3 House World

Table 4
TurtleBot3 House World environment rooms' approximate dimensions

Room No.	Dimensions
1	Length=5m Width=2.5m
2	Length=4.5m Width=2.5m
3	Length=5.5m Width=5m
4	Length=4.5m Width=2.5m
5	Length=5.5m Width=5m
6	Length=5m Width=3m

3.1.2.2 Designed simulation environments

Two simulation environments, KSJ-room World and KSJ-corridor World were designed replicas of KSJ Hostel, Universiti Teknologi Malaysia room and corridors, respectively. The measurements of each exterior and obstacle were measured manually using measuring tape. Figure 11 shows the detailed top view of KSJ-room World. It consists of two rooms, namely Room 1 and Room 2, connected by a shared bathroom as well as furniture like lockers (black), study table (green) and bed (red). Figure 12 is the comparison between the KSJ-room World simulation and real-life photos of the KSJ room hostel.

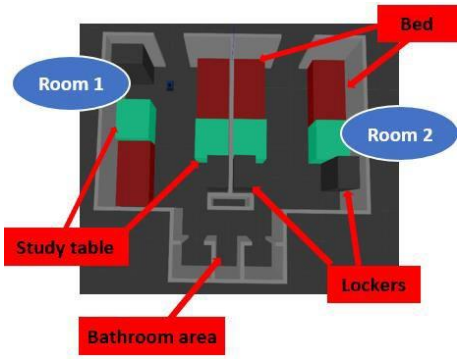


Fig. 11. Detailed top view of KSJ-room World

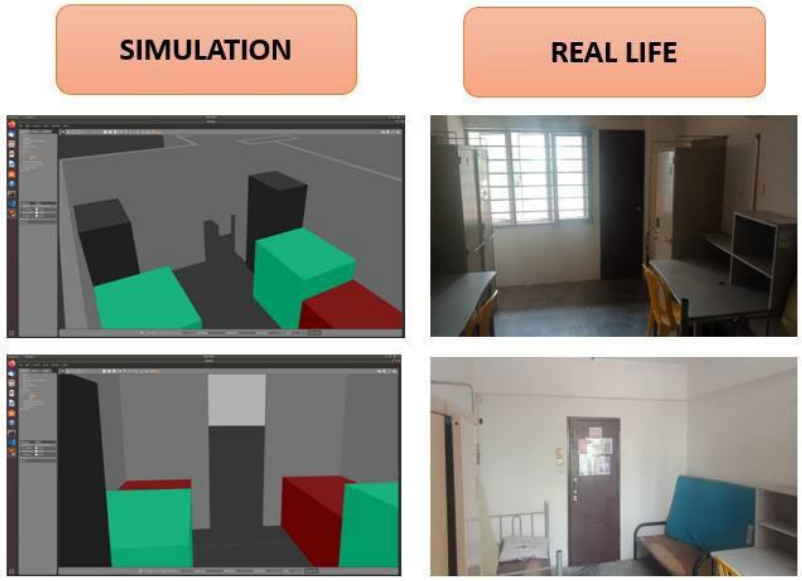


Fig. 12. Comparison of KSJ room simulation and real life

Figure 13 (a) and (b) show the detailed top view of KSJ-corridor World. It has four main corridors, namely Corridor 1, Corridor 2, Corridor 3 and Corridor 4, connected to the middle elevator area along with objects such as benches (red), washing machine (green), water dispenser (blue) and trash cans (grey). Figure 14 is the comparison between the KSJ-corridor World simulation and real-life photos of the KSJ hostel corridor. The approximate length of each corridor in the KSJ-corridor World environment is shown in Table 5.

