



## Research Article

# Real-Time SLAM Mobile Robot and Navigation Based on Cloud-Based Implementation

Jaafar Ahmed Abdulsahab <sup>1,2</sup> and Dheyaa Jasim Kadhim <sup>1</sup>

<sup>1</sup>Department of Electrical Engineering, College of Engineering, University of Baghdad, Baghdad, Iraq

<sup>2</sup>Department of Electronics and Communication, College of Engineering, Uruk University, Baghdad, Iraq

Correspondence should be addressed to Jaafar Ahmed Abdulsahab; [jaafer@uruk.edu.iq](mailto:jaafer@uruk.edu.iq)

Received 3 January 2023; Revised 10 March 2023; Accepted 15 March 2023; Published 29 March 2023

Academic Editor: L. Fortuna

Copyright © 2023 Jaafar Ahmed Abdulsahab and Dheyaa Jasim Kadhim. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

This study investigates the feasibility of a mobile robot navigating and discovering its location in unknown environments, followed by the creation of maps of these navigated environments for future use. First, a real mobile robot named TurtleBot3 Burger was used to achieve the simultaneous localization and mapping (SLAM) technique for a complex environment with 12 obstacles of different sizes based on the Rviz library, which is built on the robot operating system (ROS) booted in Linux. It is possible to control the robot and perform this process remotely by using an Amazon Elastic Compute Cloud (Amazon EC2) instance service. Then, the map to the Amazon Simple Storage Service (Amazon S3) cloud was uploaded. This provides a database to display maps and use them at any time for navigation without the need to redraw the map. This map can be accessed by using an authentication process (username and password) supervised by the cloud server administrator. After that, using the serverless image handler (SIH), with the aid of this solution, you can change the size of images, change the color of the background, format them, or add watermarks. Experiment results demonstrated the ability to build a map of an unknown location in a complex environment and use it for navigation tasks on a real mobile robot via remote control. It also showed the success of the process of storing the map for future use and the process of modifying the map using SIH.

## 1. Introduction

Due to the growing need for robots in our lives, the robotics industry has seen many important changes, such as automated production, extraterrestrial missions, unmanned rescue vehicles, disaster rescue operations, socially supported robots at home and in education, self-driving vehicles, and therapeutic robots used in health, Mars survey, or seabed analysis [1]. One of these important developments in the field of robots is the use of networked robots to share information and provide required resources such as data storage, complex computation, and so on. These networked robots can get these information and resources through servers that are connected to the Internet and programs that they have installed [2]. To address the proposed difficulties that the robots may face in accessing the information and required resources, developers and researchers have recently

proposed the creation of a cloud-connected robotics network that uses flexible resources when requested from the global cloud architecture [3]. Cloud computing philosophies have been applied by researchers in robotics as well as mobile environments, resulting in topics such as cloud robots and mobile clouds. When connected to the cloud, these robots can take advantage of the powerful computing and communications resources in the cloud for the data center and storage and can also process and share information with many robots [4]. Cloud Computing (CC) can be used to upgrade the capabilities of robots in many tasks since cloud computing features are increasingly useful for mobile robots. In general, there are two types of mobile robots: autonomous mobile robots, which can explore their environments on their own, whether they are known or unknown, and nonautonomous mobile robots, which may use a guidance system to move through their surroundings

[5]. This work deals with autonomous mobile robots (AMRs), which are needed in many fields to operate independently of humans. Planning a better path, while avoiding obstacles, is essential for AMR to move from one point to another, representing the target in their environment. There are several implementations that deal with the field of robotics, such as SLAM, which is abbreviated for simultaneous localization and mapping, as well as the large number of sensing and grasping systems that quickly lead towards the emergence of large amounts of sensor data. These data are difficult to store because most robots have limited storage capacity. Cloud robotics (CR) submits solutions towards saving space within the cloud so that all-important robot information can be stored inside this cloud for future use. This feature allows the CR to authorize mobile robotics to access huge datasets such as general localization maps and platforms of the environment [6].

Cloud robotics has become increasingly popular in recent years. Many works have a lot to do with the ways to do research that are talked about in this section. The study by Afanasyev et al. [7] presented the SLAM algorithm based on ROS for the simulated mobile robot on the Gazebo simulator, which is moveable in a three-dimensional pattern of the actual internal situation that is introduced and is offered in a 3D model form based on the actual interior environment. Then, the simulation results of the willow garage personal robot (PR2) robot are described. The camera shots taken with autodesk's Rao-Blackwellized Particle Filter (RBPF) and the data from the PR2 robot's laser for localization and mapping are used to create an image-based 3D model of the actual area. In the study by Doriya et al. [8], different cloud architectures and how they connect to one another or differ from one another in their ability to solve SLAM problems are discussed and compared. So, every framework uses various collections of special protocols within the cloud that enable the mobile robot to delegate its computational tasks to the cloud. The researchers confirmed that the framework of Rapyuta is more elastic than other frameworks, and because its implementation is open source, it can be expanded to include functions of other robots by several other researchers. Whereas, when they compared it to DaVinci, they found that DaVinci was the most suitable for SLAM. Finally, the cloud framework for cooperative tracking and mapping (C2TAM) framework is suitable for saving huge amounts of data.

Takaya et al. [9] introduced a simulation environment for mobile robots based on ROS and Gazebo. The code advanced for the simulation procedure can be directly executed on the actual robot without modifications after properly creating the models of the robot under the gazebo simulator. Then, the tasks of autonomous navigation and 3D-mapping simulation utilizing control programs under ROS are offered. The results of the simulation and experiments coincided very well and showed the usability of the advanced environment. For this study, the ROS platform was used to build models of the PeopleBot and Pioneer 3-DX robots. Limosani et al. [10] proposed a system based on the cloud robotics model that lets automated mobile robots move through and observe previously unknown interior

environments that are divided into subdivided maps. Also, the whole topological demonstration and necessary information of the world are saved in cloud structures from a distance by employing fixed ecological tags that consist of collections of QR codes as well as AR tags. Tests are being done to determine the possibility of selection, adjust the sizes, and see if the proposed procedure can be done. Megalingam et al. [11] presented the ROS implementation of SLAM by calculating how long the robot model would take to arrive. The test is conducted in an Rviz-created virtual environment. The travel time is calculated by placing various dynamic obstacles for various destinations on the map. It has been seen that the robot has a good response time and travels from the source to the destination in a reasonable amount of time, where the amount of time needed increases as the distance does. Jiang et al. [12] presented SLAM, which was put into practice by using Turtlebot2, a mobile robot, and the robot operating system's navigational package. The Turtlebot2 is outfitted with a notebook, a 3D-LiDAR ranging sensor, an RGB-D camera, and light detection. These features are used to help the robot navigate and create a full 3D sight map in addition to a novel 2.5-dimensional map. The conclusion of the collected results displays a collection of image and scans data able to enhance the performance of mapping using classical approaches. The study by Arena et al. [13] describes an innovative robot that looks like a worm and is made of an ionic polymer-metal composite (IPMC) self-actuated skeleton. It is controlled by cellular neural networks (CNNs). Worm locomotion is performed by bending the actuators sequentially from "tail" to "head," imitating the traveling wave observed in real-world undulatory locomotion. The study by Buscarino et al. [14] looks into and suggests a way to control a group of mobile robots without a central point of control using a dynamical network model. The results show that the performance of the system can be improved by letting the robots talk to each other over long distances, and a good way to control exploration and transport is presented. Small-world networks can be highly clustered, like regular lattices, yet have small characteristic path lengths, making them "small-world" dynamical systems with enhanced signal-propagation speed, computational power, and synchronizability [15]. The study by Kajita and Espiau [16] talks about the history of research on two-legged robots, the dynamics and control of general two-legged robots, the zero-moment point (ZMP), the relationship between gaits and stability, and how two-legged robots are different from each other. It also talks about performance indices like the Froude number and the specific resistance, as well as how things will change in the future. The authors in [17] made a web interface that lets you control and talk to multiple robots using ROS topic identification and registration for autonomous robots. The Gazebo simulator sends out all robots to interact with a user. To test how well the system works, the number of robots was increased. The big O representation was used to analyze the running-time complexity of algorithms. The experiment result indicated that the autonomous robot registration was successful and that the communication performance decreased gradually. The study by Rashid et al. [18] introduces a new algorithm,

called cluster matching, for multirobot localization and orientation. It involves a distance IR sensor scanning the robots and estimating the absolute positions and orientations of a number of the team robots without knowing their IDs. The localization and orientation of robots not visible to the distance IR sensor are obtained by collecting the information coming from the on-board sensors and reconstructing a complete map of the team distribution. Simulation scenarios are implemented on tens of robots to show the performance of the algorithm.

