Design and Implementation of an Autonomous Service Robot for Hospital Isolation Room Using Robot Operating System 2 and Navigation 2

Muhammad Nurul Puji[1], Tubagus Ahmad Dwinandana[2], Tota Pirdo Kasih[3], Edmond Lee[1], Giovanni Benedict Davin Kamalo[1], Patrick Anthony[1], Matthew Kunaifi[1]
[1]Automotive & Robotics Program, Computer Engineering Department, Binus ASO School of Engineering, Bina Nusantara University, Jakarta 11480 Indonesia
[2]Product Design Program, Industrial Engineering Department, Binus ASO School of Engineering, Bina Nusantara University, Jakarta 11480 Indonesia
[3]Professional Engineering Program Department, Faculty of Engineering, Bina Nusantara University, Jakarta 11480 Indonesia

ARTICLE INFO

Article history:

Received November 25, 2024
Revised December 18, 2024
Accepted January 26, 2025

Keywords:

Autonomous navigation;
Service robot;
Web app;
Robot operating system;
SLAM

ABSTRACT

Healthcare-associated infections or nosocomial infections are infections that are acquired after admission to a hospital. This type of infection extends the length of stay, increases the cost of healthcare, and increases the mortality rate. This infection is caused by pathogens that are present in the hospital environment. These pathogens are transmitted when a hospital worker comes in contact with a patient or their environment. Thus, it is important to reduce the contact between them to stop the spread of pathogens. To reduce contact an autonomous service robot is utilized to deliver food or medicine to patients. This robot will be able to go to multiple target positions autonomously and will be controlled by a web application. Furthermore, the robot can provide a video call if the patient needs help. The robot platform used is the turtlebot3 and the software framework used is robot operating system 2 humble. The inflation radius and cost scaling parameters are tuned to increase the navigation success rate. Problems encountered during testing include glass windows not being detected by lidar, noisy lidar data, and obstacles being too low to be detected. These problems are solved using tape, costmap and laser filter, and keepout zones respectively. To evaluate the performance and capability of the app the robot is tested using a set of target locations entered on the app. During testing poor odometry causes localization error that causes recovery behaviors. The final system has a navigation success of 95%, with an average navigation speed of 0.17 m/s, and an average distance to target of 0.0587 m.

Corresponding Author:

Muhammad Nurul Puji, Automotive & Robotics Program, Computer Engineering Department, Binus ASO School of Engineering, Bina Nusantara University, Jakarta 11480 Indonesia
Email: muhammad.puji@binus.edu

1. INTRODUCTION

Healthcare-associated infections or nosocomial infections are infections that are not present or incubating during admission to a hospital [1]. This infection may appear 48 hours or more after receiving healthcare [2]. There are different types of nosocomial infections CLABSI, SSI, CAUTI, and VAP. Central line-associated bloodstream infections (CLABSI) are infections caused by bacteria from the skin surface migrating along the external surface of the catheter into the intravascular space [3]. Surgery site infections (SSI) are infections that are acquired after surgery, these infections occur at the incisions or deep inside the tissue at the operation site.

This infection is caused by bacteria found on the skin surface contaminating the incision site [4]. Catheter-associated urinary tract infection (CAUTI) is a urinary tract infection caused by the use of a contaminated urinary catheter. Furthermore, the use of a urinary catheter increases the chance of an infection, this is because it provides a channel from the external environment directly to the bladder [5]. Ventilator-associated pneumonia (VAP) is an infection that develops from the use of motorized automatic ventilation and tracheal intubation. Mechanical ventilation impairs coughing and secretion clearance, this decreases the ability to expel bacteria from the airways increasing bacterial load. Tracheal intubation promotes biofilm formation, these biofilms hold infective microorganisms which can enter the airways and cause VAP [6].

Nosocomial infections increase the length of stay, [7] show that the mean length of stay of a patient who is not infected is 8.7 days while a patient who is infected is 20.3 days. Furthermore [7] also shows that the cost of a patient without nosocomial infection is $2037.30 less than a patient who is infected. Another study estimated the additional length of stay for a patient with nosocomial infection to be between 12.0 and 12.3 days [8]. Nosocomial infections increase mortality, a study in Japan shows the in-hospital mortality of patients with nosocomial infection to be 19.6% and patients with community-acquired infection to be 9.8% [9].

Transmission of pathogens causing nosocomial infections occurs mainly through direct contact [10]. Direct contact happens when healthcare workers touch a colonized patient or objects in their environment. After touching contaminated surfaces, the healthcare worker's hand could be colonized thus risking spreading these pathogens to other patients or medical equipment if proper hygiene is not carried out [11], [12]. Hence reducing contact between patient and healthcare worker is one solution that could be implemented to reduce transmission of pathogens. To reduce contact a service robot could be used for simple tasks such as delivering food or medicine.

1.1. Related Works

In the past there have been several attempts to develop robots for delivery, one example is the "Human-Robot Communication System for an Isolated Environment" [13]. This robot is designed to be used in an office environment to deliver documents between workers. It utilizes ROS 1 as the framework for its navigation, an Android app to control, and MQTT as the communication protocol between the app and ROS. A notable limitation of this system is the Android app used to control the robot. This requires the user to own an Android-based mobile phone, which is not universally accessible. Potential users may have devices that run other operating systems, such as IOS, which makes the system less inclusive. This limitation presents challenges when integrating this system into existing setups. Another research simulated TurtleBot to deliver items for a smart hospital using gazebo simulation [14]. In this project, the author created a robot that can receive tasks from a GUI, the robot's operation is managed using a state machine that will determine the behavior of the robot. The behavior of the robot could be waiting for goal, navigating to goal, and executing goal. The author created a simulation to model a hospital and place the turtlebot3 at random locations throughout the map. A limitation of this system lies in its waiting procedure, which operates based on a preset timer rather than an external input from the patients. As a result, there might be a situation where the patient might take longer to retrieve items from the robot than the wait time, potentially causing the robot to move away while they are still trying to retrieve items. Another example of a mobile robot designed for a hospital is present in a study [15]. This study is aimed to analyze the usability and effectiveness of mobile robots in a simulated hospital environment. The study is done with 30 experienced nurses executing 2 different scenarios. Interviews are done afterward to assess the usability, acceptability, and improvements of the robot. The study concluded with comments from the nurses. The nurses recognize that the robot is highly useable and useful in the healthcare setting for tasks like remote supply delivery and medication distribution. The robot used in this study is the TEMI. This robot has several functionalities: voice control, video call, autonomous navigation, and an app. A limitation to the TEMI robot is that it requires an account and subscription to access features like longer video calling and SDKs. This limitation presents challenges when trying to develop new features for the robot as access to SDK is restricted behind a subscription plan. Furthermore, video calls without the subscription plan are limited to 60 seconds per call. Another example is the "Development of Tele-Operated Mobile Robots for COVID-19 Field Hospitals" [16]. This robot is designed to be used in a hospital for item delivery to reduce contact between healthcare workers and patients. The robot is remotely operated through an internet connection and can teleconference. However, the limitation of this system is that it requires manual operation by a nurse, which could divert valuable nursing resources away from more critical patient care tasks. Furthermore, because the robot is controlled through the internet it requires a stable internet connection. When the robot is controlled, it can result in a time delay that could impact its safety.

1.2. Service Robots

Service robots are robots that could be used in a professional or personal setting, these robots can complete tasks that are useful to a human or equipment. Service robots must have partial autonomy or full autonomy. Full autonomy is where they can perform task such as locomotion, manipulation, or positioning based on their inputs without any human aid, meanwhile, Partial autonomy is where they can perform tasks with some human interaction [17], [18]. Previous application of service robot is hospitality robot that are able to converse and provide help to customers. Further development utilizes the service robot for telepresence and even emotional support [19]. Hence, the medicine delivery service robot must be able to navigate autonomously, avoid obstacles, and provide support when needed.

