Development of an Autonomous Mobile Manipulator System for Fetch-and-Carry Tasks

Isira Naotunna, and Theeraphong Wongratanaphisan

***Abstract*— This paper presents a detailed development of a mobile manipulator to perform fetch-and-carry tasks autonomously in a structured indoor environment. This system was specially designed to prevent human contact in the work environment during the Covid-19 pandemic situation. The system setup includes a Baxter manipulator mounted on a two-wheeled differential drive mobile robot. The ROS melodic platform installed on Nvidia Jetson TX1 embedded system was used for software development. Autonomous navigation and visual servoing are implemented using an Intel RealSense depth camera and a tracking camera which are collaborated with ROS Simultaneous Localization and Mapping (SLAM) and navigation. The system was validated under two experiments to 1) analyze the performance of mobile manipulator navigation and 2) validate the overall system performance. Through several trials in each experiment, it was demonstrated that the developed system can perform fetch-and-carry tasks with collision-free navigation consistently and repeatedly.**

***Index Terms*—Mobile Manipulator, Autonomous Navigation, Robot Operating System (ROS), SLAM, Baxter Manipulator, Aruco Markers**

I. Introduction

COVID-19 pandemic has immensely affected the manufacturing and service sectors restricting various daily activities. Therefore, researchers are investigating solutions to reduce human contact as much as possible to mitigate the spread of pathogens. Integration of mobile manipulator systems as a shield is becoming a powerful tool to prevent the fear of contamination. Either autonomous or teleoperated, mobile manipulators are well-suited to perform fetch-and-carry tasks in many different areas such as domestic services, aerospace, manufacturing, and agriculture [1]. In addition to these, operators engaged in robots generally require a significant amount of training and experience to operate and control a robotic mission. Therefore, researchers focus on developing fully autonomous mobile manipulator systems that can perform a specific task while safely interacting with an unknown environment that includes static and dynamic objects.

In response to that, we have contributed to these requirements by developing a ROS (Robot Operating System)-based mobile manipulator robot that exhibits autonomous and intelligent behavior to perform a fetch-and-carry task in an indoor workspace. The system is developed to reduce human responsibilities when performing the fetch-and-carry task in a structured environment. The robot includes three main sub-systems; i.e., a manipulator system (Baxter), a mobile robot, and the 3D vision system. Communication between these three systems is controlled via ROS by connecting to a host computer. The system is developed by undergoing a thorough study on robot kinematics, computer vision, visual servoing, SLAM, navigation, and object manipulation. The robot's autonomous navigation system is developed using the ROS navigation stack with real-time appearance-based mapping and localization techniques. The object detection and the manipulation techniques are developed using the Aruco marker-based visual servoing technique. The primary sensor of the robot is the Intel RealSense D435i camera, which is used in both navigation and target detection.

One of the main challenges that one has to overcome when implementing SLAM and navigation methods for the mobile robot system is to minimize the errors in odometry. The system requires accurate odometry information to obtain fault-less localization during navigation. Wheel encoders are commonly used to acquire the odometry information of mobile robotic systems. In addition to that, some of the robots are developed by fusing the Internal Measurement Unit (IMU) and wheel encoder data to minimize the odometry errors. When implementing a ROS navigation stack for the mobile robotic system, the robot's transformation tree (TF) of the ROS navigation stack is published based on the odometry information. Because of this, the accuracy of the robot's localization depends on the robot's TF tree. However, we have used the Intel RealSense T265 tracking camera information to develop the robot's TF tree by fusing the TF tree of the T265 camera to the robot's TF tree instead of using the common approach of ROS. This approach avoids localization errors that occur due to wheel slipping and the robot's vibration.