The main problems have been considered and are our motivation to move forward with the suggested oversight strategies. These problems are listed sequentially as follows:

- (1) Navigation is the first and main problem for a mobile robot that needs to navigate its unknown environment and define its location so that it can design a path to other destinations. SLAM technology is considered the dominant solution to such problems.
- (2) The other challenge with SLAM technology occurs during navigation in unknown environments since the robot takes a long time to explore the whole environment to build its map. When another robot has to navigate the same environment, it will rebuild a new map. So, we need to save all constructed maps, while the storage capacity needs to store the acquired data, such as maps, within private or public clouds, so that it can access them at any time and from anywhere via the Internet.

The principal contributions made by this research project are listed as follows:

- (1) Simulating and implementing the SLAM algorithm in an unknown environment that is built on the Rviz simulator on the ROS. Following that, when the number of obstacles reaches 12, study and evaluate a mobile robot platform named TurtleBot3 Burger in the Rviz simulator and implement the SLAM algorithm for different environments based on the Rviz library, which is built on the ROS in the Linux operating system. Also, control the robot and perform this process remotely.
- (2) Simulating an Amazon Web Service (AWS) cloud server to store maps made by robots and to store and download different required files (PDF, words, video, and photos), which can be accessed through an authentication process (username and password) overseen by the cloud server administrator.

In this study, SLAM is implemented by using a mobile robot named Turtlebot3 Burger, using gmapping tools in ROS based on the Rviz platform, and by placing different objects on the map and in different scenarios. This article is organized as follows: Section 2 discusses navigation and simultaneous localization and mapping (SLAM) and describes hardware and software tools; Section 3 discusses Modeling of Mobile Robot Navigation, and Section 4 displays and discusses the simulation results. Finally, the conclusion of this paper is represented in Section 5.

## 2. Research Methods and Tools

*2.1. Navigation and SLAM.* The navigation of robotics denotes the activity of a robot to control its location in relation to a specific reference and then outline a track to arrive at the wanted location. It can contain a set of jobs such as track planning, localization, and mapping [19]. The procedure for building a map demands large amounts of storage space and is computationally heavy. Moreover, searching for a map demands large quantities of information and data, and this is a huge challenge, especially when the region of navigation is large. CR delivers an actual, favorable solution for the upcoming cloud based on navigation. Also, the cloud not only supplies storage area to save a large quantity of map data but, moreover, supplies processing strength to make it easy to quickly build and search the map [20]. A prerequisite for successfully navigating is the ability to localize and create maps for unknown environments, which allows the robot to navigate in them using SLAM.

SLAM was first presented in mid-1986 by Smith and Durrant-Whyte [21]. SLAM is considered one of the AI techniques that benefits from the mobile robot's navigation in an unknown environment to construct a new map for this environment. When a mobile robot is placed in an unknown environment, it has to move and acquire data about landmarks in the environment by using its sensors, while at the same time, it has to assign its position concerning that created map. The landmark is defined as a symmetrical object that can be easily distinguished from a scanning sensor, and it is utilized to update the evaluated position of the robot [22]. Therefore, a good identification based on good benchmarks is important to support the operation of SLAM, as it cannot rely only on the frequently inaccurate odometry approach, which is the most common technique that is utilized in robotics to evaluate its present position. The odometer function integrates growing information as the motion of the wheels relative to the starting position is recorded. So, the SLAM problem can be defined as a case study of a chicken-or-egg problem, since building a map of the environment needs the robot position as well as assigning the robot position needs the map of the environment. The main problem with SLAM is that the measurements that are read from the sensors are always noisy and their positions are not always certain because the robot is moving.

*2.2. TurtleBot3 Burger.* The TurtleBot3 Burger is a small mobile robot that runs on ROS and can be programmed [23]. The TurtleBot3 burger is one of two robots, with the TurtleBot3 waffle being the other. The waffle is lower, but it is wider all around. The robot's base can move around, thanks to two wheels and a metal ball that keeps it steady. It can be fully programmed with open source, and the software that comes with it is licensed under an Apache 2.0 license. Figure 1 shows the three different TurtleBot3 versions, and Figure 2 shows the TurtleBot3 Burger, while Table 1 shows the specifications of the TurtleBot3 Burger.

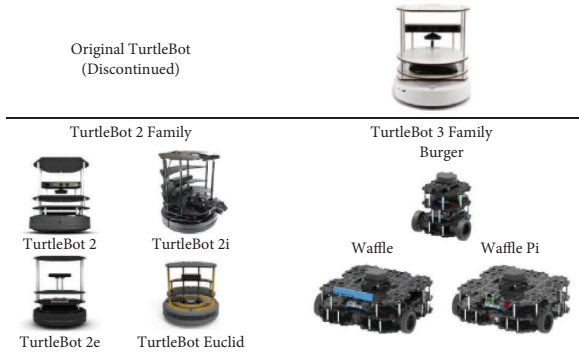


FIGURE 1: Turtlebot family.

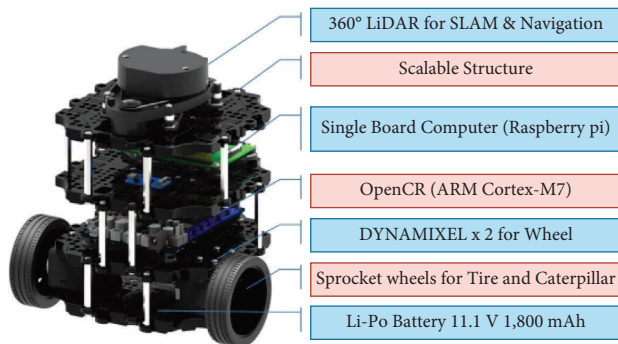


FIGURE 2: The TurtleBot3 Burger.

**2.3. Robot Operating System (ROS).** ROS is an operating system for robots that includes drivers, libraries, and tools that can be used to build and improve robot systems [24]. Willow Garage made it in 2007, and it is now the most popular and widely used robotics platform. ROS has Linux as a command instrument and an interprocess communication system [25]. It contains many autonomous nodes, each of which communicates with the other nodes by utilizing publish or subscribe messaging patterns. ROS relies on various languages, such as Python and C++, which can be utilized with each other in the ROS system, which runs on the Ubuntu Linux operating system [26].

**2.4. Rviz Platform.** In computer science and computer imaging, it is necessary to describe and visualize information in the real world. To achieve this purpose, several structures and different types of algorithms are proposed for data imaging world. Unfortunately, often the data stream platform that is known to handle the issues in visualization systems is not elastic and is not suitable to imagine newly invented data frames and algorithms since this sketch can only admit specified data frameworks. So, to handle this issue, a novel instrument of visualization that is defined by Rviz is suggested [27]. Rviz is defined as a 3D imaging instrument in ROS that allows us to imagine what the mobile robot will grasp and see in any area. Rviz can also imagine all of the types of sensors on the robot, which include laser scanners, cameras, and point clouds. Observing the raw values of the sensors [28], Rviz has an autonomous view of the input data structures.

TABLE 1: Specification of the TurtleBot3 Burger.