1.3. Novelty

There are 2 novel ideas presented in this research. The first one is the design of a navigation system that can manage multiple goals. This navigation system will allow the robot to move to multiple goals consecutively. The second is the design of an app that runs on multiple operating systems that can control the robot and communicate. The designed app will be able to send goals, cancel navigation, and track the status of the robot. This app will also need to include a feature that provides communication between the patient and the operator.

Autonomous navigation requires several steps which are mapping, path planning and path following. Mapping uses an algorithm called SLAM. SLAM uses sensor data such as LiDAR and odometer to detect landmarks such as walls or other obstacles and places them in a map. Path planning uses the generated map to plan a path from the robot position to the destination. The path that is created takes into account the shortest distance and any possible collision. Path following moves the robot towards the destination by following the path created. During this step any dynamic objects that is in the way of the robot will be detected and avoided [20], [21].

2. METHOD

Fig. 1 shows the block diagram of the robot. The robot comprises of 3 systems that is the turtlebot3, PC, firebase, and the web app. Fig. 1 shows that the turtlebot3 is made up of IMU, opencr, dynamixel motors, LiDAR, and Raspberry PI. The Raspberry Pi is connected to the PC through ROS communications. The PC will run the main navigation stack. The PC will also connect to Firebase through HTTP. Firebase is a database with a "NoSQL" data structure that can process data in real-time commonly used in IoT projects [22]. In this project, Firebase is used as an intermediate between ROS and the web app. The web app will be connected to Firebase through HTTP. This app will send commands to Firebase that will be translated by ROS to command the navigation stack.

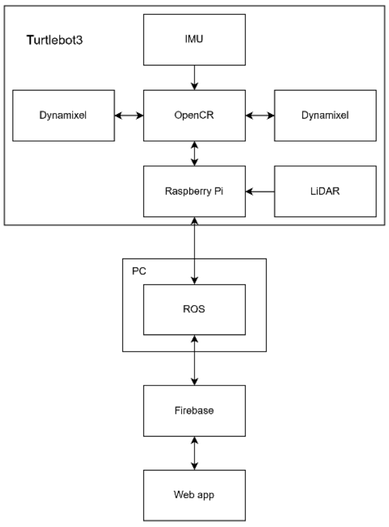


Fig. 1. Block diagram

Fig 2. Shows the environment where the testing of the robot and web app will take place. The arrows represent the Home and goal position and orientation. Areas marked by green are obstacles, red are doors, and blue are windows. The experiments are done with no dynamic obstacles blocking the path.

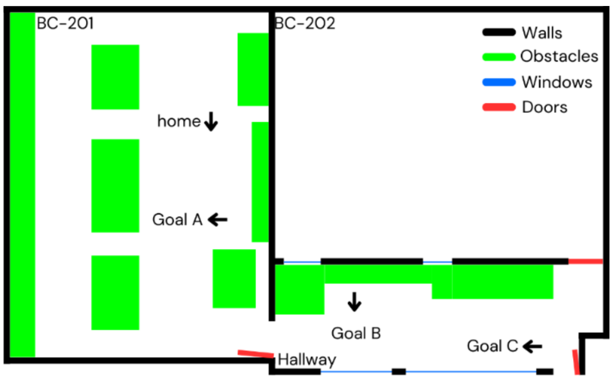


Fig. 2. Testing environment

2.1. Robot Platform

Based on the aspects mentioned before this section the ROBOTIS Turtlebot3 is chosen as the Robot platform. The Turtlebot3 is the third iteration of TurtleBot which is created to teach the user about programming robots and the Robot operating system. Fig. 3 shows that the TurtleBot3 kit includes LiDAR, IMU, 2 Dynamixel motors, OpenCR, and a Raspberry Pi [23], [24]. The Turtlebot3 waffle pi has a carrying capacity of 20 kg, with a maximum translational and rotational velocity of 0.26 m/s and 1.82 rad/s respectively. The included Raspberry Pi 4 (1.8 GHz Quad core Cortex-A72, 4GB RAM, Ubuntu 22.04) connects to a PC using Robot Operating System (ROS) communication methods to relay LiDAR data and the sensor data from OpenCR 1.0. The Raspberry Pi also receives velocity commands and converts them to the individual angular velocity for the 2 motors before sending these commands to the OpenCR 1.0. The OpenCR 1.0 board is the main controller for the Dynamixel motor, this board is connected to the Raspberry Pi via USB serial. The OpenCR 1.0 board includes an Inertial measurement unit (ICM-20648) which is connected to the OpenCR 1.0 by SPI. The ICM-20648 is a 3-axis gyroscope, 3-axis accelerometer, and a digital motion sensor. The embedded gyroscope has a programmable full-scale range of ±250, 500, 1000, and 2000 degrees/s, and the accelerometer has a programmable full-scale range of ±2, 4, 8, and 16 g. The included Dynamixel XM430-W210 connects to the OpenCR board using TTL serial. This motor contains a contactless absolute encoder and an ARM CORTEX-M3 MCU which allows the angular velocity of the motor to be controlled. The Dynamixel XM430-W210 has a stall torque of 3.0 Nm at 12 V, 2.3 A, and a no-load speed of 77 rev/min. The included LiDAR connects to the Raspberry Pi through USB serial and has a detection distance of 160 – 8000 mm with an accuracy of ±10 mm for distances of 160 – 300 mm, ±3.0% for distances of 300 – 6000 mm, and ±5.0% for distances of 6000 – 8000 mm.

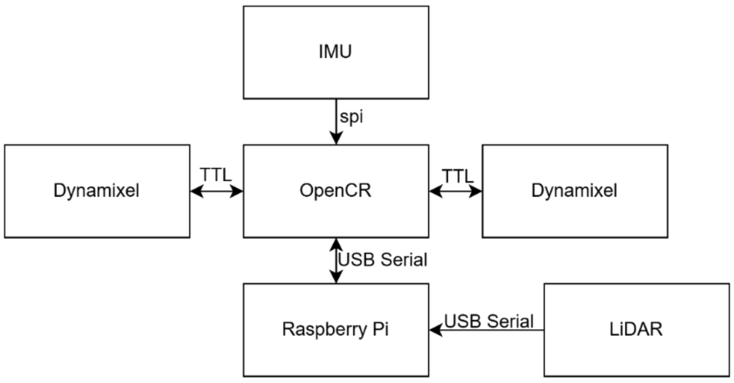


Fig. 3. Turtlebot3 Waffle Pi

2.2. Software Setup

Robot operating system, ROS, is a collection of software, libraries, algorithms, and tools to make robot development easier. ROS 2 is the second version of ROS that comes with improved security and performance. The goal of using ROS is to break up big chunks of code into smaller specific nodes to reduce code complexity and add fault isolation if an error occurs. For these nodes to function as one program, ROS gives them a way to communicate with 3 main methods which are topics, services, and actions [25], [26]. In the topic communication method, one node will be a publisher and another will be a subscriber. The publisher will publish a message to a certain topic and the subscriber will subscribe to this topic and receive the message. In the service communication method, one node will be a server and another will be a client. In this method, the client will need to send a request to the server for the server to send a message. The action method consists of 2 services and 1 topic. In the action method one node will be the server and the other will be the client. The action method provides a feedback topic to monitor the progress of the process and the ability to cancel the process [25], [26].