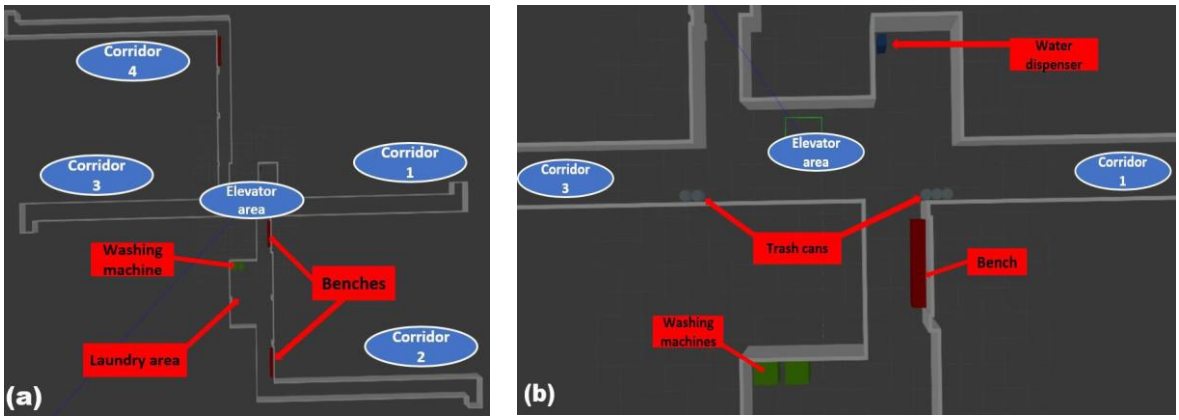


Fig. 13. Detailed view of (a) KSJ-corridor World and (b) Elevator area centre

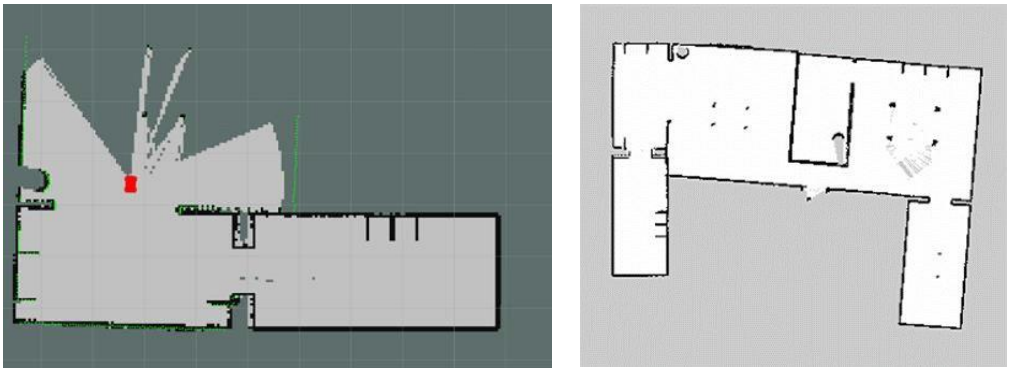
Table 5
KSJ-corridor World environment corridors' approximate lengths

Corridor no.	Length (meters)
1	38.00
2	38.40
3	38.60
4	39.40

Fig. 14. Comparison of KSJ corridor simulation and real-life photos

3.2 RCM Training and Mapping

Figure 15 (a) shows the RCM manually controlled *via* tele-operation to map the entire TurtleBot3 House World environment creating a 2D map of it shown in Figure 15 (b). The generated 2D map, although not precisely aligned, follows the same dimensions as the original footprint of the rooms of the TurtleBot3 House World environment.



(a) (b)

Fig. 15. (a) The robot car (in red) mapping the House environment resulting in (b) Generated 2D occupancy grid map of the House World

The mapping process was executed on all four simulation environments to experiment with the algorithm's mapping accuracy. Figures 16 (a), (b) and (c) show the Turtlebot3 World, Turtlebot3 House World and KSJ-room World simulation environments respectively with their resulting 2D occupancy map generated by SLAM_gmapping node when controlling the RCM to map the entire simulation environment. It was found that the 2D map generated from the mapping of these three simulation environments was quite accurate with little to no error in terms of the environment's dimension lengths.