Items	TurtleBot3 Burger specification
Maximum translational velocity	0.22 m/s
Maximum rotational velocity	2.84 rad/s (162.72 deg/s)
Maximum payload	15 kg
Size ( $L \times W \times H$ )	138 mm $\times$ 178 mm $\times$ 192 mm
Weight (+SBC + battery + sensors)	1 kg
SBC (single board computers)	Raspberry Pi
Embedded controller	OpenCR (32 bit ARM <sup>®</sup> Cortex <sup>®</sup> - M7)
Sensor	HLS-LFCD2 3-axis gyroscope 3-axis accelerometer 3-axis magnetometer

### 3. Modeling of Mobile Robot Navigation

In this part, the Rviz platform is proposed to use SLAM technology to simulate how mobile robots move through an unknown environment, control the robot, and do all of these remotely. For practical considerations, a mobile robot platform named TurtleBot3 Burger has been adopted here to be used to do tests in the Rviz library built on ROS on Linux. The procedure begins with navigating in an unknown indoor environment with many obstacles, which it must detect, and creating a map of this unknown environment. As a result, the SLAM algorithm accomplishes all of these goals and can be controlled via cloud services access, also known as AWS. Figure 3 shows the software architecture that came out of this. The program is run on a master PC, which the robot acts as a slave to. A “roscore,” which is the central ROS process that connects nodes, needs to be started by the hand by running a command on the RPi through an SSH connection. A similar process is started in the background on the master PC by the robotics system toolbox. After the two processes have been set up, they share data that can be used on either the PC or the RPi. The TurtleBot 3 and the remote computer must be configured to set up communications on the same network. Control of the remote computer and storage space is required to store acquired data, such as maps, in private or public clouds, so that it can access them at any time and from anywhere via the Internet. The procedure for SLAM techniques in ROS with AWS is shown in Figure 4.

**3.1. Cloud Computing.** IT departments and developers can focus on what matters most by using cloud computing instead of doing undifferentiated tasks such as capacity planning, maintenance, and buying. A variety of models and deployment strategies have emerged as cloud computing has grown in popularity, in order to meet the various needs of users. Depending on the type of cloud service you use and the deployment strategy, you have varying degrees of control, flexibility, and management. Knowing the differences between infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS), as well as what deployment methodologies you can use, will help you

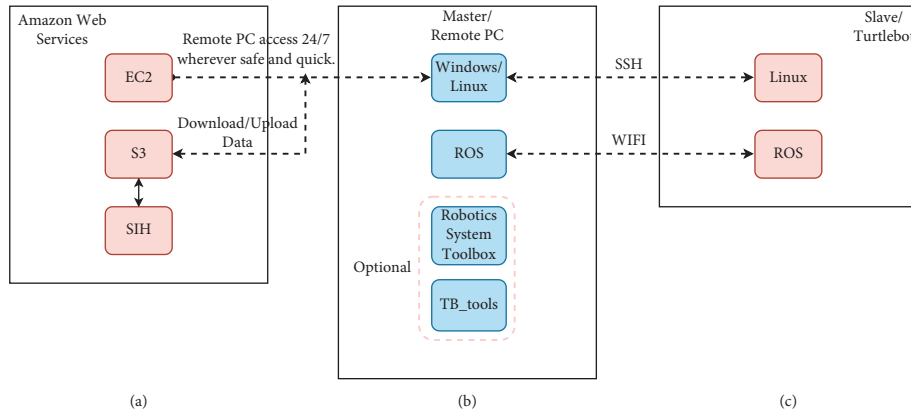


FIGURE 3: Software architecture. (a) AWS services are used to remotely control (services explained in the next subsection 3.2). (b) A remote PC is used to control TurtleBot3. (c) The TurtleBot3 used for SLAM and navigation techniques.

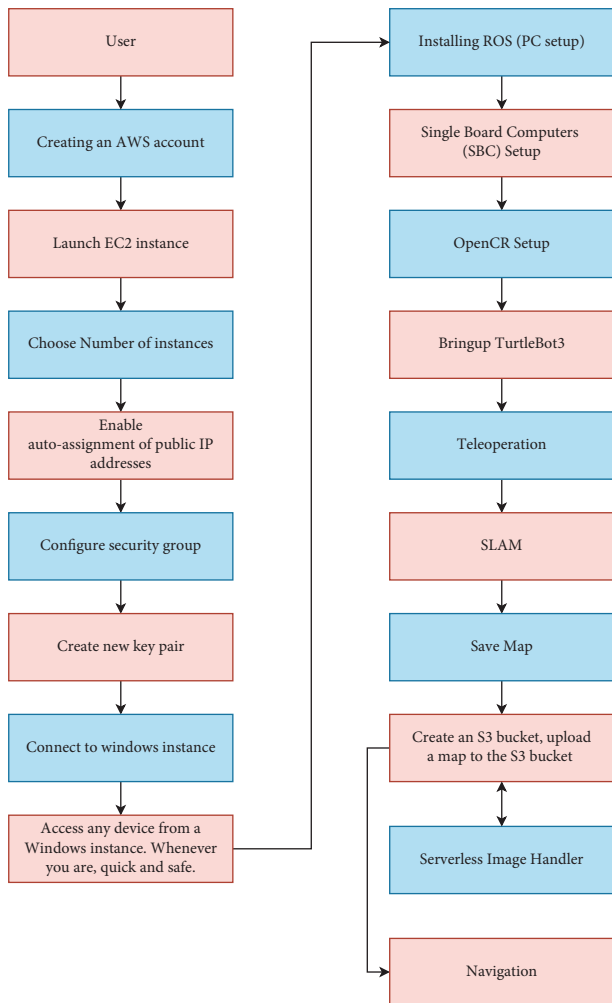


FIGURE 4: SLAM technique procedure in ROS.

choose the set of services that are best for your needs [22]. As far as the features they offer, IaaS, PaaS, and SaaS are different from each other in the following ways: IaaS gives the most freedom when it comes to hosting custom-built apps and storing data in a general data center. PaaS allows developers to spend more time developing applications while

the provider manages the infrastructure. SaaS offers ready-to-use and out-of-the-box solutions to meet specific business needs. Figure 5 illustrates the distinctions between IaaS, PaaS, and SaaS. In this work, AWS is a provider of IaaS. As the most complete and widely used cloud platform in the world, AWS provides over 200 fully functional services from data centers around the world. Millions of clients use AWS to reduce costs, increase agility, and accelerate innovation, including the largest corporations, most successful governmental organizations, and fastest-growing start-ups. Compared to other cloud providers, AWS offers a much wider range of services and features within those services. As a result, moving your current applications to the cloud is quicker, simpler, and more cost-effective, and you can build almost anything you can think of. In the next subsection, some of these services are explained.

3.2. *Cloud Services Access (AWS)*. The collection of cloud services offered by Amazon.com, Inc. is known as AWS services. It started out in July 2002 by offering storage and computing services. Since then, it has grown into a flexible cloud platform with IaaS, PaaS, and SaaS services, as well as a number of other services.

AWS is dispersed among numerous datacenters located in various cities around the globe. Every location is a region, and each region is made up of physically distinct datacenters called availability zones. Each service provided distinct availability zones and regions, with some limited to a single availability zone and others spanning multiple regions. Some services also provide replication capabilities across regions and availability zones. The following is an explanation of the different AWS services that are important to this study:

- (1) The simple storage service, also known as S3, is Amazon’s cloud-based storage service. It allows users to store standard files (objects) up to 5 TB in size and only charges for the storage space actually used. It uses buckets, each of which may hold an infinite number of things and has a name that is globally unique [29]. A bucket is located in a region and that region contains numerous places for storing all of its data. Through API requests and the management

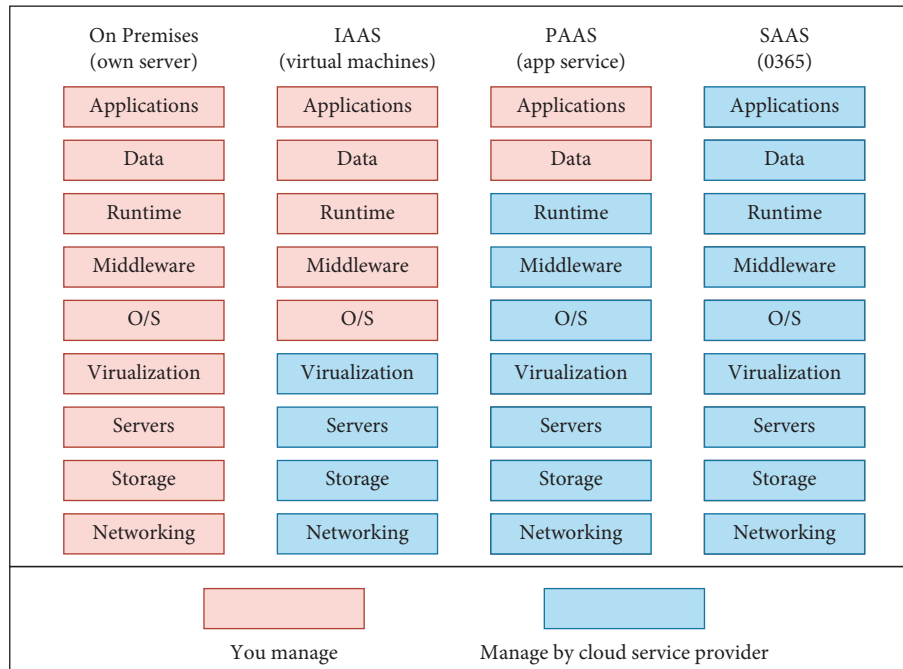


FIGURE 5: IaaS, PaaS, and SaaS.

console, data can be moved to another region. A key that mimics a file path on \*NIX systems is used to uniquely identify each item in a bucket. They are all accompanied by metadata, and each one can have a different set of rights applied to it. An object may be made publicly accessible, after which it may be downloaded using an HTTP URL. S3 can be accessed through the AWS management web interface, SOAP, and REST APIs, and access can be controlled with the help of access keys and certificates. As Figure 6 shows, the Management Console, the command-line interface (CLI), SDKs, and third-party tools can all upload and download objects to S3 over HTTPS.