Fig. 4 shows a diagram of all the nodes that are utilized and their relationships. The "Turtlebot_node" is the node running on the Raspberry Pi. This node publishes /scan, /odom, /tf, and /battery_state topics. This node subscribes to the /cmd_vel topic, this topic controls the linear and angular velocity of the robot. The /scan topic contains the data from the LiDAR. The /odom topic contains the odometry data, odometry is the estimated position and orientation of the robot based on its motion. The /tf topic contains transformation data between the coordinate frames of sensors and motors to the base coordinate frame of the robot. These 2 topics will be subscribed by the navigation stack to position the robot within the map. The /battery_state topic contains the battery percentage of the robot, this topic will be subscribed by the "battery_state_node". The "battery_state_node" will store the battery percentage in JSON format and send it to Firebase using the http/2 protocol. The "firebase_node" is the main bridge between firebase and the robot, this node takes the list of goals, cancel signal, and go signal from firebase and publishes them to /checklist_goal, /cancel_, and /go topics respectively. The /checklist_goal topic contains the list of destinations for the robot. The /cancel_ topic contains data to stop the robot's current navigation, if the data is "1" the robot will stop navigating and if the data is "0" the robot will continue navigating. The /go contains data to signal the robot to move to the next destination, if the data is "1" the robot will go to the next destination in the list, and if the data is "0" the robot will wait until it is "1". This node also subscribes to /navi and /c_dest topics and sends them to Firebase. The /navi topic contains data of the navigation state of the robot, if the robot is currently navigating the data would be "1" if the robot is not navigating it would be "0". The /c_dest topic contains data of the current destination of the robot, if the robot is currently going to Goal A the data would be "A" if the robot is going to Home the data would be "HOME". The "Simple_commander_node" uses the "simple commander API" provided by Navigation 2 to create an action client to interact with the "basic_navigator" action server within the navigation stack. This node will send goals one at a time to the action server and wait until "success" feedback is received. If a "failed" is received all the goals in the list will be discarded and the node is ready to receive new goals. This node publishes the /navi topic and /c_dest topic. The distance_measure node subscribes to /odom and /navi topic. If the data from the /navi topic is "1" this node will start counting the distance traveled by the robot using equation (1). Equation (1) shows the distance calculation equation where t represents the time step, D represents the total distance traveled, x_t represents the current x coordinate from the /odom topic while, x_{t-1} represents the previous x coordinate and y_i represents the current y coordinate from the /odom topic while, y_{i-1} represent the previous y coordinate. The distance traveled is the sum of the Euclidean distances of points from the odometry while the robot is navigating. This calculation is done 40 times a second, which corresponds to the publish rate of the /odom topic.

$$D = \sum_{t=2}^{n} \sqrt{(x_t - x_{t-1})^2 + (y_t - y_{t-1})^2} \tag{1}$$

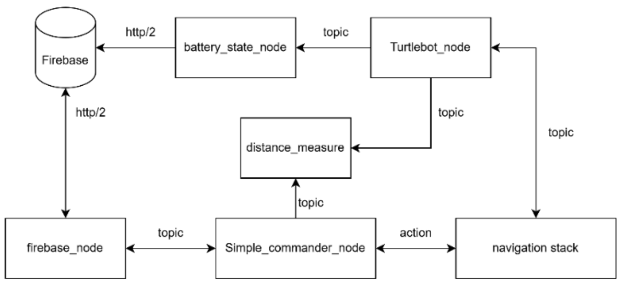


Fig. 4. Node diagram

2.3. Simultaneous Localization and Mapping

Simultaneous localization and mapping (SLAM) is a method of building a map of an unknown environment while estimating the position of the robot [27]. The library used in this process is cartographer.

Cartographer is a real-time SLAM algorithm developed by Google. Cartographer is designed to be used with a modest hardware requirement because it does not use a particle filter. Without the particle filter, the cartographer algorithm does pose estimation regularly to minimize the accumulated error. Cartographer uses LiDAR scan data which will be divided into submaps. After submaps have been created it undergoes scan matching against other recent submaps to estimate the robot's position. Scan matching is also done to submaps to align them to a master map.

Before mapping the robot is placed parallel to a wall, this ensures the final map is straight, not at an angle, making it easier to modify. During mapping the robot is manually operated using the teleop code provided by turtlebot3. The robot is driven around the testing environment at a maximum of 0.1 m/s to maintain the accuracy of the resulting map. After mapping the map is cleaned up using an image manipulation program. Occupied cells that are not present in the real environment are changed to free cells and free cells that are not present in the real environment are changed to occupied.

2.4. Navigation Setup

To create the navigation system, the Navigation 2 (Nav2) library will be used. Nav2 includes action servers for local planning, global planning, localizing, recoveries, and more [28]. To manage the behavior of the navigation system a behavior tree (BT) is utilized. To manage the navigation stack the BT will connect to the action servers inside the stack and control them. Fig. 5 shows the BT used, when using this BT, the robot will continuously loop to compute a path to the goal and follow it. If a failure occurs a contextual recovery will be executed. Only when contextual recovery efforts fail the main recovery branch will be executed [28], [29], [30]. The localizer used is the Adaptive Monte Carlo localizer (AMCL), this localizer uses /odom topic and /scan topic to estimate the position of the robot within the premade map [31]. During navigation the global planner will create a path to the goal position based on prior knowledge provided by the global costmap created during SLAM, this path will then be followed by the local planner [32]. The local planner will not use the map from SLAM but will create its local costmap based on the current /scan topic, thus the local planner can plan around dynamic objects that may not be present during mapping [32]. The global planner used in this project is the "Navfn" planner. This planner uses the A-STAR path-finding algorithm. The A-STAR algorithm finds the shortest path between the initial position to the goal position [33], [34]. The algorithm will start with the initial position as the current node and the goal position as the goal node. Next, the algorithm will find neighboring nodes that have a small cost calculated using the sum of two functions, the first function represents the distance of the neighboring node to the current node and the second represents the distance of the neighboring node to the goal node. The neighboring node with the least cost will be the next current node, these steps will continue until the current node is the goal node. The result will be a set of nodes from the starting position to the goal position. The local planner used in this project is the "model predictive path integral" (MPPI) controller. This local planner is compatible with differential drive, omnidirectional, and Ackermann motion models. The MPPI controller algorithm finds the best path through an iterative approach. It takes the best path from the previous time step, applies noise from a Gaussian distribution, and scores the generated set of paths using several critic plugins to find the best path [35]. A total of 2000 paths will be generated at a rate of 30 Hz. 11 critic plugins are available, but only 9 will be used: Constraint critic, Cost critic, Goal critic, Goal Angle critic, Path Align critic, Path Follow critic, Path Angle critic, Prefer Forward critic and Twirling critic. Each of these critics has a different purpose to create an optimal path. The constraint critic ensures that the control velocities of the path are within the kinematic and dynamics limit of the robot. The cost critic ensures the path is far away from obstacles and critical collision by checking its footprint and costmap value. The goal critic creates a path that navigates towards the goal within a threshold distance from the goal. The goal angle critic ensures that the robot achieves the final goal angle within a threshold distance from the goal. The path align critic creates a path that that aligns with the global path. The path-follow critic incentivizes the robot to make progress along the path. The path angle critic ensures that the trajectories created have a low relative angle to the path. Prefer forward critic favors trajectories that move the robot forward. Twirling critic ensures that the robot does not do unnecessary twisting. The recovery action server will execute the recovery actions. The Nav 2 library includes four recovery actions: wait, back up, spin, and drive on heading [36]. The costmap used in Navigation 2 is called the layered costmap. A layered costmap uses multiple maps to create one master costmap. These costmap layers include the static map, inflation map, obstacle map, and more. This enables easier fine-tuning of the costmap and also makes it easier to add extra maps and filters to the costmap [37].