Most of the existing ROS-based mobile manipulator systems such as PR2 [2], Robotnik RB-1[3], Tiago [4], and Rob@work4 [5] consists of multiple sensors such as laser scanners, LIDAR, and 3D vision systems. Localization techniques in these systems are developed based on ROS AMCL which uses multiple laser scanner data. They also use multiple sensor sources for obstacle detection as well. But in this research, we have set up the ROS navigation stack by using only the Intel RealSense D435i camera with the help of the TF tree developed based on the Intel RealSense T265 tracking camera. The developed ROS navigation technique with RealSense T265 tracking camera and D435i depth camera is tested and validated by experiments. In addition to that, the overall performance of the system is also validated. The main objective of this paper is to provide a complete overview of the developed system as a helpful guide for ROS developers. This guide will help any robotic developer to transform an existing robot manipulator system into an autonomous mobile manipulator system to perform a basic

This paper is submitted on March 5th, 2021. This work was supported in part by Science and Technology Park, Chiang Mai University under grant no.165/2563 and Office of the National Commission Ministry of Digital Economy and Society under grant no.1028/63

I. Naotunna is a master-degree student in the Graduate Program in Mechanical Engineering, Faculty of Engineering, Chiang Mai University, Chiang Mai, Thailand 50200 (e-mail: naotunna_isira@cmu.ac.th).

T. Wongratanaphisan is with Department of Mechanical Engineering and Center for Mechatronic System and Innovation, Chiang Mai University, Chiang Mai, Thailand 50200 (e-mail: theeraphong.wong@cmu.ac.th).

fetch-and-carry task without adding multiple sensor sources. The following subsections provide a detailed description of the proposed mobile manipulator's hardware systems, software systems and control strategies. Experimentation tests and their results are presented and discussed in the final sections.

II. HARDWARE ARCHITECTURE AND SYSTEM DESIGN

The robot setup is developed with a 7-DOF, dual-arm Baxter manipulator system mounted on a differential drive two-wheeled mobile robot, as shown in Fig. 1. These subsystems are controlled by ROS melodic operated in the Nvidia Jetson TX1 platform. Autonomous navigation and visual servoing are implemented using the Intel RealSense D435i depth camera and T265 tracking camera which, also communicate with Nvidia TX1 via RealSense SDK [6] and the RealSense ROS wrapper [7].

Fig. 1. The mobile manipulator system developed for fetch-and-carry tasks

A. Baxter Manipulator System

Baxter robot is an industrial hardware system consisting of two 7-DOF arms. The robot is approximately 3 feet tall with a weight of 75Kg. Both Baxter's arms have angle position and joint torque sensing facilities. Baxter robot has three integrated cameras, and these cameras are located at the end effectors of each joint and the robot head. In addition to this, the Baxter robot also consist of sonar sensors, accelerometers, and range-finding sensors that allow it to perform collision-free manipulation [8].

B. Mobile Robot Development

The mobile robot is a two-wheeled differential drive system with a platform constructed using mild steel. It has a length of 0.8m, a width of 0.5m, and a height of 0.7m, as shown in Fig. 2. The mobile robot system is controlled by Arduino MEGA 2560, which is communicated with the NVidia TX1 via ROS Serial package [9] with a baud rate of 512 kbps. The system is driven using a couple of MY1016Z3 brushed DC motors, each controlled separately using ELMO-VIO 25/60 motor control units. Since the motor controller requires an analog control signal to perform smooth motor controlling, MCP4922 Digital to Analog Converter (DAC) is used to generate the analog control signal from Arduino. The converter establishes an SPI communication between itself and Arduino with a frequency of 10MHz. Feedback from the motors is acquired using AMS's ASP5147P absolute magnetic encoders communicate with Arduino via Serial Peripheral Interface (SPI) with the frequency of 1MHz. Readings from the encoder are recorded every 1ms using the Arduino timer 1 interrupts.

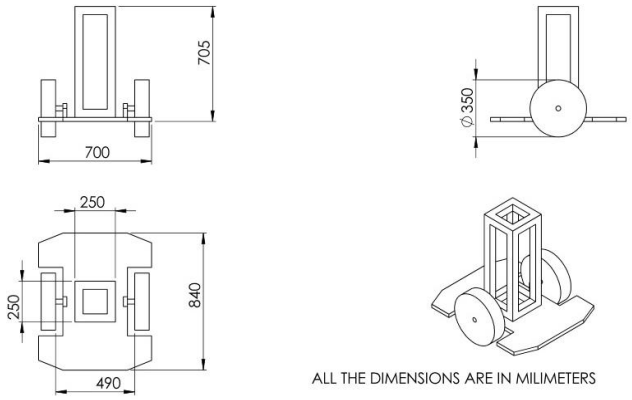


Fig. 2. Mobile robot structure and dimensions

The mobile robot controller is designed based on the kinematic model given in (1). As shown in Fig. 3, in (1), R, L, V_l, and V_r represent the wheel radius of 0.175m, the wheel distance of 0.6m, the linear velocity of the left wheel, and the linear velocity of the right wheel respectively. In addition to that, V_x, V_y, ω, and φ_0 denote the robot's linear and angular velocity components, and angular displacement.