- (2) Elastic Compute Cloud, or EC2, is the name of Amazon's IaaS product. It was introduced in 2006 and provides virtual machines with various specs across all the AWS availability zones [30]. An AMI, or amazon machine image, is a customized operating system that is installed on an EC2 virtual instance. It is possible to build customized AMIs by either altering an existing image or by developing a brand new one if any of the available official or unofficial AMIs does not satisfy the user's demands. Custom AMIs can be used privately, in which case they can hold crucial data like authentication keys for S3 access, or they can be made available to the public.

When EC2 first came out in 2006, the only storage options for instances were S3 for long-term storage and ephemeral instance storage for files that only needed to be kept for a short time. Elastic block storage, amazon's storage area network (SAN) solution for EC2, was introduced in 2008, which provides EC2 instances with access to networked block devices for permanent storage; elastic block

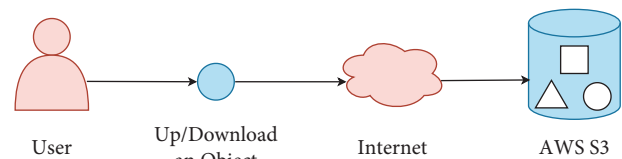


FIGURE 6: AWS S3 design.

storage (EBS) volumes can employ up to 4,000 preallocated input/output operations per second (IOPS) and have a maximum capacity of 1 TB. The cost of volumes is determined by their size and for provisioned IOPS, by the number of hours allotted to them. EC2 instances can be set to be EBS optimized when they are launched, improving performance. A key pair (to verify your identity) and a security group are specified when launching your instance to secure it. The private key of the key pair you specified when launching your instance must be provided when connecting to it, as shown in Figure 7.

**3.3. Serverless Image Handler.** With the aid of this solution, cost-effective image processing can be started in the AWS cloud using a serverless architecture. The architecture is designed for dynamic image manipulation and combines AWS services with sharp open-source image processing software and maintains high-quality images with the help of the solution's dynamic image handling. Figure 8 shows how to set up the serverless architecture.

## 4. Experimental and Simulation Results

This section shows the results of the implementation of SLAM in ROS, which was simulated in Rviz using an AWS EC2 instance service for real-time SLAM mobile robots in





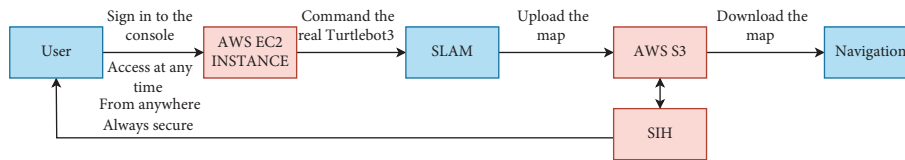


FIGURE 9: Scenario process.

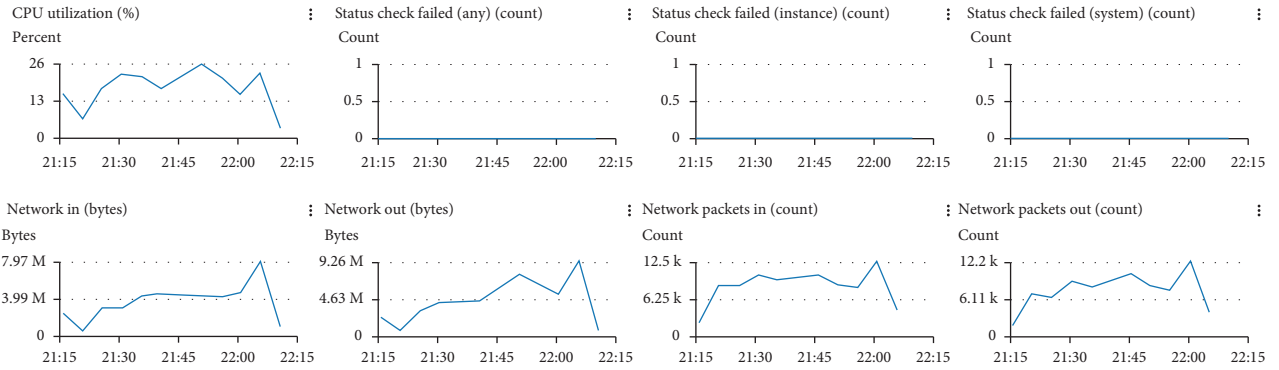
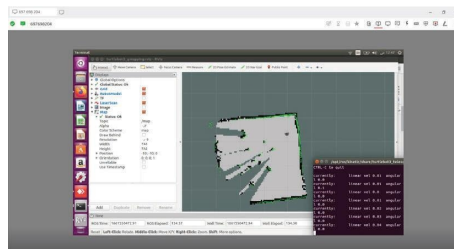
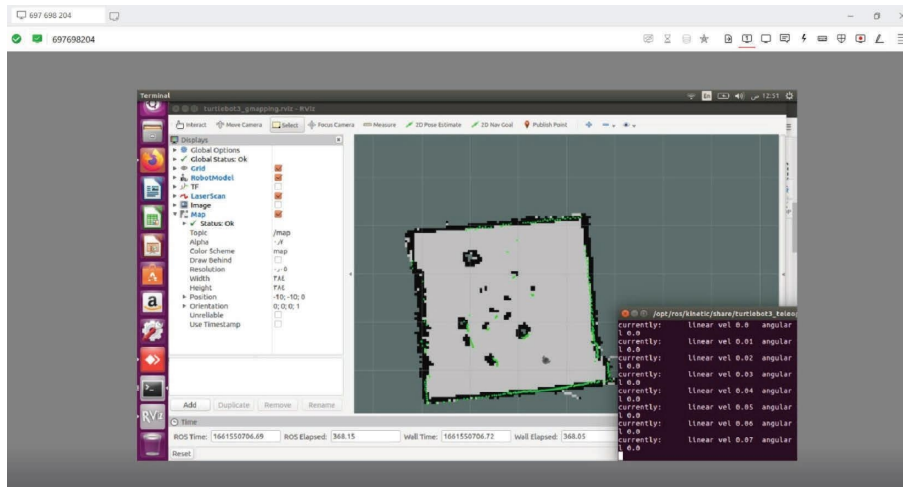


FIGURE 10: EC2 monitoring.



(a)



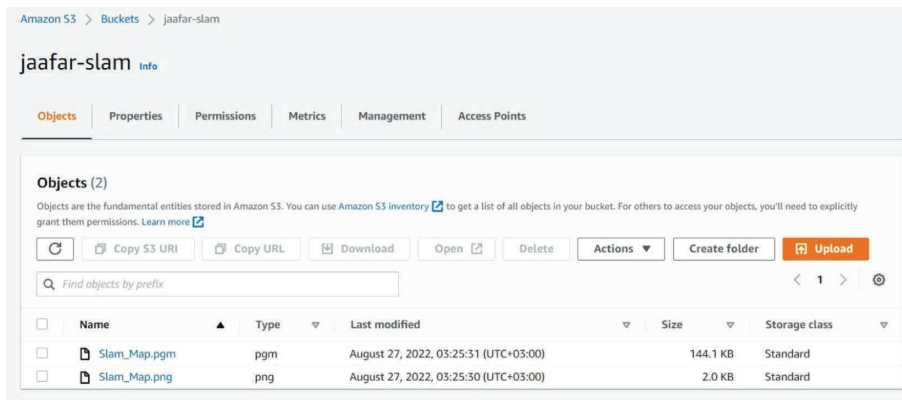
(b)

FIGURE 11: Via AWS EC2: (a) the start of the slam; (b) end of slam.