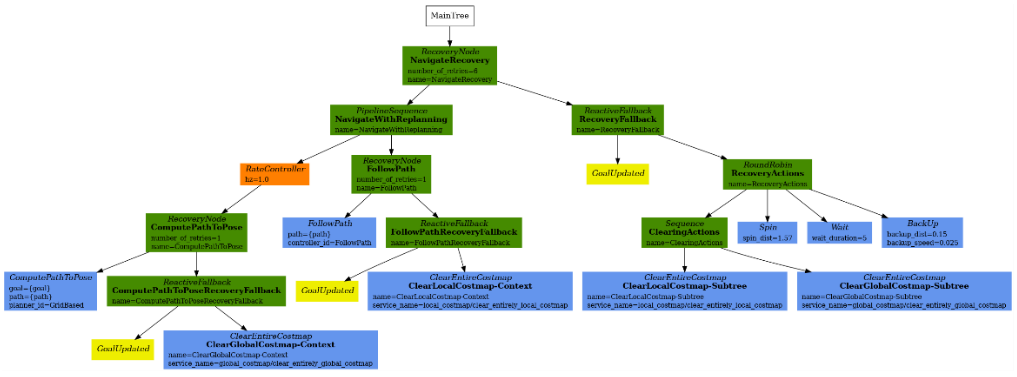

MainTree
RecoveryNode NavigateRecovery number_of_retries=6 name=NavigateRecovery
PipelineSequence NavigateWithReplanning name=NavigateWithReplanning
ReactiveFallback RecoveryFallback name=RecoveryFallback
RateController hz=1.0
RecoveryNode FollowPath number_of_retries=1 name=FollowPath
GoalUpdated
RoundRobin RecoveryActions name=RecoveryActions
RecoveryNode ComputePathToPose number_of_retries=1 name=ComputePathToPose
FollowPath path={path} controller_id=FollowPath
ReactiveFallback FollowPathRecoveryFallback name=FollowPathRecoveryFallback
Sequence ClearingActions name=ClearingActions
Spin spin_dist=1.57
Wait wait_duration=5
BackUp backup_dist=0.15 backup_speed=0.025
ComputePathToPose goal={goal} path={path} planner_id=GridBased
ReactiveFallback ComputePathToPoseRecoveryFallback name=ComputePathToPoseRecoveryFallback
GoalUpdated
ClearEntireCostmap ClearLocalCostmap-Context name=ClearLocalCostmap-Context service_name=local_costmap/clear_entirely_local_costmap
ClearEntireCostmap ClearLocalCostmap-Subtree name=ClearLocalCostmap-Subtree service_name=local_costmap/clear_entirely_local_costmap
ClearEntireCostmap ClearGlobalCostmap-Subtree name=ClearGlobalCostmap-Subtree service_name=global_costmap/clear_entirely_global_costmap
GoalUpdated
ClearEntireCostmap ClearGlobalCostmap-Context name=ClearGlobalCostmap-Context service_name=global_costmap/clear_entirely_global_costmap

Fig. 5. Behavior tree [38]

2.5. Web Apps

For patients and hospital staff to interact with the robot 3 web application has been developed. The application is created by using HTML, Node.JS, and its packages. Node.JS is a programming language mainly used by web developers, it features a package manager, NPM, that has millions of packages [39]. Node.js is used as the backend that will bridge the web app with Firebase.

Fig. 6 shows the patient app main screen and the video call screen. The app will be on the main screen by default and will only switch to the video call screen when the help button is pressed. The main screen consists of 2 buttons labeled accept and help respectively. The accept button is pressed after the patient has taken their item out of the robot. This button will send the robot to the next location in the list of goals. The help button is pressed if the patient requires help with their item. When pressed the app will switch to the video call screen, in this screen the app will automatically connect to an operator through a video call. After the conversation is done the patient will press the done button, to go back to the main screen, and then press the accept button.

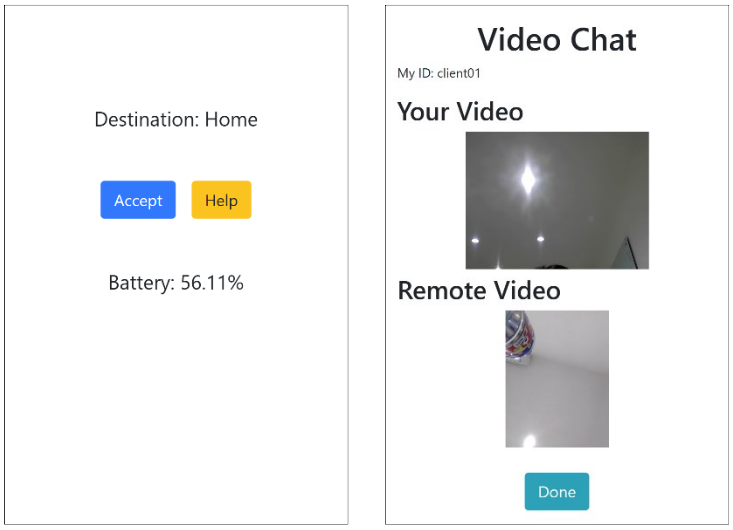
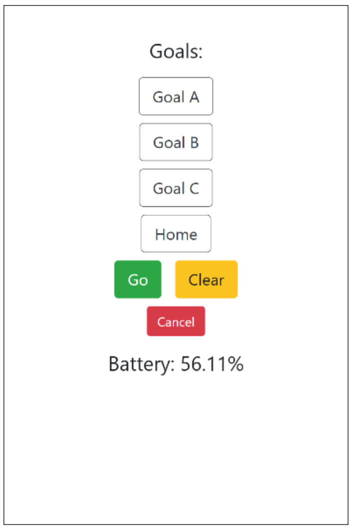


Fig. 6. Main screen (left), video call screen (right)

Fig. 7 shows the control app and the video call app. The control app and video call app will be opened at the same time in different browser tabs by the operator. The operator will use the control app to create a list of goals by pressing the desired location's button, the list will be displayed next to the "Goals:" text. After the list has been created the operator will press the Go button and the robot will immediately navigate to the first goal on the list. For example, if the operator wants the robot to go to Goal A then Goal C then Goal B, the operator would press the button labelled Goal A, Goal C, Goal B, and Go in this order. The clear button will clear the goal list if a mistake happens during the creation. This button does not clear the list that has been sent to the robot, to do this the cancel button could be pressed. If the cancel button is pressed the robot will stop navigating, clear the received list, and will be ready to receive a new goal list.

The video call app will play a sound and display a notification after the patient presses the help button. This notification will not be cleared until the operator acknowledges it and accepts the patient's video call. After accepting both video calls app will be connected. The cancel button will end the video call redirecting the patient ap to the main screen.



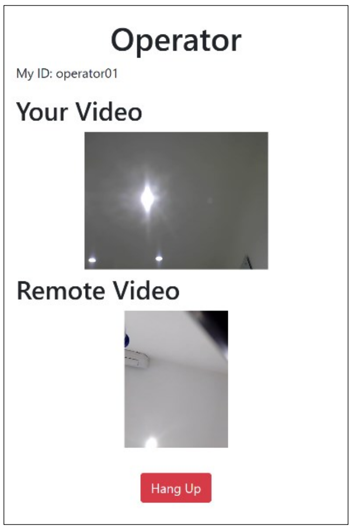


Fig. 7. Control app (left), Video call screen (right)

2.6. Flowchart

Fig. 8 shows the flowchart for the system. It starts with the operator using the control app to choose the goal positions, then sending the goal list to Firebase by pressing go. When the list reaches firebase the firebase_node will fetch the goal list and send it to the simple_commander_node then the robot will start moving to the destination. After reaching the destination the simple_commander_node will check if there are more destinations left in the list if there are not the node resets and becomes ready to receive a new goal list. If there are more the robot will go to a waiting state where the help button and accept button will be continuously checked if they have been pressed. If the help button is pressed a video call will be established until the call is done. If the accept button is not pressed the robot will continue being in the waiting state indefinitely. If the accept button is pressed the robot will navigate to the next goal in the list and will continue to loop until all goals have been reached or cancelled.