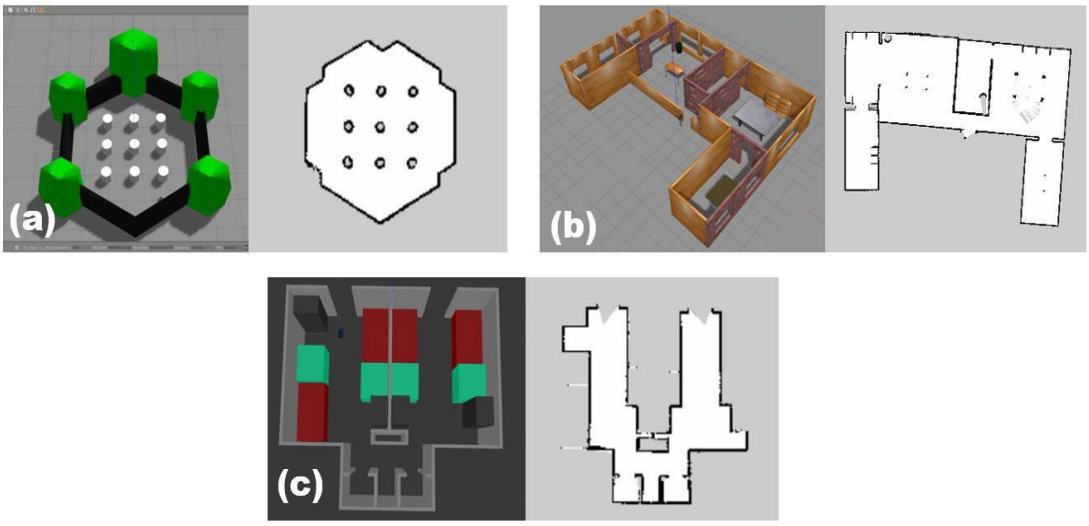


Fig. 16. Simulation environments with the resulting 2D occupancy map of (a) Turtlebot3 World, (b) Turtlebot3 House World and (c) KSJ-room World

However, mapping for KSJ-corridor World created inaccurate 2D-map dimensions of the environment. Figure 17 shows Corridor 1 of the KSJ-corridor World simulation environment and the resulting 2D-map.

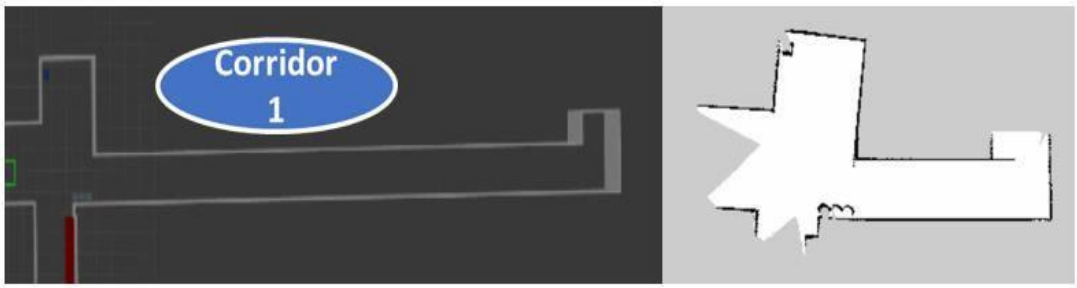


Fig. 17. Corridor 1 of KSJ-corridor World environment with its resulted 2D occupancy map

Table 6 shows the comparison between the length of Corridor 1 in the simulation environment and the length of its generated 2D map. Based on Table 6, the length of the 2D-map generated from the mapping of the KSJ-corridor World environment was far from accurate. Even from the first glance of Figure 17, it can be assumed that the 2D-map generated was much shorter than the simulation environment. This can lead to conflict later in the autonomous driving test when the inaccurate 2D map is applied and users place destination goals based on the 2D map and the RCM will still drive autonomously but will not arrive at the desired goal in the simulation environment.

This inaccuracy of mapping along the simulation corridor is known as the "Corridor Problem" where the mapping algorithm thinks that the RCM is not moving when its laser sensor data scans are always the same in long corridors [14]. Two different approaches were tested to solve this "Corridor Problem".

Table 6
Comparison lengths of Corridor 1 environment and resulted in 2D-map

Corridor 1	Length (meters)
KSJ-corridor environment	38.00
2D occupancy map	5.00

3.2.1 "Corridor problem" approach 1: Adjusting mapping parameters

The first approach to solve the "Corridor Problem is adjusting the RCM's mapping algorithm's mapping parameters. Table 7 shows the trial and error of focused mapping parameters mentioned in Table 2 with different value combinations to get the distance mapped of the corridor as close as possible to the actual simulation distance of Corridor 1 KSJ-corridor World environment. The mapping accuracy was calculated using the accuracy percentage formula,

$$\text{Mapping accuracy} = \frac{Distance\ mapped}{Corridor\ 1\ actual\ distance} x\ 100\% \times \quad (1)$$