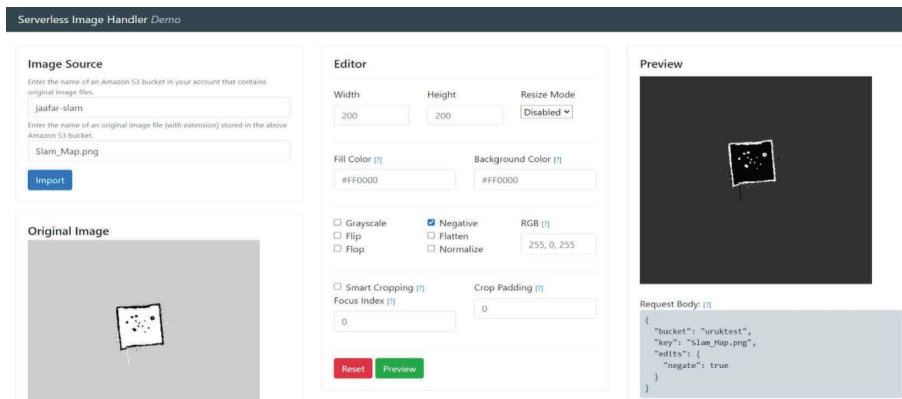




FIGURE 12: Rviz platform environment vs. real environment.



(a)



(b)

FIGURE 13: (a) Upload of the environment to AWS S3; (b) the SIH solution.

leaves a particular instance; and status check failed reports whether the instance has recently passed the system status check. Table 3 is displaying the Amazon EC2 Monitor.

Figure 10 shows that the robot is fully controlled by AWS EC2. This shows the beginning of the process of drawing the map of an unknown place and the end of the process. This method can allow us to control the robot even if it is in one country and the person in control is in another country. The properties of the robot are mentioned in subsection 2.2. It is possible to control the speed of the robot so that the maximum speed is 0.22 m/s. This is the relationship between the speed of the drawing and the speed of the robot.

In Figure 11, it can be noted by the visual comparison that the arena drawn by Rviz is identical to the “real field,” where the robot was able to discover all the obstacles of different sizes. It is now possible to save the map in AWS S3 and use it in the future from the same robot or any other robot without having to redraw it by redownloading it from AWS S3 and getting the most out of it. The same principle can be applied to a group of sites, and these maps can be used to make a real application that serves those sites, like a large store or military application, or to help find survivors after a disaster.

Figure 13 shows the process of uploading the file in pgm format, which is slam, and png format, which is the image

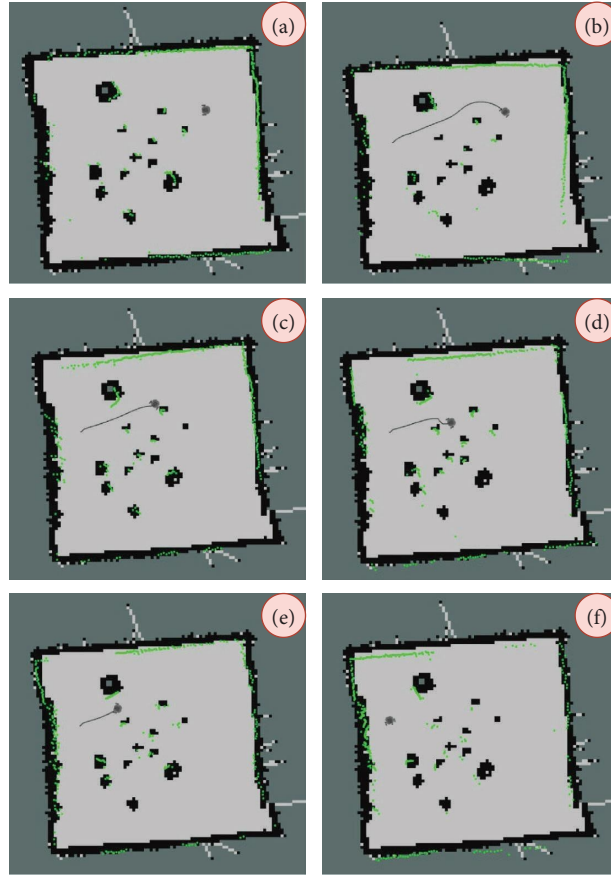


FIGURE 14: Navigation process.

TABLE 3: Amazon EC2 monitor.

Metric	Max value after one hour
CPU utilization	26 percentage
Network in	8 MB
Network out	9.26 MB
Network packets in	12.5 k count
Network packets out	12.2 k count
Status check failed	0
Status check failed (instance)	0
Status check failed (system)	0

format that can be used through SIH, where there are several options shown in figure (b) to deal with the image and clarify it, where this process is a feedback on the nature of the place and clarification of the image by the user and where it is possible to save the image after modification in S3 as well.

After downloading the map, it is now possible to direct the robot from one place to another by suggesting the

destination. The robot, depending on the available information on the map, chooses the most appropriate route for itself. The robot can avoid collisions with fixed and moving obstacles, where the distance depends on the lidar settings and ranges from 0.12 meters to 3.5 meters. Figure 14 shows the process of moving the robot from the starting point to the point to which it is intended and the path. The tuning guide for navigation is shown in Table 4.

TABLE 4: Tuning guide for navigation in Figure 14.

Item	Value
obstacle_range	3
raytrace_range	3.5
max_vel_x	0.22
min_vel_x	- 0.22
max_vel_trans	0.22
min_vel_trans	0.11
max_vel_theta	2.75
min_vel_theta	1.37
acc_lim_x	2.5
acc_lim_theta	3.2
xy_goal_tolerance	0.05
yaw_goal_tolerance	0.17
sim_time	1.5
vx_samples	20
vy_samples	0
vth_samples	40

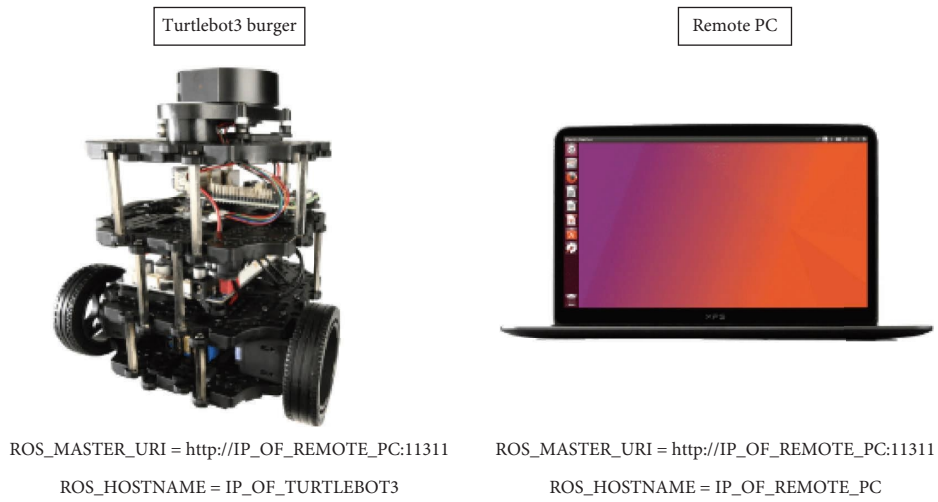


FIGURE 15: ROS master running on a remote PC.



FIGURE 16: LDS-01 and LDS-02.

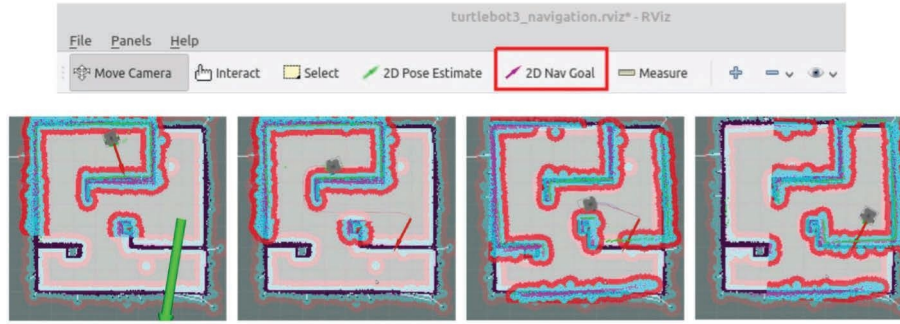


FIGURE 17: Estimate initial pose.