$$\begin{bmatrix} \omega \\ V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \frac{-R}{L} & \frac{R}{L} \\ \frac{R\cos(\varphi_0)}{2} & \frac{R\cos(\varphi_0)}{2} \\ \frac{R\sin(\varphi_0)}{2} & \frac{R\sin(\varphi_0)}{2} \end{bmatrix} \begin{bmatrix} V_l \\ V_r \end{bmatrix} \quad (1)$$

Based on the wheeled mobile robot's kinematic equation, the angular velocities of the left (ω_l) and right (ω_r) motors are calculated as shown in equations (2) and (3). The G value in the equations represents the speed ratio between motor and wheel, which is 4.9:1 for this system.

$$\omega_r = \frac{G(2V + L\omega)}{2R} rads^{-1} = \frac{G(2V + L\omega)}{2R\pi} \deg s^{-1} \quad (2)$$

$$\omega_l = \frac{G(2V - L\omega)}{2R} rads^{-1} = \frac{G(2V - L\omega)}{2R\pi} \deg s^{-1} \quad (3)$$

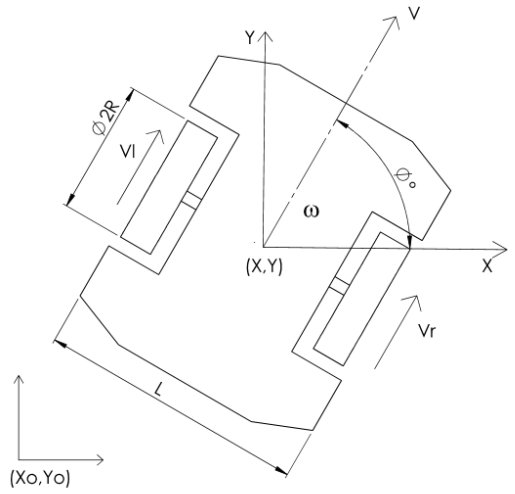


Fig. 3. Kinematic model of the mobile robot

According to (2) and (3), a velocity feedback Proportional and Integral (PI) controller is designed to control the velocity of the system using the velocity feedback calculated from the angle values obtained from the wheel encoders. The control signals are calculated as 14-bit values to generate the analog motor control signals via the MCP4922 DAC. In this system, the MCP4922 DAC generates the maximum voltage (V_{max}) of 4.2V for the highest 14-bit value of 4096. Therefore, the reference voltage (V_{ref}) is selected as 2.1V. If the bit value of the control signal is K, then the input control voltage signal (V_{cmd}) is calculated as follows.