Table 7
Trial and error of different combination values of mapping parameters

	Parameters												
Trial	Laser range	Linear update	Angular update	Map update	Particles	xmin	xmax	ymin	ymax	lskip	Distance mapped	Corridor 1 actual distance	Mapping accuracy (%)
1	3	1	0.2	2	100	-10	10	-10	10	0	6	38	15.79
2	80	1	0.5	5	30	-100	100	-100	100	0	6	38	15.79
3	80	1	0.2	1	100	-10	10	-10	10	0	8	38	21.05
4	80	1	0.2	2	100	-100	100	-100	100	0	7	38	18.42
5	80	0.1	0.2	2	30	-100	100	-100	100	0	7	38	18.42
6	80	0.1	0.2	2	100	-10	10	-10	10	10	14~15	38	36.84~ 39.47
7	3	1	0.2	2	100	-100	100	-100	100	10	13-14	38	34.21~ 36.84
8	80	1	0.2	2	100	-100	100	-100	100	5	9-10	38	23.68~ 26.31
9	80	1	0.2	2	100	-100	100	-100	100	0	7	38	18.42

Based on Table 7, the difference in distance between the distance mapped during environment mapping and the actual distance improved at trials 6 and 7. In those trials, the parameter "lskip" which was the number of laser beams to be skipped or turned off is increased, it allows less computation for laser work for the mapping algorithm. This is ideal since the resulting map was only a 2D map and does not need very detailed mapping. This allowed prioritization on the mapping distance aspect of the mapping algorithm. Despite the improved precision of the distance mapped, the result was still inaccurate, and the results were not consistent with multiple trials.

3.2.2 "Corridor problem" approach 2: Adding obstacles in the simulation environment

The second approach was adding obstacles along Corridor 1 of the KSJ-corridor World environment so that the mapping algorithm of RCM can sense some changes when mapping the corridor and assume the RCM was certainly moving. Figure 18 shows the comparison change between the original Corridor 1 design to the new version. The new version was modified by adding trash cans (blue) along the corridor of Corridor 1 KSJ-corridor World environment. Figure 19 shows the modified Corridor 1 and its resulting 2D-map of the modified environment.

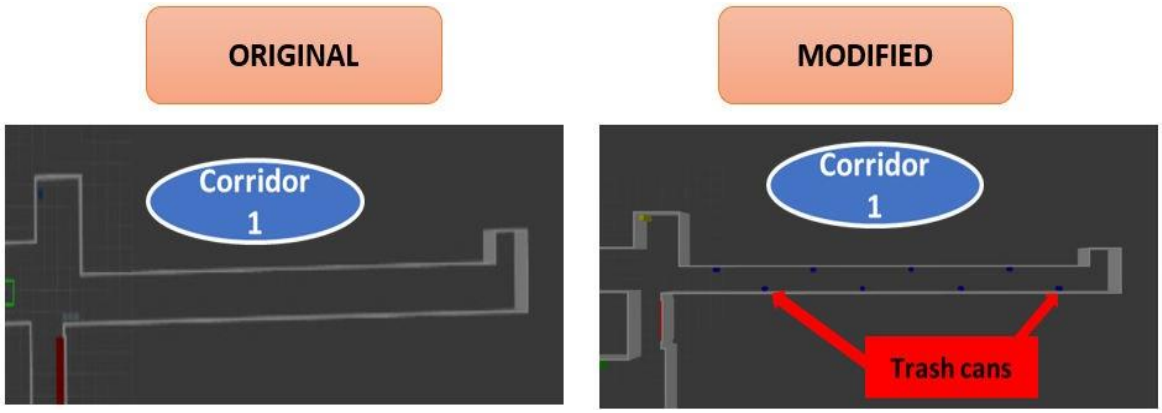


Fig. 18. Comparison changes between the original Corridor 1 design to the new modified version