TABLE 5: Specification of Turtlebot3 Burger.

Items	Burger
maxUrange	The lidar sensor's maximum practical range is set by this setting
map_update_interval	Setting a value too large will necessitate more processing power for the map calculation; the smaller the value, the more frequently the map is updated
minimumScore	The minimum score value that determines whether the sensor's scan data matching test is successful or unsuccessful is established by this option. This can lessen errors in the robot's predicted position over a large area if configured properly
linearUpdate	When the robot has moved more than this distance, the scanning process will start
angularUpdate	The robot will begin the scan process if it detects more than this amount. Setting this number lower than linearUpdate is advised

TABLE 6: Tuning guide of Turtlebot3 Burger.

Items	Burger
inflation_radius	The obstacle's inflation area is reduced by this parameter. The route would be designed so that it would not pass through this region. Setting this to a value greater than the robot's radius is secure
cost_scaling_factor	The cost value is multiplied by this factor. Because of the reciprocal relationship, the cost decreases as this parameter is increased
max_vel_x	The maximum value of translational velocity is set for this factor
min_vel_x	The translational velocity's minimum value is set for this factor. If this is set to zero, the robot can go backward
max_trans_vel	Maximum translational velocity value in reality; this is the fastest the robot is capable of going
min_trans_vel	The actual translational minimum velocity; this is the fastest the robot can go
max_rot_vel	Maximum rotational velocity value in reality; this is the fastest the robot is capable of going
min_rot_vel	The precise rotational velocity; this is the fastest the robot can go
acc_lim_x	The translational acceleration limit's actual value
acc_lim_theta	The rotational acceleration limit's actual value
xy_goal_tolerance	The permitted x, y distance when the robot is in its desired pose
yaw_goal_tolerance	The yaw angle is permitted once the robot has assumed its final position
sim_time	This factor is advanced in the simulation by a few seconds. Too high a value prevents quick rotation, whereas too low a value allows passage through a limited space. Differences in the length of the yellow line can be seen in the image below

## 5. Conclusion

The approach proposed in this work is to solve the SLAM problem, controlling the robot via the AWS cloud, creating a map database, and storing it without the need to redraw the map, as well as using an authentication process (username and password) under the supervision of the cloud administrator, and using the SIH to initiate cost-effective image

processing in the AWS cloud. The practical results were shown using a real TurtleBot3 Burger robot, the success of the process of creating a complex map containing 12 obstacles of different sizes, where the robot was remotely controlled by the EC2 server with certain specifications, and after one hour, the EC2 monitor value was as follows: CPU utilization: 26 percent, 8 MB network in, 9.26 MB network out, network packets. In the 12.5 K count, network packets

out: 12.2 k, status check failed 0. After that, the map has been successfully stored in S3 for future use. Also, the process of moving the robot from start to target inside the map was tested, and the robot was able to move and avoid collision with the obstacles it encounters on the way. From the foregoing, a person in a certain country can control a robot located in another country, as this method offers many prospects for use in the military, medical, and educational fields. For future work, the extended Kalman filter (EKF) algorithm can be used to improve the SLAM, and EKF-SLAM is used for various unknown environments. Then, intelligent optimization algorithms are suggested to improve the performance of the mobile robot's EKF-SLAM path.

## Appendix

### A. Detailed Steps for ROS in SLAM and Navigation

#### (1) Installing ROS (PC setup)

The instructions were tested on Linux with Ubuntu 16.04 and ROS Kinetic Kame, which are talked about as follows.

Step 1: Install ROS on remote PC

First, the robotic operating system (ROS) is installed on an Ubuntu Linux OS. The steps to complete the installation are clearly shown on the website <https://www.ubuntu.com/>.

In this work, the version of Ubuntu 16.04 LTS is used. "The Kinetic" is the codename for the latest ROS distribution, which is compatible with Ubuntu 16.04. ROS is put on Ubuntu by using terminal software. To start the installation, different steps were taken.

Step 2: Open the terminal and type each of the following instructions individually.

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ wget https://raw.githubusercontent.com/ROBOTIS-GIT/robotis_tools/master/install_ros_kinetic.sh
$ chmod 755./install_ros_kinetic.sh
$ bash./install_ros_kinetic.sh
```

Step 3: Install the required packages for ROS.

```
$ sudo apt-get install ros-kinetic-joy ros-kinetic-teleop-twist-joy \
ros-kinetic-teleop-twist-keyboard ros-kinetic-laser-proc \
ros-kinetic-rgbd-launch ros-kinetic-depthimage-to-laserscan \
ros-kinetic-rosserial-arduino ros-kinetic-rosserial-python \
ros-kinetic-rosserial-server ros-kinetic-rosserial-client \
ros-kinetic-rosserial-msgs ros-kinetic-amcl ros-kinetic-map-server \
```

```
ros-kinetic-move-base ros-kinetic-urdf ros-kinetic-xacro \
```

```
ros-kinetic-compressed-image-transport ros-kinetic-rqt* \
```

```
ros-kinetic-gmapping ros-kinetic-navigation ros-kinetic-interactive-markers
```

Step 4: Set up the TurtleBot3 packages.

```
$ sudo apt-get install ros-kinetic-dynamixel-sdk
$ sudo apt-get install ros-kinetic-turtlebot3-msgs
$ sudo apt-get install ros-kinetic-turtlebot3
```

Step 5: Name your TurtleBot3 model.

```
$ echo "export TURTLEBOT3_MODEL = burger"
>> ~/.bashrc
```

Step 6: Network Configuration.

Figure 15 shows an example of network configuration when ROS Master is running on a remote PC.

Use the command below to connect your PC to a WiFi device and discover the assigned IP address.

```
$ ifconfig
```

Now, use the command listed in the following to update the ROS IP settings after opening the file.

```
$ nano ~/.bashrc
```

Place the cursor at the end of the line, then use the ifconfig command to replace localhost's IP address in the ROS MASTER URI and ROS HOSTNAME variables.

Use the command below to source the bashrc file.

```
$ source ~/.bashrc
```

#### (2) Single Board Computers (SBC) Setup

Step 1: TurtleBot 3 SBC Image Download For your hardware and ROS version, download the appropriate image file. images using the Kinetic version of the Raspberry pi OS (raspbian OS).

Step 2: The downloaded picture file must be unzipped. Save the .img file on the local disk after being extracted.

Step 3: Burn the image file. There are many image-burning tools available (the Linux disks utility and raspberry pi imager are two examples).

Step 4: The raspberry pi should start up.

- Connect the HDMI cable from the monitor to the raspberry pi's HDMI port
- Attach input devices to the USB port on the raspberry pi
- Insert your microSD card
- To start the raspberry pi, connect the power (either using USB or OpenCR)

Step 5: Configure the Raspberry Pi

- Connect to the Wi-Fi network linked to the PC once raspbian OS is up and running

- (b) Use the following command to determine the raspberry pi's assigned ip address. The wlan0 section is typically where you will find the raspberry pi's wireless IP address
 

```
$ ifconfig
```
- (c) Open the terminal on your PC, enter the ip address of the raspberry pi, and connect. turtlebot has been set as the default password
 

```
$ ssh pi@{IP_ADDRESS_OF_RASPBERRY_PI}
```
- (d) After logging in, run the following commands on the raspberry pi to get the time in sync
 

```
$ sudo apt-get install ntpdate
$ sudo ntpdate ntp.ubuntu.com
```
- (e) Open the configuration interface for the raspberry pi
 

```
$ sudo raspi-config
```
- (f) Expand the file system by choosing advanced options, then click Exit.
- (g) ROS network configuration
 

```
$ nano ~/.bashrc
```
- (h) Modify the ip addresses for ROS MASTER URI and ROS HOSTNAME at the file's end
 

```
export ROS_MASTER_URI = http://{IP_ADDRESS_OF_REMOTE_PC}:11311
export ROS_HOSTNAME = {IP_ADDRESS_OF_RASPBERRY_PI_3}
```
- (i) After saving the document, close the nano editor
- (j) Apply modifications using the following command
 

```
$ source ~/.bashrc
```

#### Step 6: Configuration new lds-02

The TurtleBot3 LDS has been replaced by the LDS-02 in models made in 2022. To use TurtleBot3's SBC, adhere to the following directions. The two categories of leaders are shown in Figure 16.

- (a) Install the TurtleBot3 package update and the LDS-02 driver
 

```
$ sudo apt update
$ sudo apt install libudev-dev
$ cd ~/catkin_ws/src
$ git clone -b develop https://github.com/ROBOTIS-GIT/ld08_driver.git
$ cd ~/catkin_ws/src/turtlebot3 && git pull
$ rm-r turtlebot3_description/turtlebot3_teleop/
turtlebot3_navigation/turtlebot3_slam/
turtlebot3_example/
$ cd ~/catkin_ws && catkin_make
```
- (b) The LDS MODEL should be exported to the bashrc file. Use LDS-01 or LDS-02 depending on the model of LDS.
 