$$V_{cmd} = \frac{V_{\max}}{4096} K \tag{4}$$

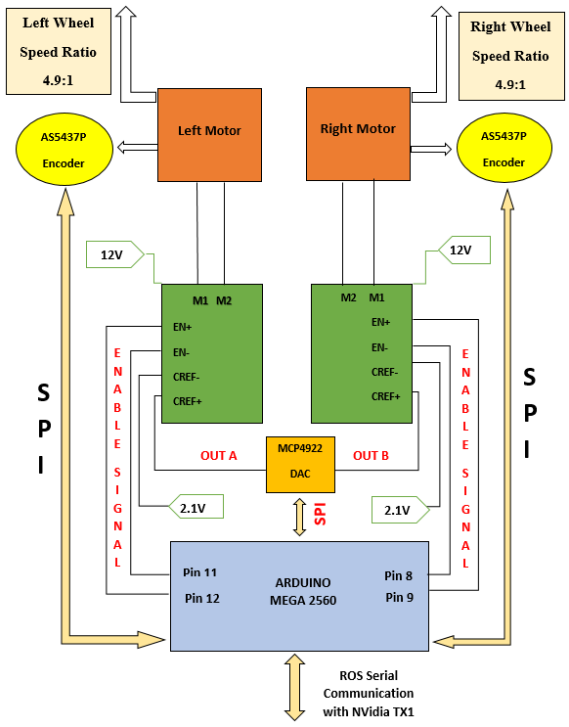


Fig. 4. Mobile robot controlling unit

Hence, the actual control signal $V_{control}$ is represented as (5),

$$V_{control} = V_{cmd} - V_{ref} \tag{5}$$

According to the relationship generated by (4), the robot's motion can be defined based on the range of K values. K should be in 2048<K<4096 for the forward motion, K=2048 for the neutral conditions, and for the backward motion, K should be 0<K<2048. Fig. 4 represents the overall control unit of the mobile robot. As shown in Fig. 4, digital pins 8, 9, 11, and 12 of Arduino Mega connects to the Enable channels EN+ and EN- of the motor controllers. When the robot operates, the motor controllers are activated by supplying a 5V signal to the EN+ channels of the motor controllers. The reference signal of 2.1V connects to the CREF- pins and the output channels of the MCP4922 DAC connect to the CREF+ channels of the motor controllers. Using these voltages supplied to the CREF+ and CREF- channels, the required control signals are generated to drive the motors.

C. 3D Vision System

The vision system comprises an Intel RealSense D435i depth camera and an Intel RealSense T265 tracking camera. The D435i is a depth camera that can be used in both indoor and outdoor environments. It uses an Active IR stereo depth technology with a minimum depth of 0.105m and 1280 x 720 depth resolution with a depth frame rate of up to 60 fps. The sensor resolution is 1980 x 1080 RGB with a 30 fps RGB frame rate. The T265 tracking camera includes two fish-eye lenses which provide a combined, close to hemispherical 163±5° field of view for robust tracking. However, to avoid unnecessary data overloading during the operation of the robot, the frame rates of the cameras are reduced to 15 fps. In this system, the tracking data gathered from the T265 camera is used to identify the location of the robot. In addition to these cameras, the head and arm cameras of the Baxter robot can be used if necessary. The two RealSense cameras are connected to the Nvidia TX1 via a USB hub, and Intel RealSense Linux drivers are installed to Nvidia TX1 to communicate with the cameras. Since all vision tasks related to autonomous navigation and visual servoing will be carried out in the ROS environment, RealSense ROS wrapper libraries are used to collaborate RealSense cameras with ROS Simultaneous Localization and Mapping (SLAM), and navigation.

D. Software Architecture

A cluster of software running inside the Nvidia TX1 development board with Ubuntu 18.04 operating system and Jetpack 4.2 drivers were installed. To operate the system, the Nvidia TX1 has to establish communication within all three subsystems simultaneously. ROS melodic software was used for this purpose. Baxter and Arduino SDKs are installed into the ROS workspace for programming and control. The vision system is communicated through the RealSense SDK 2.35 and RealSense ROS wrapper. The overall software architecture is shown in Fig. 5. Additionally, remote communication is achieved by establishing a remote desktop server between Nvidia TX1 and a remote desktop using a VNC server via a

Wi-Fi connection. Remote communication is required to control the system while in operation.