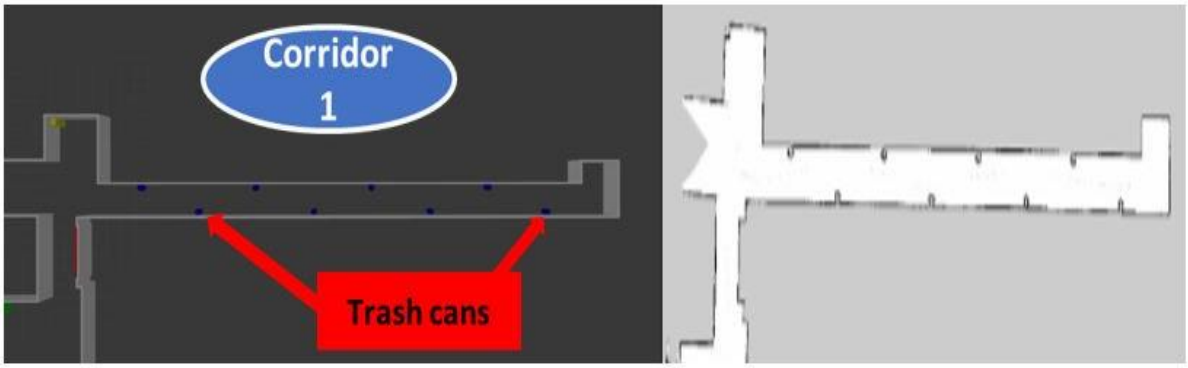


Fig. 19. Modified Corridor 1 of KSJ-corridor World environment with its resulted 2D occupancy map

Table 8 shows the multiple trials of mapping of the changed Corridor 1 KSJ-corridor World to compare the distance mapped and mapping accuracy of the environment with the actual environment length.

Table 8
Mapping accuracy trials of new KSJ-corridor World environment

Trial	Distance mapped	Corridor 1 distance	Mapping accuracy (%)
1	38.00	38.00	100
2	37.50	38.00	98.68
3	38.00	38.00	100
4	38.00	38.00	100
5	37.00	38.00	97.37
6	37.50	38.00	98.68

Based on Table 8, the mapping results of the Corridor 1 KSJ-corridor World after adding obstacles along the corridor give little to no error in distance difference. Hence, this approach was used as the solution to the "Corridor Problem" due to better mapping accuracy. The simulation environment of KSJ-corridor World was changed by adding trash cans (blue) along each corridor. Figure 20 shows the comparison between the previous and updated KSJ-corridor World environment.