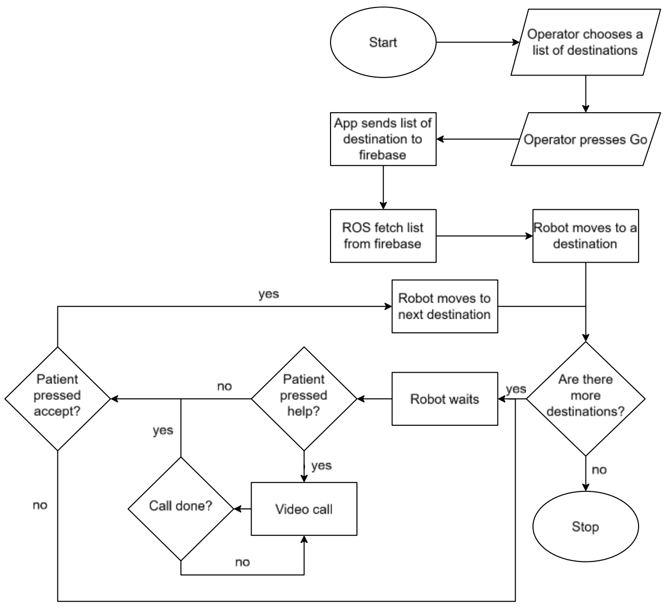

Start
Operator chooses a list of destinations
Operator presses Go
App sends list of destination to firebase
ROS fetch list from firebase
Robot moves to a destination
Are there more destinations?
no
Stop
yes
Robot waits
Patient pressed help?
yes
Video call
Call done?
no
yes
no
Patient pressed accept?
yes
Robot moves to next destination
no

Fig. 8. Flowchart of delivery robot

2.7. Experimental Setup

To evaluate the performance of the apps and the navigation, experiment scenarios are created. For each scenario, the accept button will be pressed to advance to the next goal. The average speed, average distance to target and success state will be recorded for each scenario. The average speed will be calculated by using the sum of the distance traveled in meters and the sum of the navigation time in seconds. The average distance will be calculated between the robot's final base_link coordinates in the map_frame and the target coordinates sent to the navigation stack using the Euclidean distance equation found in (2). If a failure happens during the scenario the experiment will be stopped and the success state will be FALSE otherwise it will be TRUE. The scenario is as follows:

1. The robot will start at Home. Goal B will be pressed, and then Go will be pressed.
2. The robot will start at Goal C. Home, Goal B and Goal C will be pressed, and then the Go will be pressed.
3. The robot will start at Home. Goal A, Goal B, Goal C, and Home will be pressed, and then Go will be pressed.
4. The robot will start at Home. Goal A and Home will be pressed, and then Go will be pressed.

$$d = \sqrt{\left(x_{target} - x_{final}\right)^2 + \left(y_{target} - y_{final}\right)^2} \tag{2}$$

3. RESULTS AND ANALYSIS

3.1. Dimension and Components

When determining the dimensions of the robot several aspects that are to be considered are the distance of the patient's hand to reach the item compartment and the distance of the tablet to the patient, to maintain accessibility and readability respectively [40]. Fig. 9 shows the dimensions and components of the delivery robot. The robot has a dimension of 40 cm×40 cm×100 cm. This robot is equipped with various components, including LiDAR for navigation, a tablet holder for the tablet, an item compartment to store items, a TurtleBot3 base, and an AC power plug.

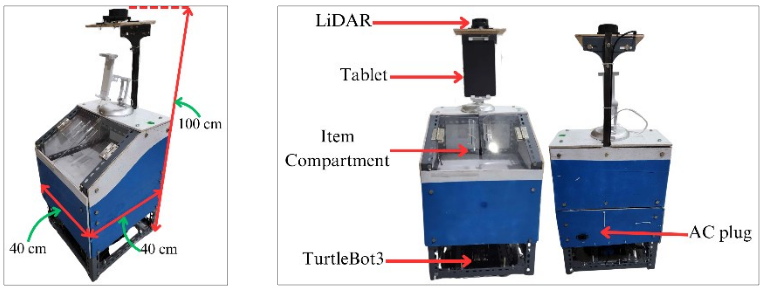


Fig. 9. Dimension of the delivery robot (left), Components of the delivery robot (right)

3.2. Costmap Parameter Tuning

Before testing 2 parameters need to be tweaked to ensure that the robot wouldn't hit obstacles while navigating. These 2 parameters are inflation radius and cost scaling factor, both parameters work together to set how far from an obstacle the robot will navigate and they will also affect how wide a gap the robot can pass through. Inflation is an area around obstacles that has a cost value of 0 – 255, how far this area extends from an obstacle is determined by the inflation radius parameter. The cost value inside this area is determined by a decay curve. The area that is closer to the obstacles has a higher cost than areas that are further away. A cost value of 0 means that the area is clear and a cost of 255 means that if the robot passes through this area a collision will happen. Fig. 10 shows the effect of the inflation radius parameter on the size of the inflation zone, as the parameter increases the size of the inflation zone also increases. The areas marked with red are high-cost areas and dark blue are low-cost areas [41].

The cost scaling factor controls the steepness of the decay curve on the inflation radius. From Fig. 11 areas of low cost are colored dark blue while high-cost areas are colored red. The cost scaling factor will change how far the red area expands from the obstacles. The smaller the value the farther the high-cost area will expand the higher the value the closer the high-cost area will expand; this effect can be seen in Fig. 11. This parameter will control how far from the obstacles the robot will move as the robot will favor lower cost dark blue areas then high-cost red areas [41].

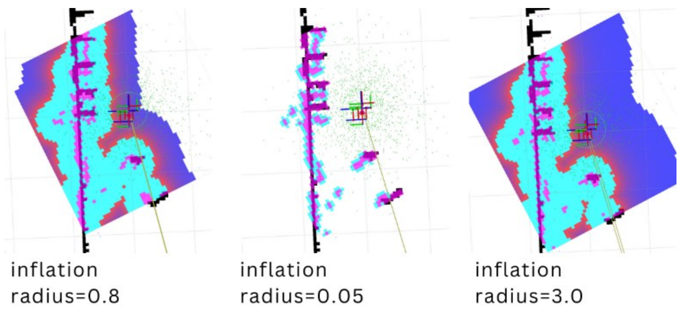


Fig. 10. Inflation radius parameter

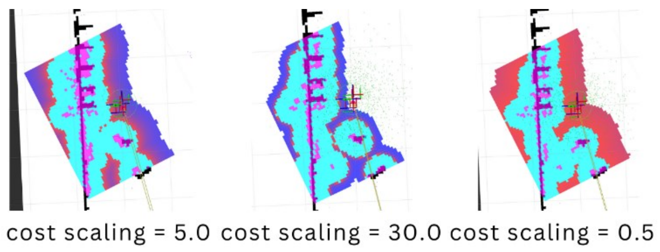


Fig. 11. Cost scaling parameter

For the robot, the optimal inflation radius and cost scaling factor is found to be 0.9 and 7.0 for the local costmap and 0.5 and 2.0 for the global costmap. With the specified parameters, the robot could navigate with fewer recovery behaviors being triggered.

3.3. Problems Encountered and Solutions

During testing, several problems were found. The first problem is during mapping and navigation the windows located in the hallway are not detected by the LiDAR, this causes the robot to not detect it as an obstacle and also causes unrecoverable localization errors. To get around this problem the windows are covered with a layer of masking tape as shown in Fig. 12.

Fig. 12. Taped window

The second problem is the appearance of fake obstacles in the costmap. The fake obstacles would block global and local planning causing the navigation to fail. To solve this a costmap filter and laser speckle filter are implemented in the navigation stack. The laser speckle filter detects outliers in the LiDAR data and removes them [42]. The filter subscribes to the /scan topic to access the LiDAR data and publishes the filtered result to /scan_filtered. Fig. 13 shows a diagram of how the laser speckle filter works, the blue rectangle represents a wall, the orange square represents an obstacle, the black circle represents the LiDAR, the red dots represent the LiDAR points and the green dot represent an anomalous LiDAR point. This filter measures the distance of a point to neighboring points. If the distance measured is larger than the threshold the point will be removed. The speckle filter has 4 parameters filter type max range, max range difference and filter window. The filter type parameter determines the type of filtering used, range-based filtering or Euclidean filtering based on radius outlier search. The max range parameter determines the maximum range value of the lidar data, points that is further than this value will not be considered during filtering. The max range difference parameter determines the maximum distance between consecutive points, values larger than this parameter will be considered as an outlier and will be filtered out. The filter window parameter determines the number of consecutive points to be taken account when detecting outlier [43], [44].