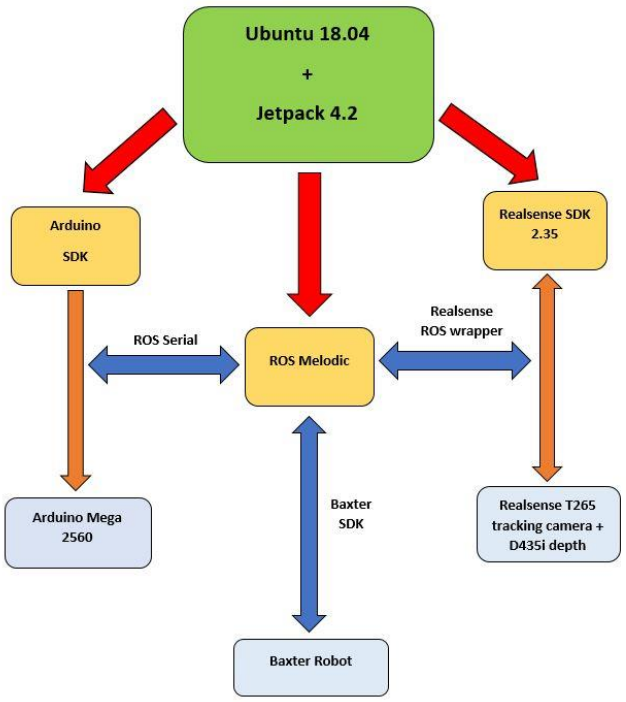


Fig. 5. Software architecture of the system

E. Power Unit

The developed mobile manipulator system is powered by two 12V lead-acid batteries. Although the system runs on DC voltage, an AC voltage is required to power up the Baxter robot. The AC voltage is generated by using a MeanWell TS1000 inverter. The inverter produces an output voltage of 230V from two serially connected 12V batteries. The schematic diagram of the power unit of the system is shown in Fig. 6

F. Simultaneous Localization and Mapping

The mobile manipulator is developed to autonomously navigate through a map while searching the target and transport the object to the desired goal location. Therefore, before performing the tasks, it is necessary to create a map of the operating environment. The map is developed using the ROS RTAB-Map package [10]. RTAB-Map can be used to generate a 3D map of the environment using 3D point cloud data using an RGBD-SLAM approach based on an incremental appearance-based global loop closure detector with real-time constraints.