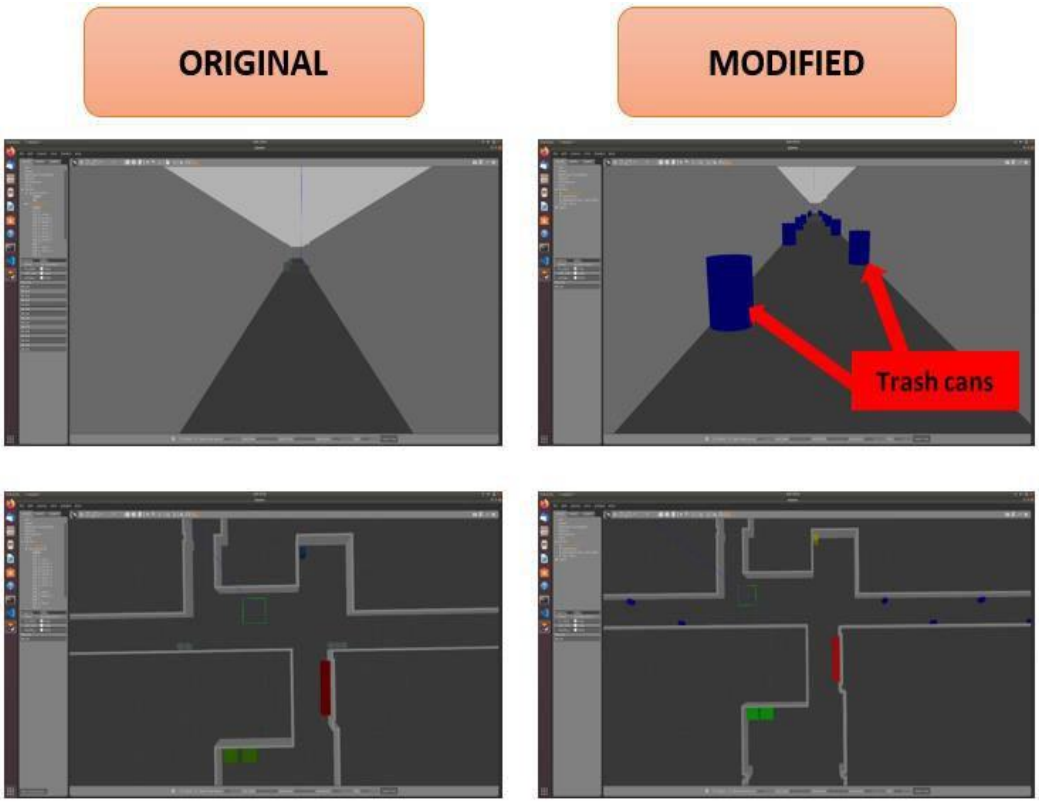


Fig. 20. Comparison between the original and modified KSJ-corridor World environment

3.3 Autonomous Driving Test

Figure 21 shows the initial position of the RCM after the autonomous driving simulation program launch. The initial position of the RCM was in Room 1. The blue-coloured area represents the robots' local map planner that gave velocity commands to send to the robot car corresponding with the current nearest environment around it. The green lines were true obstacles picked up by the robot's laser sensor. In this case, the green lines match closely with the created 2D map.

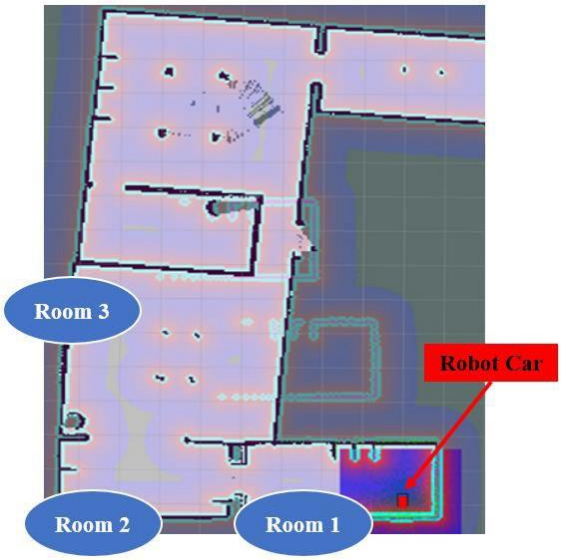
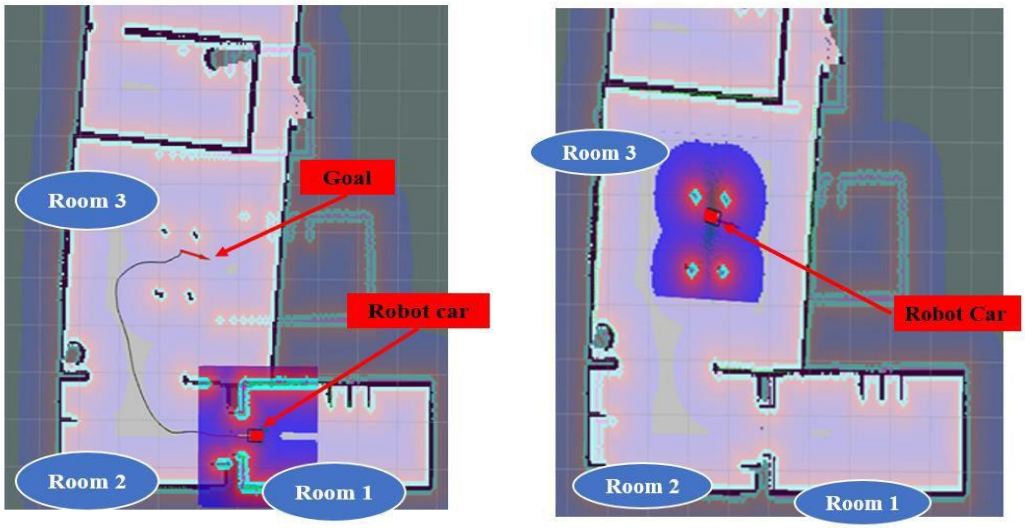


Fig. 21. Initial position of robot car before autonomous driving

Then, a destination goal was set as shown in Figure 22 (a) which was supposed to be right under the table in Room 3. The navigation stack then generated a course for the robot car to follow towards the goal, which was to drive from Room 1, passing Room 2 and entering Room 3 until the robot car reaches the goal such as in Figure 22 (b), avoiding obstacles along the way.