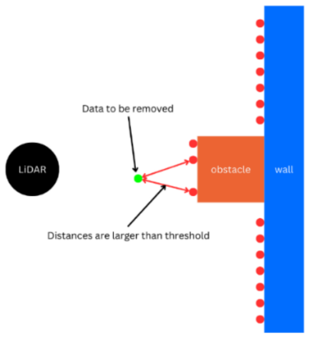


Fig. 13. Diagram of speckle filter

To filter the costmap, the costmap denoise layer from the navigation 2 library is used [45]. The filter will take the local or global costmap as inputs. This filter has 2 parameters which are minimal group size and group connectivity type. The minimal group size parameter determines the minimum number of occupied cells within a group. This parameter also determines the algorithm used in denoising. If the value is 2 an erosion-based function will be used. If the value is more than 2 a group-size filtering will be used. The group connectivity type determines the grouping behavior, this parameter only has 2 values 4 and 8. A value of 4 will group occupied cells that are connected vertically and horizontally. A value of 8 will group occupied cells that are connected diagonally, vertically, and horizontally. Fig. 14 shows the algorithm used in this project, the group-size filtering. This algorithm first groups adjacent occupied cells into one group and counts the number of occupied cells in each group. The groups that have a lesser number of occupied cells will be determined as noise and changed into unoccupied cells.

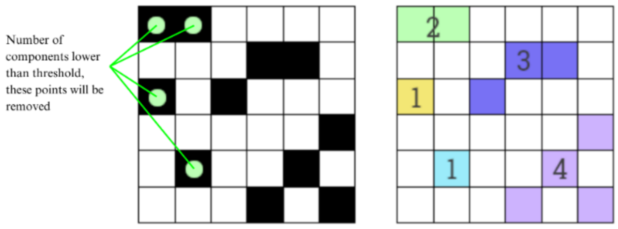

Number of components lower than threshold, these points will be removed
2
3
1
1
4

Fig. 14. Diagram of costmap filter [45]

The third problem found is that because of the position of the LiDAR being too high some lower obstacles could not be detected, to fix this problem keepout zones will be used. Keep out zones are masks that mark an area on the map as obstacles, these work by taking values from the mask and interpreting them into cost on the master cost map [37], [46]. As a result, the robot will navigate away from these marked areas. Fig. 15 shows the original map and the keep out zone marked with red. To create the keepout zone the original map is modified by marking the area black.

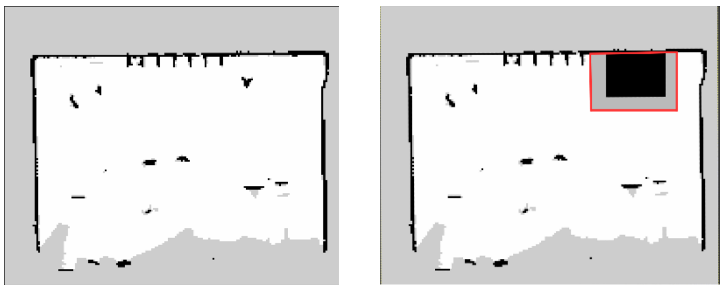


Fig. 15. Original map (left), Altered map with keep out zones (right)

However, during testing, it was found that the robot would navigate right against the keepout zone and get stuck inside of it. This is due to an error in localization which causes the estimated position to shift around during navigation and occasionally the estimated position will be inside of the keepout zone causing the robot to stop moving. To prevent this from happening the mask for the keepout zone is modified by coloring a small area around the keepout zone grey. The robot will navigate around this grey area however if the estimated position shifts inside this area the robot will still be allowed to move, essentially creating an inflation area around the keepout zones.

3.4. Result of Scenario 1

Scenario 1 contains 2 navigation sections, Home to Goal B and Goal B to Home. Table 1 shows the result of scenario 1. The average speed of this scenario is 0.165 m/s with a standard deviation of 0.0227 m/s. The average distance to the target is 0.050 m with a standard deviation of 0.0160 m. The success rate is 90% from the 10 runs.

Table 1. Results of scenario 1.

No.	Average speed (m/s)	Average distance to target (m)	Succes state
1	0.169	0.061	TRUE
2	0.179	0.077	TRUE
3	0.179	0.055	TRUE
4	0.177	0.058	TRUE
5	0.183	0.046	TRUE
6	0.135	0.024	TRUE
7	-	-	FALSE
8	0.172	0.057	TRUE
9	0.118	0.035	TRUE
10	0.176	0.038	TRUE

Experiments 2, 5, and 10 exhibited no abnormal behaviors such as low downs or recovery actions. Experiments 1, 3, 4, 6, 8, and 9 experienced slowdowns when moving from Home to Goal B due to localization error. In experiment 6 the localization error causes recovery behaviors to be triggered. Experiment 9 experienced a major localization error but no recovery behavior was triggered. Experiments 1, 6, and 8 exhibited slowdowns when approaching the Home position, likely due to the robot's attempt to navigate around the designated keep-out zone. Experiments 6 and 9 exhibited recovery behavior due to localization error when going from Goal B to Home. Experiment 7 Failed when going from Home to Goal B due to an unrecoverable localization error.

3.5. Result of Scenario 2

Scenario 2 contains 3 navigation sections, Goal C to Home, Home to Goal B and Goal B to Goal C. Table 2 shows the result of Scenario 2. The average speed of this scenario is 0.172 m/s with a standard deviation of 0.0114 m/s. The average distance to the target is 0.046 m with a standard deviation of 0.0197 m. The success rate is 100% from the 10 runs.

Table 2. Speed of navigation sections in scenario 2.

No.	Average speed (m/s)	Average distance to target (m)	Success state
1	0.172	0.057	TRUE
2	0.176	0.035	TRUE
3	0.173	0.029	TRUE
4	0.180	0.093	TRUE
5	0.169	0.023	TRUE
6	0.145	0.040	TRUE
7	0.182	0.050	TRUE
8	0.163	0.054	TRUE
9	0.179	0.040	TRUE
10	0.184	0.037	TRUE

Experiments 4, 7, 9, and 10 exhibited no abnormal behaviors. Experiments 1 and 2 demonstrated recovery behavior when moving through the door from Goal C to Home, attributed to localization errors encountered near the door. Experiments 3, 6, and 7 experienced recovery behavior when passing through the door going from Home to Goal B due to localization errors during the approach to the door. Experiments 1, 5, and 8 experienced a slowdown when going from Home to Goal B due to localization error.

3.6. Result of Scenario 3

Scenario 3 contains 4 navigation sections, Home to Goal A, Goal A to Goal B, Goal B to Goal C and Goal C to Home. Table 3 shows the result of scenario 3. The average speed of this scenario is 0.167 m/s with a standard deviation of 0.0144 m/s. The average distance to the target is 0.055 m with a standard deviation of 0.0109 m. The success rate is 90% from the 10 runs.

Table 3. Speed of navigation sections in scenario 3.

No.	Average speed (m/s)	Average distance to target (m)	Success state
1	0.153	0.043	TRUE
2	0.178	0.073	TRUE
3	0.177	0.064	TRUE
4	0.162	0.061	TRUE
5	0.176	0.037	TRUE
6	0.167	0.052	TRUE
7	0.138	0.054	TRUE
8	-	-	FALSE
9	0.182	0.061	TRUE
10	0.175	0.051	TRUE

Experiments 2, 3, 8, 9, and 10 exhibited no abnormal behaviors. Experiments 4 and 5 experienced a slowdown when going from Goal A to Goal B due to localization error. Experiments 1 and 7 experienced a recovery behavior when going from Goal A to Goal B due to localization error. Experiment 6 experienced a recovery behavior when going from Goal C to Home due to localization error. Experiment 8 experienced a localization error that it could not recover from.

3.7. Result of Scenario 4

Scenario 4 contains 2 navigation sections, Home to Goal A and Goal A to Home. Table 4 shows the result of scenario 4. The average speed of this scenario is 0.161 m/s with a standard deviation of 0.0034 m/s. The average distance to the target is 0.084 m with a standard deviation of 0.0323 m. The success rate is 100% from the 10 runs.

Table 4. Speed of navigation sections in scenario 4.