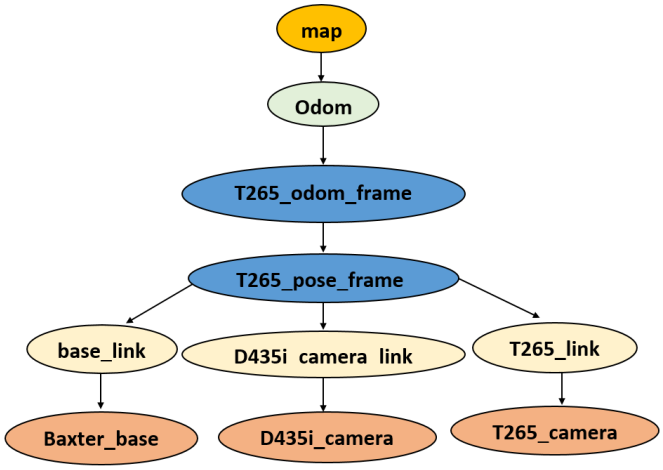


Fig. 7. Robot TF tree

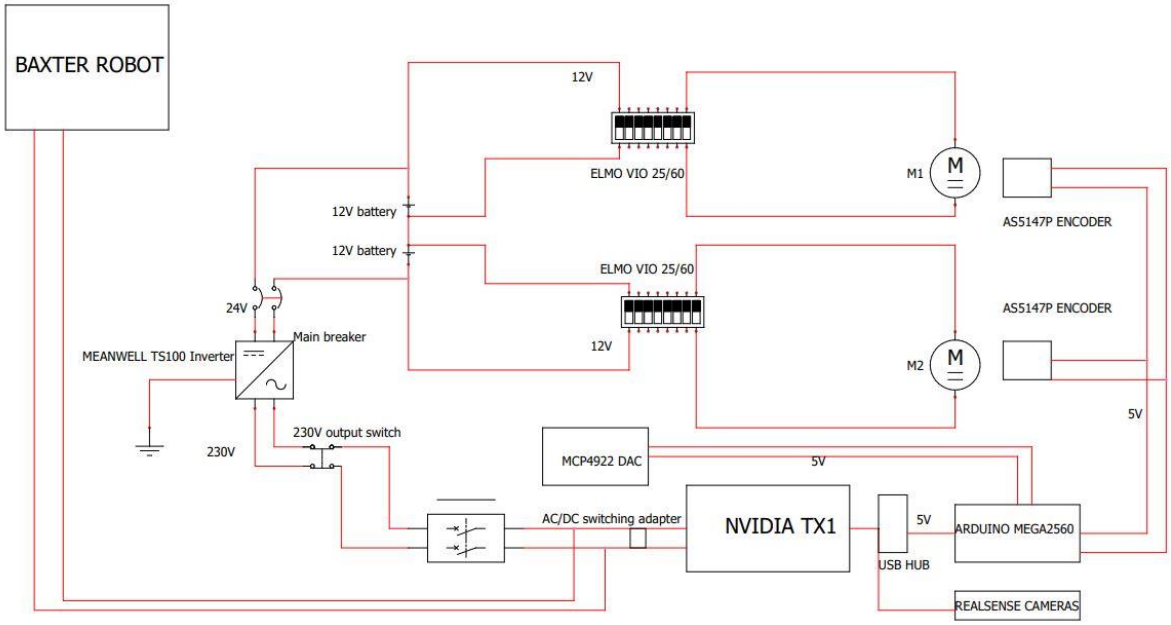


Fig. 6. Power distribution diagram of the system

To perform accurate SLAM and navigation with ROS, it is necessary to provide the correct transformation (TF) tree of the robot. Generally, most of the ROS applications use the REP 105 coordinate frame system to set up the TF tree of robots [11]. In the traditional REP 105 setup, the odometry frame always directly links to the base frame of the robot. But in this system, we have developed the robot's TF tree with respect to the T265 tracking camera's odometry frame, as shown in Fig.7. The implemented TF setup gives an accurate transformation directly from the odometry data provided by the T265 tracking camera. This approach helps to avoid the odometry errors caused due to wheel slipping and quick rotations.

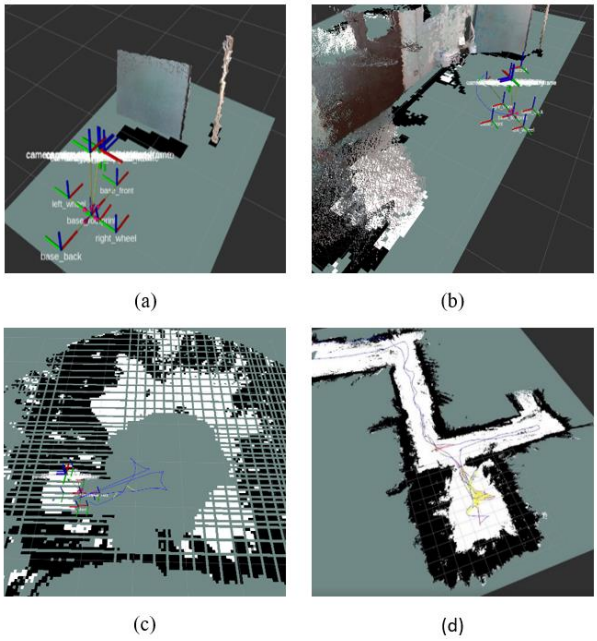


Fig. 8. Example scenario of map generation with RTAB-MAP

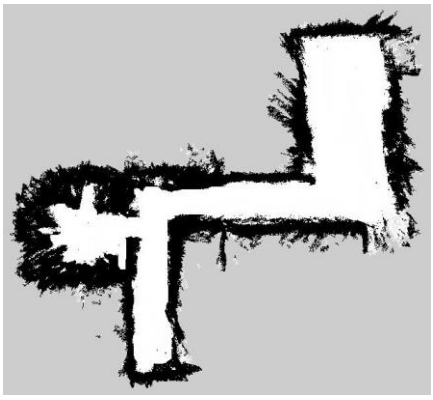


Fig. 9. Generated 2D occupancy grid map

During the mapping process, RTAB-Map generates the 2D grid map and a 2D projected map, as shown in Fig. 8. Fig. 8(a) shows the initial configuration of the map generation process. Here, a TF tree of the robot setup is configured correctly, and point cloud data is correctly subscribed to the RTAB-Map to initiate mapping. Fig. 8(b) shows the example 3D mapping scenario of the map generation process, and Fig. 8(c) shows the generated 2D sample grid map from the RTAB–Map. Finally, the 2D project map generated from the RTAB-Map by point cloud data, and the robot path during the mapping process is shown in Fig. 8(d). The map data and the image data are saved to the RTAB-Map database. The robot's navigation uses saved map data and image data. The generated 2D occupancy grid map from RTAB-Map is shown in Fig. 9.