(a) (b)

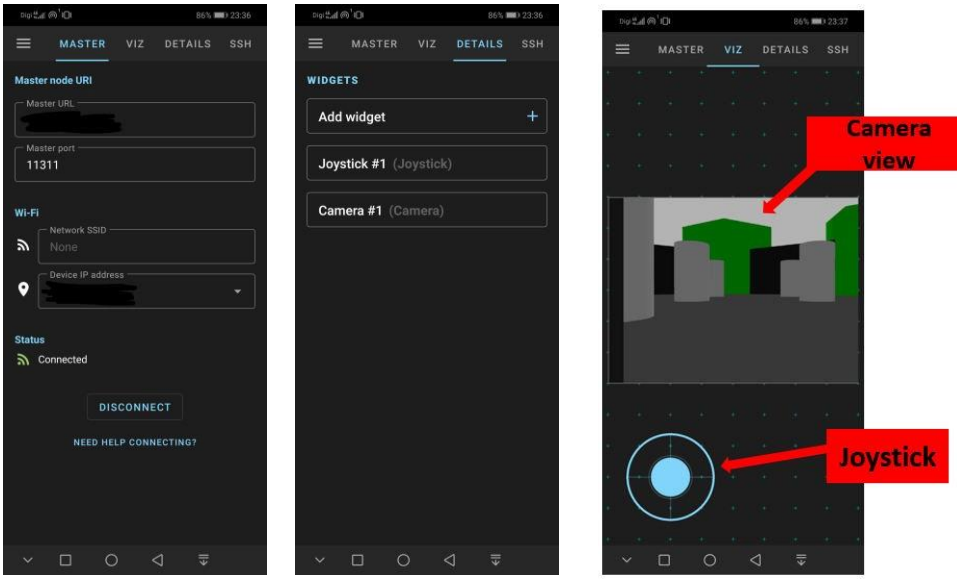
Fig. 22. (a) The simulation of the robot car driving autonomously towards the goal and (b) The robot car reaching the goal

3.4 Mobile App Connection and Simulation Control

The final step was testing the RCM simulation on a mobile application, ROS-Mobile app. The first was establishing a connection with the simulation computer's IP address at the "MASTER" tab. A stable internet connection was needed and both computer and mobile phone need to be in the same network for the connection to successfully establish. Figure 23 (a) shows the ROS-Mobile app at the "MASTER" tab where the phone was currently connected to the RCM simulation's computer's IP address.

Next, assign widgets at the "DETAILS" tab to aid the user interface with the simulation from the mobile app. Figure 23 (b) shows two selected widgets at the "DETAILS" tab which are a joystick widget to control the RCM and a camera widget to view the simulation environment from the RCM camera plugin view angle.

Finally, visualizing the simulation at the "VIZ" tab. Figure 23 (c) shows the "VIZ" tab with already assigned camera and joystick widgets. The RCM was controlled *via* the joystick widget. The camera widget shows the simulation environment the RCM is currently in which is the Turtlebot3 World environment. The demonstration of the robot car simulation and ROS-MOBILE control can be seen in the video link [15].



(a) (b) (c)

Fig. 23. ROS-Mobile application running the simulation control at (a) MASTER tab, (b) DETAILS tab and (c) VIZ tab

4. Conclusion

This research was carried out using the 3D simulator GAZEBO as not only the main environment setter but also as the main editing tool for RCM and simulation environment designing. Using the mentioned mapping stack packages and SLAM, the RCM was then trained to map four simulation environments namely Turtlebot3 World, Turtlebot3_House World, KSJ-room World and KSJ-corridor World, and simultaneously generating 2D maps of each mapped environment. From the mapping of KSJ-corridor World, it was found that the mapping algorithm produced inaccurate results which was the inaccurate mapping of corridor path length. Two approaches were taken to improve the RCM's mapping capability which were to adjust its mapping parameters and modifying the environment by adding obstacles around the inaccurate path. The latter approach improved the RCM mapping effectively and was used as a solution to the inaccurate mapping problem. Next, the RCM's autonomous driving capabilities were simulated by setting a location goal and running the navigation stack. Rviz software was applied as a visualization tool to display the data captured from the RCM's sensors. Finally, the ROS-Mobile application was applied to connect to the simulation's computer's IP address to control and navigate the RCM simulation through the mobile application.