No.	Average speed (m/s)	Average distance to target (m)	Success state
1	0.161	0.093	TRUE
2	0.157	0.148	TRUE
3	0.162	0.041	TRUE
4	0.160	0.114	TRUE
5	0.153	0.082	TRUE
6	0.163	0.060	TRUE
7	0.165	0.101	TRUE
8	0.161	0.053	TRUE
9	0.165	0.087	TRUE
10	0.161	0.061	TRUE

In this scenario, the robot travels faster when moving from Home to Goal A than when it is moving from Goal A to Home, this is because when going to Goal A the robot travels in a straight line meanwhile when going back to Home the robot must turn 90 degrees towards Goal A and has to 180 degrees after reaching Home.

3.8. Average Distance and Time

The average distance, time, distance to target, and success percentage of 10 tries of each scenario have been calculated in Table 5. From the data in Table 5 a scatter plot of average distance vs average time is plotted in Fig. 16. The blue points represent the value for each scenario and the red line is the best fit line. The best fit line has an R^2 value of 0.997 indicating that the data fits almost perfectly on the line and a gradient of 0.17 m/s. This shows that as the distance increases the time would also increase linearly.

Table 5. Average distance and time of scenario.

Scenario	Start to goals	Average distance (m)	Average time (s)	Average distance to target (m)	Success rate
Scenario 1	Home, Goal B, Home	19.00	118.07	0.0501	90%
Scenario 2	Goal C, Home, Goal B, Goal C	26.04	151.93	0.0457	100%
Scenario 3	Home, Goal A, Goal B, Goal C, Home	26.49	159.41	0.0551	90%
Scenario 4	Home, Goal A, Home	5.09	31.67	0.0840	100%

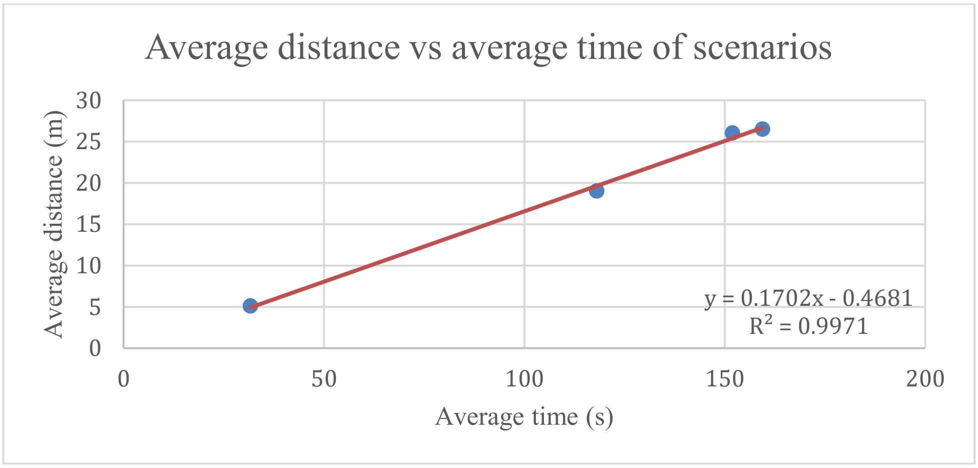


Fig. 16. Plot of Average distance vs Average time

4. CONCLUSION

In conclusion, the research objectives stated have been achieved. The navigation system can navigate the robot to multiple goals autonomously and wait for a button press before moving on to the next goal. The web app can send a list of goals, cancel the current navigation, and when needed provide help through video call. The final system has an average success rate of 95% with an average speed of 0.17 m/s and an average distance to target of 0.0587 m. The result of the scenario shows that the robot has a localization problem. This problem is caused by the poor odometry. The odometry of this robot is provided by the wheel encoders to measure distance traveled and imu to measure the angular acceleration of the robot. To improve the localization sensor fusion using techniques such as extended Kalman filter could be utilized. Another limitation is the absence of a depth sensor. Obstacles with irregular shapes, such as tables and chairs, are more challenging to detect using LiDAR. While keepout zones can mitigate this limitation, they must be preconfigured and cannot effectively address dynamic objects.

REFERENCES

[1] A. F. Monegro, V. Muppidi, and H. Regunath, *Hospital-Acquired Infections*. 2024. Accessed: Dec. 09, 2024. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/28160957.

[2] M. Haque, M. Sartelli, J. McKimm, and M. Abu Bakar, "Health care-associated infections - an overview.," *Infect Drug Resist*, vol. 11, pp. 2321–2333, 2018, https://doi.org/10.2147/IDR.S177247.

[3] Y. Haddadin, P. Annamaraju, and H. Regunath, *Central Line–Associated Blood Stream Infections*. 2024, https://europepmc.org/article/nbk/nbk430891.

[4] M. Zabaglo, S. W. Leslie, and T. Sharman, *Postoperative Wound Infections*. In StatPearls [Internet]. StatPearls Publishing. 2024, https://www.ncbi.nlm.nih.gov/books/NBK560533/.

[5] G. T. Werneburg, "Catheter-Associated Urinary Tract Infections: Current Challenges and Future Prospects.," *Res Rep Urol*, vol. 14, pp. 109–133, 2022, https://doi.org/10.2147/RRU.S273663.

[6] F. Howroyd *et al.*, "Ventilator-associated pneumonia: pathobiological heterogeneity and diagnostic challenges*,"* *Nature communications*, vol. 15, no. 1, p. 6447, 2024, https://doi.org/10.1038/s41467-024-50805-z.