G. Navigation

After generating the map, the saved map data can be used to perform the robot's navigation. Map-based navigation is done using a configured ROS navigation stack according to the system requirement. Fig.10 shows the general setup of the ROS navigation stack for the developed mobile manipulator system

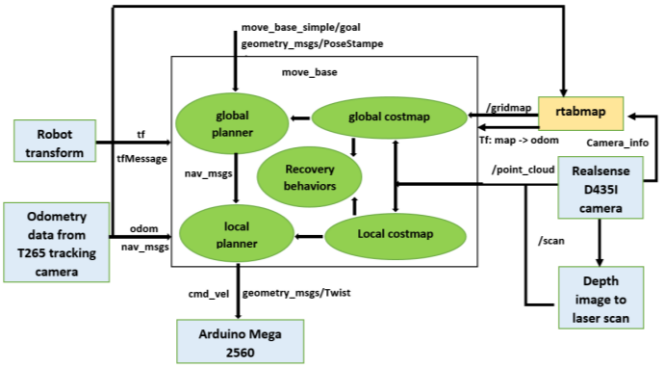


Fig. 10. ROS navigation stack configuration of the robot

Localization and path planning are the main components of map-based navigation. As shown in Fig. 10, RTAB-Map provides the localization by finding the loop closure based on the image data saved in the RTAB-Map database during the map generation process. The loop closure detector uses a bag-of-words approach to determine if a new image detected by the D435i camera is from a new location or a location it is already visited. Once a loop closure is found, the odometry is corrected, and the real-time point cloud data will be aligned to the map.

There are two main concerns in path planning.i.e.; computing a safe path to travel without collision and sending direct speed commands to the controller to allow the robot to follow the generated path. As shown in Fig. 10, the move_base node [12] is responsible for path planning in the ROS navigation stack. Path planning also contains two sub-levels known as the global planner level and the local planner level. Both the local and global planners are calculated based on the information provided by the local and global costmaps. Global costmap is generated by inflating the obstacles on the static map provided by RTAB-Map. The global costmap is used to generate the global plan for the navigation using the navfn package [13], which is developed based on the Dijkstra algorithm [14]. The local costmap deals with the real-time data provided by the D435i camera. In this setup, the point-cloud provided by the D435i camera, and fake laser scan data generated using the depth images from the D435i depth camera are used to detect the obstacles. An example configuration of the generated laser scan model visualization in Rviz and the actual robot position is shown in Fig. 11

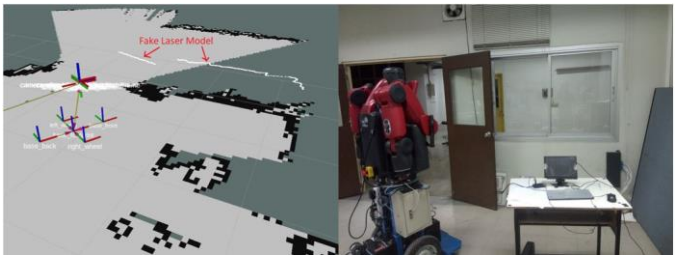


Fig. 11. Fake laser model visualization in rviz and corresponding robot position

With the help of odometry data provided by the T265 tracking camera and the local costmap data, the selected local planner package develops a local path. There are different local planner packages available in ROS that implement different algorithms and approaches for this task. This system uses the teb_local_planner ROS package [15], which uses the timed-elastic-band algorithm [16]. Based on the calculated local path, the move_base node sends the robot velocity data to the Arduino Mega 2560 and based on the velocity commands provided robot follows the local path.

To obtain a better navigation performance, the navigation stack parameters should be properly tuned based on the robot's configuration. Table I shows a few navigation stack parameters that are tuned to obtain a better performance of the experimental setup. In addition to that, other minor parameters are also tuned to increase the performance of the system. The safety of the system and the quality of path planning are the primary concerns of parameter selection

TABLE I
NAVIGATION STACK PARAMETERS

Parameter Category	Parameters
Velocity and Acceleration	Maximum linear velocity: 0.35 m/s Maximum Angular velocity: 0.4 rad/s Linear acceleration limit: 1m/s^2 Angular acceleration limit: 1rad/s^2 Maximum backward velocity: 0.1m/s
Global Planner Parameters	lethal_cost: 253 neutral_cost: 66 cost_factor: 0.55
Local Planner Parameters	xy_goal_tolerance: 0.25 yaw_goal_tolerance: 0.2 min_obstacle_dist: 0.7 Footprint model radius: 0.6m
Costmap Parameters	Obstacle range: 0.7 m Obstacle layers' observation sources: pointcloud/Scan Global costmap inflation radius: 0.4m Local costmap inflation radius: 0.3m Cost scaling factor: 3