```
$ echo 'export LDS_MODEL = LDS-01' >>
~/.bashrc
```
- (c) Use the command below to make the modifications.
 

```
$ source ~/.bashrc
```

### (3) OpenCR Setup

Step 1: Utilize the micro-USB connection to join the OpenCR and the raspberry pi

Step 2: To upload the OpenCR firmware, install the necessary packages on the raspberry pi

```
$ sudo dpkg--add-architecture armhf
```

```
$ sudo apt-get update
```

```
$ sudo apt-get install libc6:armhf
```

Step 3: Choose between the names Burger and Waffle for the OPENCR MODEL depending on the platform

```
$ export OPENCNCR_PORT = /dev/ttyACM0
```

```
$ export OPENCNCR_MODEL = burger
```

```
$ rm-rf./opencr_update.tar.bz2
```

Step 4: After downloading the loader and firmware, extract the file.

```
$ wget https://github.com/ROBOTIS-GIT/OpenCR-Binaries/raw/master/turtlebot3/ROS1/latest/opencr_update.tar.bz2
```

```
$ tar-xvf opencr_update.tar.bz2
```

Step 5: Uploading firmware to OpenCR

```
$ cd./opencr_update
```

```
./update.sh
```

```
$OPENCNCR_PORT
```

```
$OPENCNCR_MODEL.opencr
```

### (4) Bringup

(1) Run Roscore on a computer

```
$ roscore
```

(2) Bringup TurtleBot3

Step 1: On a computer, launch a new terminal and use the raspberry pi's ip address to connect.

```
$ ssh pi@{IP_ADDRESS_OF_RASPBERRY_PI}
```

Step 2: Bring up basic packages to start TurtleBot3 applications

```
$ roslaunch turtlebot3_bringup
turtlebot3_robot.launch
```

### (5) Teleoperation

Different remote controllers can be used to teleoperate the TurtleBot3 (Keyboard, RC100, PS3 Joystick, XBOX 360 Joystick, Wii Remote). Verify that SBC and ROS versions support the required ROS packages. In this study, keyboard teleoperation is used. For keyboard teleoperation, start the TurtleBot3 teleop key node from the remote PC. You should substitute the name of your model, such as burger, waffle, or waffle pi, for the \$TB3 MODEL parameter.

```
$ export TURTLEBOT3_MODEL =
${TB3_MODEL}
```

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

The following set of instructions will appear in the terminal window if the node is successfully launched



## Control Your Turtlebot3mmc1

Moving around

$w$

$a$   $s$   $d$

$x$

$w/x$ : increase/decrease linear velocity

$a/d$ : increase/decrease angular velocity

space key,  $s$ : force stop

CTRL-C to quit

## (6) SLAM

Step 1: Launch the SLAM node from a new terminal that is open in remote PC. Gmapping is the standard SLAM technique. Frontier exploration, Hector, Karto, and Cartographer are additional approaches. For the TurtleBot3 MODEL parameter, select the appropriate keyword from burger, waffle, and waffle pi.

```
$ export TURTLEBOT3_MODEL=burger
```

```
$ roslaunch turtlebot3_slam turtlebot3_slam.launch
```

Once the SLAM node is operational, TurtleBot3 will use teleoperation to explore uncharted territory on the map. It is crucial to refrain from jerky motions, including abrupt changes in linear and angular speed. It is a good idea to scan the entire map while using the TurtleBot3 to generate a map.

Step 2: Tuning Guide for slam: Gmapping offers a variety of parameters to adjust performance for various settings. When configuring GMapping parameters, this tuning guide offers advice. The file TurtleBot3 slam/config/gmapping params.yaml is defined in Table 5 with the following parameters:

## Step 3: Save Map

The map is produced using the robot's odometry, tf, and scan data. The map information was shown in the Rviz window as the TurtleBot3 was moving. Save the map data to the local drive after creating a complete map of the area so you can use it later. Run the map saver node of the map server package to generate map files. In the directory where the map saver node is launched, the map file is saved. If no other file name is specified, the word "Map" will be used as the default and will create the files map.pgm and map.yaml. The -f option specifies a folder location and file name when saving files. That command will save Map.pgm and Map.yaml to the user's home directory.

```
$ rosrunc map_server map_saver-f ~/map
```

Step 4: The two-dimensional occupancy grid map (OGM), which is popular in ROS, is utilized for the map. The saved map will resemble the following illustration, with the white areas denoting collision-free zones, the black areas denoting occupied and inaccessible zones, and the gray areas denoting uncharted territory. The navigation is done using this map.

## (7) Navigation

The goal of navigation is to move the robot through a given environment from one location to the desired location. A map with geometry data on the walls, furniture, and other objects in the environment is necessary for this. The map was created using the sensor's distance data and the robot's own position data, as we discussed in the SLAM section earlier. A robot can navigate from its current position to a destination pose on a map by using the map, encoder, IMU sensor, and distance sensor of the robot. The following are the steps to complete this task.

Step 1: Launch the navigation

```
$ export TURTLEBOT3_MODEL=burger
```

```
$ roslaunch turtlebot3_navigation turtlebot3_navigation.launch map_file:= $HOME/map.yaml
```

Step 2: Estimate initial pose

Prior to starting the navigation, initial pose estimation is necessary since it initializes the crucial AMCL parameters. TurtleBot3 must be in the right place on the map because the LDS sensor data neatly overlaps the map, as shown in Figure 17:

- (1) Click the 2D Pose Estimate button in the Rviz menu.
- (2) Drag the green arrow in the direction you want the robot to face by clicking on the map and setting the robot's destination. A marking that can indicate the robot's destination is this green arrow. The arrow is based on the  $x$  and  $y$  coordinates of the target, while the angle is based on the direction of the arrow. As soon as  $x$ ,  $y$ , and  $\theta$  are set, TurtleBot3 will start moving to the destination immediately.

- (8) The Tuning Guide for Navigation (shown in Table 6): The navigation stack has many parameters to change performance for different robots. This tuning is defined in

```
turtlebot3_navigation/param/  
dwa_local_planner_params_{TB3_MODEL}.yaml
```

## B. ROS for Obstacle Detection and Point Operation

Robot path planning (RPP) can be divided into two categories based on the environment where the robot is located: static (environment with fixed obstacles) and dynamic (the environment has moving obstacles). These two categories can each be further broken down into subgroups, including global path planning (GPP), which allows for the advance knowledge of all fixed and moving obstacles. As a result, the GPP can be prepared in advance of the robot moving (offline), and local path planning (LPP) is available. It is impossible to know beforehand what the environment is like here. Therefore, as the mobile robot travels through the world, sensors (online) gather data about what is nearby [33].

LDS data can move or stop the TurtleBot3. The TurtleBot3 burger stops moving when it sees something in its path. [remote PC] Start the file with the obstacle.

```
$          roslaunch          turtlebot3_example
turtlebot3_obstacle.launch
```

For point operation, the TurtleBot3 can be moved using 2D ( $x$ ,  $y$ ) points and  $z$ -angular movements. For instance, TurtleBot3 moves to the point ( $x=0.7$  m,  $y=0.2$  m) and then rotates  $60^\circ$  if you enter (0.7, 0.2, 60).