[7] S. Moradi, Z. Najafpour, B. Cheraghian, I. Keliddar, and R. Mombeyni, “The Extra Length of Stay, Costs, and Mortality Associated With Healthcare-Associated Infections: A Case-Control Study.,” *Health Sci Rep*, vol. 7, no. 11, p. e70168, Nov. 2024, https://doi.org/10.1002/hsr2.70168.
[8] H. Arefian *et al.*, “Estimating extra length of stay due to healthcare-associated infections before and after implementation of a hospital-wide infection control program,” *PLoS One*, vol. 14, no. 5, May 2019, https://doi.org/10.1371/journal.pone.0217159.
[9] N. Takahashi *et al.*, “Incidence and mortality of community-acquired and nosocomial infections in Japan: a nationwide medical claims database study,” *BMC Infect Dis*, vol. 24, no. 1, Dec. 2024, https://doi.org/10.1186/s12879-024-09353-6.
[10] A. F. Monegro, V. Muppidi, and H. Regunath, “Hospital-Acq uired Infections.” *Surgical Clinics*, vol. 92, no. 1, pp. 65-77, 2024, https://www.surgical.theclinics.com/article/S0039-6109(11)00151-4/abstract.
[11] A. Facciolà *et al.*, “The role of the hospital environment in the healthcare-associated infections: a general review of the literature.,” *Eur Rev Med Pharmacol Sci*, vol. 23, no. 3, pp. 1266–1278, Feb. 2019, https://doi.org/10.26355/eurrev20190217020.
[12] G. Suleyman, G. Alangaden, and A. C. Bardossy, “The Role of Environmental Contamination in the Transmission of Nosocomial Pathogens and Healthcare-Associated Infections,” *Current infectious disease reports,* vol. 20, pp. 1-11. 2018, https://doi.org/10.1007/s11908-018-0620-2.
[13] I. Diddeniya, I. Wanniarachchi, H. Gunasinghe, C. Premachandra, and H. Kawanaka, “Human-Robot Communication System for an Isolated Environment,” *IEEE Access*, vol. 10, pp. 63258–63269, 2022, https://doi.org/10.1109/ACCESS.2022.3183110.
[14] R. Safin, R. Lavrenov, K. H. Hsia, E. Maslak, N. Schiefermeier-Mach, and E. Magid, “Modelling a TurtleBot3 Based Delivery System for a Smart Hospital in Gazebo,” in *SIBCON 2021 - International Siberian Conference on Control and Communications*, pp. 1-6, May 2021. https://doi.org/10.1109/SIBCON50419.2021.9438875.
[15] H. J. Yoo, E. H. Kim, and H. Lee, “Mobile robots for isolation-room hospital settings: A scenario-based preliminary study,” *Comput Struct Biotechnol J*, vol. 24, pp. 237–246, Dec. 2024, https://doi.org/10.1016/j.csbj.2024.03.001.
[16] S. Vongbunyong, K. Thamrongaphichartkul, N. Worrasittichai, A. Takutruea, and T. Prayongrak, “Development of Tele-Operated Mobile Robots for COVID-19 Field Hospitals,” in *ICSEC 2021 - 25th International Computer Science and Engineering Conference*, pp. 319–324, 2021, https://doi.org/10.1109/ICSEC53205.2021.9684621.
[17] “Service robots.” Accessed: Nov. 14, 2024. [Online]. Available: https://ifr.org/service-robots.
[18] J. Holland *et al.*, “Service robots in the healthcare sector,” *Robotics,* vol. 10, no. 1, p. 47, 2021, https://doi.org/10.3390/robotics10010047.
[19] P. Zhang, J. Jackie Chen, D. Jin, and S. Shawn Jung, “Service robots in crowded environments: How crowd dynamics shape robotic adoption intention at events,” *Journal of Hospitality and Tourism Management*, vol. 61, pp. 251–260, Dec. 2024, https://doi.org/10.1016/j.jhtm.2024.10.005.
[20] Y. Dong, D. Wang, and J. Liu, “Research on Autonomous Robot Navigation Algorithm in Indoor Environment,” in *Frontiers in Artificial Intelligence and Applications*, pp. 281–286, 2023, https://doi.org/10.3233/FAIA230820.
[21] K. Ozdemir and A. Tuncer, “Navigation of autonomous mobile robots in dynamic unknown environments based on dueling double deep q networks,” *Eng Appl Artif Intell*, vol. 139, p. 109498, 2025, https://doi.org/https://doi.org/10.1016/j.engappai.2024.109498.
[22] W.-T. Sung, I. Griha Tofik Isa, and S.-J. Hsiao, “An IoT-Based Aquaculture Monitoring System Using Firebase,” *Computers, Materials & Continua*, vol. 76, no. 2, pp. 2179–2200, 2023, https://doi.org/10.32604/cmc.2023.041022.
[23] Y. Bergeon, J. Motsch, and V. Krivánek, “Robotics in master’s degree including multi-disciplinary projects Case of Semester in first year of master’s degree,” in *IFAC-PapersOnLine*, pp. 168–173, Jul. 2022, https://doi.org/10.1016/j.ifacol.2022.09.275.
[24] ROBOTIS, “TurtleBot3 overview.” Accessed: Oct. 16, 2024. [Online]. Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/.
[25] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, “Robot Operating System 2: Design, architecture, and uses in the wild,” *American Association for the Advancement of Science*, vol. 7, no. 66, 2022, https://doi.org/10.1126/scirobotics.abm6074.
[26] S. Macenski, T. Moore, D. Lu, A. Merzlyakov, and M. Ferguson, “From the Desks of ROS Maintainers: A Survey of Modern & Capable Mobile Robotics Algorithms in the Robot Operating System 2,” *Robotics and Autonomous Systems*, vol. 168, p. 104493, 2023, https://doi.org/10.1016/j.robot.2023.104493.
[27] P. K. Panigrahi and S. K. Bisoy, “Localization strategies for autonomous mobile robots: A review,” *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 8, pp. 6019–6039, Sep. 2022, https://doi.org/10.1016/j.jksuci.2021.02.015.
[28] S. Macenski, F. Martin, R. White, and J. G. Clavero, “The Marathon 2: A Navigation System,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, Oct. pp. 2718–2725, 2020, https://doi.org/10.1109/IROS45743.2020.9341207.
[29] M. Iovino, E. Scukins, J. Styrud, P. Ögren, and C. Smith, “A survey of Behavior Trees in robotics and AI,” *Rob Auton Syst*, vol. 154, p. 104096, Aug. 2022, https://doi.org/10.1016/j.robot.2022.104096.
[30] M. Schrick, J. Hinckeldeyn, M. Thiel, and J. Kreutzfeldt, “A microservice based control architecture for mobile robots in safety-critical applications,” *Rob Auton Syst*, vol. 183, p. 104795, Jan. 2025, https://doi.org/10.1016/j.robot.2024.104795.

[31] Brian P. Gerkey, "AMCL Documentation." Accessed: Nov. 11, 2024. [Online]. Available: http://wiki.ros.org/amcl.
[32] P. Marin-Plaza, A. Hussein, D. Martin, and A. de la Escalera, "Global and Local Path Planning Study in a ROS-Based Research Platform for Autonomous Vehicles," *J Adv Transp*, vol. 2018, pp. 1–10, 2018, https://doi.org/10.1155/2018/6392697.
[33] D. Rachmawati and L. Gustin, "Analysis of Dijkstra's Algorithm and A∗ Algorithm in Shortest Path Problem," in *Journal of Physics: Conference Series*, vol. 1566, 2020. https://doi.org/10.1088/1742-6596/1566/1/012061.
[34] A. Goyal, P. Mogha, R. Luthra, and M. N. Sangwan, "PATH FINDING: A* OR DIJKSTRA'S?," *International Journal in IT and Engineering*, vol. 2, no. 1, 2014, [Online]. Available: http://www.ijmr.net.
[35] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1433–1440, May 2016, https://doi.org/10.1109/ICRA.2016.7487277.
[36] S. Macenski, R. White, and J. Wallace, "Configuring behavior server." Accessed: Nov. 12, 2024. [Online]. Available: https://docs.nav2.org/configuration/packages/configuring-behavior-server.html.
[37] D. V. Lu, D. Hershberger, and W. D. Smart, "Layered costmaps for context-sensitive navigation," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 709–715, Sep. 2014, https://doi.org/10.1109/IROS.2014.6942636.
[38] S. Macenski, R. White, and J. Wallace, "Detailed Behavior Tree Walkthrough." Accessed: Dec. 10, 2024. [Online]. Available: https://docs.nav2.org/behavior_trees/overview/detailed_behavior_tree_walkthrough.html.
[39] C. Ntantogian, P. Bountakas, D. Antonaropoulos, C. Patsakis, and C. Xenakis, "NodeXP: NOde.js server-side JavaScript injection vulnerability DEtection and eXPloitation," *Journal of Information Security and Applications*, vol. 58, p. 102752, May 2021, https://doi.org/10.1016/j.jisa.2021.102752.
[40] T. A. Dwinandana, T. P. Kasih, and M. N. Puji, "The Development of Conceptual Design of Nurse Assistance Robot's Exterior with Ergonomic Approach*," Jurnal Asiimetrik: Jurnal Ilmiah Rekayasa & Inovasi,* vol. 6, pp. 253–264, 2024, https://doi.org/10.35814/asiimetrik.v6i2.6320.
[41] K. Zheng, "ROS Navigation Tuning Guide," *Arxiv,* 2017, [Online]. Available: http://arxiv.org/abs/1706.09068.
[42] T. Foote, "laser_filters package documentation." Accessed: Oct. 23, 2024. [Online]. Available: https://wiki.ros.org/laser_filters.
[43] G. Jiang, D. D. Lichti, T. Yin, and W. Y. Yan, "A Maximum Entropy Based Outlier Removal for Airborne LiDAR Point Clouds," *IEEE J Sel Top Appl Earth Obs Remote Sens*, vol. 17, pp. 19130–19145, 2024, https://doi.org/10.1109/JSTARS.2024.3478069.
[44] J.-I. Park, J. Park, and K.-S. Kim, "Fast and Accurate Desnowing Algorithm for LiDAR Point Clouds," *IEEE Access*, vol. 8, pp. 160202–160212, 2020, https://doi.org/10.1109/ACCESS.2020.3020266.
[45] S. Macenski, R. White, and J. Wallace, "Filtering of Noise-Induced Obstacles." Accessed: Oct. 25, 2024. [Online]. Available: https://docs.nav2.org/tutorials/docs/filtering_of_noise-induced_obstacles.html.
[46] S. Macenski, R. White, and J. Wallace, "Navigating with Keepout Zones." Accessed: Oct. 25, 2024. [Online]. Available: https://docs.nav2.org/tutorials/docs/navigation2_with_keepout_filter.html.