References

[1] Badue, Claudine, Rânik Guidolini, Raphael Vivacqua Carneiro, Pedro Azevedo, Vinicius B. Cardoso, Avelino Forechi, Luan Jesus et al. "Self-driving cars: A survey." *Expert Systems with Applications* 165 (2021): 113816. https://doi.org/10.1016/j.eswa.2020.113816

[2] United States National Transportation Safety Board. Highway Preliminary Report: HWY19FH008, https://www.ntsb.gov/investigations/accidentreports/pages/hwy19fh008 preliminary-report.aspx

[3] "Tesla Autopilot crash driver 'was playing video game'", BBC News, 2021. [Online]. Available: https://www.bbc.com/news/technology-51645566

[4] Padmaja, B., PV Narasimha Rao, M. Madhu Bala, and E. Krishna Rao Patro. "A novel design of autonomous cars using IoT and visual features." In *2018 2nd International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC) I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), 2018 2nd International Conference on*, pp. 18-21. IEEE, 2018. https://doi.org/10.1109/I-SMAC.2018.8653736

[5] Olson, Elizabeth A., Nathalie Risso, Adam M. Johnson, and Jonathan Sprinkle. "Fuzzy control of an autonomous car using a smart phone." In *2017 CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON)*, pp. 1-6. IEEE, 2017. https://doi.org/10.1109/CHILECON.2017.8229692
[6] S. Rao, "How to Build a Data Pipeline for Autonomous Driving | NetApp Blog", NetApp Blog, 2019. [Online]. Available: https://blog.netapp.com/how-to- build-a-data-pipeline-for-autonomous-driving/
[7] Lange, Stefan, Fritz Ulbrich, and Daniel Goehring. "Online vehicle detection using deep neural networks and lidar based preselected image patches." In *2016 IEEE Intelligent Vehicles Symposium (IV)*, pp. 954-959. IEEE, 2016. https://doi.org/10.1109/IVS.2016.7535503
[8] Hou, Yew Cheong, Khairul Salleh Mohamed Sahari, Leong Yeng Weng, Hong Kah Foo, Nur Aira Abd Rahman, Nurul Anis Atikah, and Raad Z. Homod. "Development of collision avoidance system for multiple autonomous mobile robots." *International Journal of Advanced Robotic Systems* 17, no. 4 (2020): 1729881420923967. https://doi.org/10.1177/1729881420923967
[9] "ROBOTIS e-Manual", ROBOTIS e-Manual. [Online]. Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/simulation/#gazebo- simulation
[10] Sankalprajan, P., Thrilochan Sharma, Hamsa Datta Perur, and Prithvi Sekhar Pagala. "Comparative analysis of ROS based 2D and 3D SLAM algorithms for Autonomous Ground Vehicles." In *2020 International Conference for Emerging Technology (INCET)*, pp. 1-6. IEEE, 2020. https://doi.org/10.1109/INCET49848.2020.9154101
[11] Chong, T. J., X. J. Tang, C. H. Leng, Mohan Yogeswaran, O. E. Ng, and Y. Z. Chong. "Sensor technologies and simultaneous localization and mapping (SLAM)." *Procedia Computer Science* 76 (2015): 174-179. https://doi.org/10.1016/j.procs.2015.12.336
[12] "gmapping-ROS Wiki", Wiki.ros.org. [Online]Available: http://wiki.ros.org/gmapping
[13] Chatziparaschis, Dimitrios, Michail G. Lagoudakis, and Panagiotis Partsinevelos. "Aerial and ground robot collaboration for autonomous mapping in search and rescue missions." *Drones* 4, no. 4 (2020): 79. https://doi.org/10.3390/drones4040079
[14] "Problems in hector slam -ROS Answers: Open Source Q&A Forum", Answers.ros.org, 2021. [Online]. Available: https://answers.ros.org/question/235304/problems-in hector-slam/
[15] L.H. Zulkifli, Final Year Project (FYP) Video Presentation, 2021. [Video file]. Available: https://www.youtube.com/watch?v=zYkUGm3ZfZI&t=2s