```
$          roslaunch          turtlebot3_example
turtlebot3_pointop_key.launch
```

## Data Availability

Data sharing is not applicable to this article as no datasets were generated or analyzed during the current study.

## Conflicts of Interest

The authors declare that they have no conflicts of interest.

## References

- [1] F. H. Ajeil, I. K. Ibraheem, M. A. Sahib, and A. J. Humaidi, "Multi-objective path planning of an autonomous mobile robot using hybrid PSO-MFB optimization algorithm," *Applied Soft Computing*, vol. 89, Article ID 106076, 2020.
- [2] A. Hussnain, *Application of Cloud Robotics for Automatic Manipulation*, University of Tampere, Tampere, Finland, 2018.
- [3] M. Lacity and E. Whitley, "service automation: robots and the future of work hashtag for twitter users: #lserobots department of management public lecture," 2016, <https://www.lse.ac.uk/Events/2016/05/20160509t1830vOT/Service-Automation>.
- [4] X. Ma and Y. Huang, "Research on mobile cloud robotics based on cloud computing," in *Proceedings of the 2016 International Forum on Management Education and Information Technology Application*, Guangzhou, China, January 2016.
- [5] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, Massachusetts Institute of Technology, Cambridge, MA USA, 2011.
- [6] J. Wan, S. Tang, H. Yan, D. Li, S. Wang, and A. V. Vasilakos, *Cloud Robotics: Current Status and Open Issues*, IEEE Access, Piscataway, NJ, USA, 2016.
- [7] I. Afanasyev, A. Sagitov, and E. Magid, "Ros-based slam for a gazebo-simulated mobile robot in image-based 3d model of indoor environment," in *Proceedings of the International Conference on Advanced Concepts for Intelligent Vision Systems*, pp. 273–283, Catania, Italy, October 2015.
- [8] R. Doriya, P. Sao, V. Payal, V. Anand, and P. Chakraborty, "A review on cloud robotics based frameworks to solve simultaneous localization and mapping (slam) problem," 2015, <https://arxiv.org/abs/1701.08444>.
- [9] K. Takaya, T. Asai, V. Kroumov, and F. Smarandache, "Simulation environment for mobile robots testing using ROS and Gazebo," in *Proceedings of the 2016 20th International Conference on System Theory, Control and Computing (ICSTCC)*, pp. 96–101, Sinaia, Romania, October 2016.
- [10] R. Limosani, A. Manzi, L. Fiorini, F. Cavallo, and P. Dario, "Enabling global robot navigation based on a cloud robotics approach," *International Journal of Social Robotics*, vol. 8, no. 3, pp. 371–380, 2016.
- [11] R. K. Megalingam, C. R. Teja, S. Sreekanth, and A. Raj, "Ros based autonomous indoor navigation simulation using slam algorithm," 2019, <http://www.ijpam.eu>.
- [12] G. Jiang, L. Yin, S. Jin, C. Tian, X. Ma, and Y. Ou, "A simultaneous localization and mapping (SLAM) framework for 2.5D map building based on low-cost LiDAR and vision fusion," *Applied Sciences*, vol. 9, no. 10, 2019.
- [13] P. Arena, C. Bonomo, L. Fortuna, M. Frasca, and S. Graziani, "Design and control of an IPMC wormlike robot," *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, vol. 36, no. 5, pp. 1044–1052, 2006.
- [14] A. Buscarino, L. Fortuna, M. Frasca, and A. Rizzo, "Dynamical network interactions in distributed control of robots," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 16, no. 1, 2006.
- [15] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.
- [16] S. Kajita and B. Espiau, "Legged robots," in *Springer Handbook Of Robotics*, pp. 361–389, Springer Berlin Heidelberg, Berlin, Germany, 2008.
- [17] U. U. S. Rajapaksha, C. Jayawardena, and B. A. MacDonald, "Design, implementation, and performance evaluation of a web-based multiple robot control system," *Journal of Robotics*, vol. 2022, Article ID 9289625, pp. 1–24, 2022.
- [18] A. T. Rashid, M. Frasca, A. A. Ali, A. Rizzo, and L. Fortuna, "Multi-robot localization and orientation estimation using robotic cluster matching algorithm," *Robotics and Autonomous Systems*, vol. 63, pp. 108–121, 2015.
- [19] J. S. Pershina, S. Y. Kazdorf, and A. V. Lopota, "Methods of mobile robot visual navigation and environment mapping," *Optoelectronics, Instrumentation and Data Processing*, vol. 55, no. 2, pp. 181–188, 2019.
- [20] T. C. Smith, *An Investigation on a Mobile Robot in a ROS Enabled Cloud Robotics Environment*, Georgia Southern University, Statesboro, GA, USA, 2016.
- [21] G. Tuna, K. Gulez, V. Cagri Gungor, and T. Veli Mumcu, "Evaluations of different simultaneous localization and mapping (SLAM) algorithms," in *Proceedings of the IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, pp. 2693–2698, Montreal, Quebec, October 2012.
- [22] J. Ahmed Abdulsahab and D. Jasim Kadhim, "Robot path planning in unknown environments with multi-objectives using an improved COOT optimization algorithm," *International Journal of Intelligent Engineering and Systems*, vol. 15, no. 5, pp. 548–565, 2022.
- [23] R. Amsters and P. Slaets, *Turtlebot 3 as a Robotics Education Platform*, Purdue University, West Lafayette, IN, USA, 2020.
- [24] H. Deng, J. Xiong, and Z. Xia, "Mobile manipulation task simulation using ROS with MoveIt," in *Proceedings of the 2017 IEEE International Conference on Real-Time Computing and Robotics (RCAR)*, pp. 612–616, Okinawa, Japan, July 2017.
- [25] S. Haider Abdulredah and D. Jasim Kadhim, "Developing a real time navigation for the mobile robots at unknown environments," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 20, no. 1, p. 500, 2020.
- [26] M. Robert and L. Avanzato, "Work in progress: introductory mobile robotics and computer vision laboratories using ros and matlab introductory mobile robotics and computer vision laboratories using ros and matlab," in *Proceedings of the 2018 ASEE Annual Conference and Exposition*, Salt Lake City, UT, USA, June 2018.

- [27] F. Penné, *Implementation of Particle Filtering in a Turtlebot*, University of Chicago, Chicago, IL, USA, 2015.
- [28] H. Wang, W. Liu, F. Zhang, S. X. Yang, and L. Zhang, “A GA-fuzzy logic based extended Kalman filter for mobile robot localization,” in *Proceedings of the 2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, pp. 319–323, Zhangjiajie, China, August 2015.
- [29] J. Nadon, “Your content solution: an introduction to AWS S3,” in *Website Hosting and Migration with Amazon Web Services*, pp. 15–24, Apress, Berkeley, CA, USA, 2017.
- [30] R. Saini and R. Behl, “An introduction to aws—ec2 (elastic compute cloud),” 2020, [https://aws.amazon.com/pm/ec2-amd/?trk=17a47518-9bfe-4f1e-a67f-42bdd19264e5&sc\\_channel=ps&ef\\_id=EAIAIQobChMI3-fCvL3s\\_QIV2yMrCh20AArOEAAYASAAEgKkCvD\\_BwE:G:s&s\\_kwcid=AL!4422!3!525855180410!p!!g!!amazon%20web%20server!13385427590!122597168825](https://aws.amazon.com/pm/ec2-amd/?trk=17a47518-9bfe-4f1e-a67f-42bdd19264e5&sc_channel=ps&ef_id=EAIAIQobChMI3-fCvL3s_QIV2yMrCh20AArOEAAYASAAEgKkCvD_BwE:G:s&s_kwcid=AL!4422!3!525855180410!p!!g!!amazon%20web%20server!13385427590!122597168825).
- [31] Amazon, “What is amazon EC2? - amazon elastic compute cloud,” 2023, <https://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/concepts.html>.
- [32] Amazon, “Fast and cost-effective image manipulation with serverless image handler | aws architecture blog,” 2023, <https://aws.amazon.com/blogs/architecture/fast-and-cost-effective-image-manipulation-with-serverless-image-handler/>.
- [33] H. Miao and Y.-C. Tian, “Robot path planning in dynamic environments using a simulated annealing based approach,” in *Proceedings of the 2008 10th International Conference on Control, Automation, Robotics and Vision*, pp. 1253–1258, Hanoi, Vietnam, December 2008.