Fig. 12 shows an example scenario of map-based navigation in the developed experimental setup with example images of rviz simulation and the real-time robot motion. Fig. 12(a) shows the robot's approach to the goal location, at which the obstacle is not in the minimum obstacle range of the costmap. Therefore, the navigation stack provides a straight path to the goal location, as shown in the rviz simulation. Fig. 12(b) shows the obstacle detecting scenario. The behavior of the local costmap and the local planner with the obstacle can be observed in the rviz simulation of Fig. 12. Lastly, Fig. 12(c) and Fig. 12(d) show the obstacle avoidance and goal-approaching scenarios.

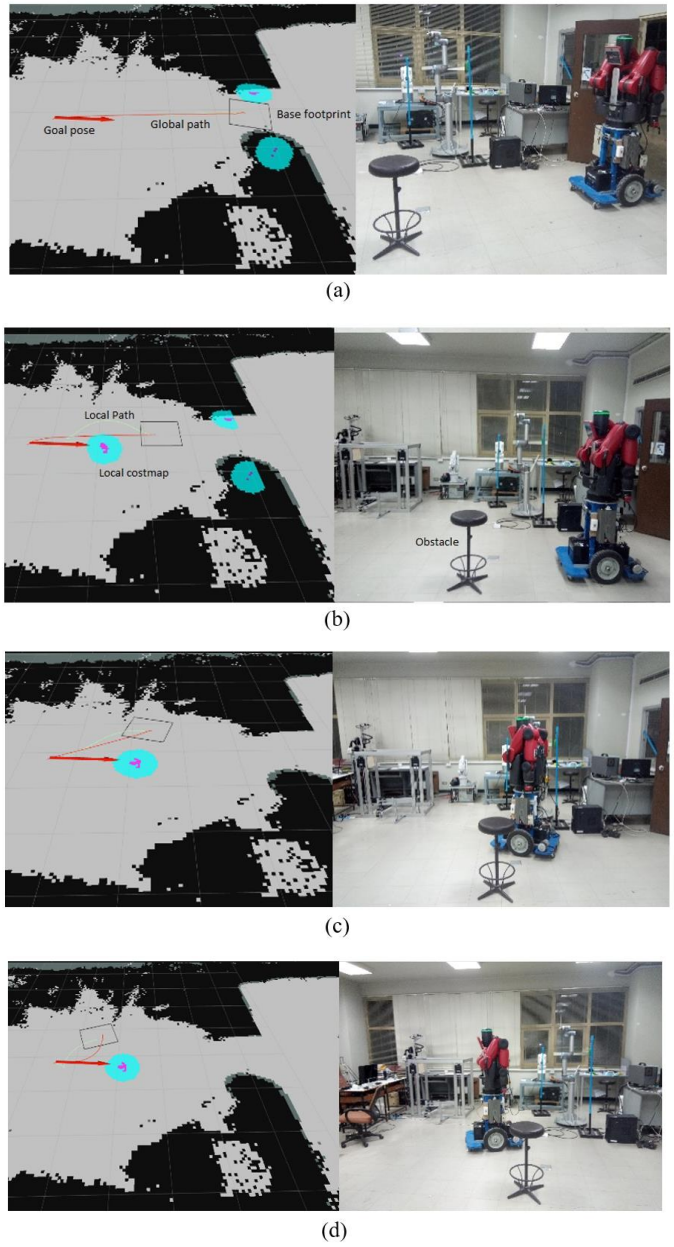


Fig. 12. Example scenario of navigation: (a) Navigation without obstacles, (b) Obstacle detection, (c) Avoiding the obstacle, and (d) Reaching to the goal location.

III. METHOD

As explained in section II, the developed mobile manipulator system can autonomously navigate through a constructed map of a workplace. This section describes the development of the Aruco marker-based fetch-and-carry methodology and mobile manipulator's control algorithm. An experimental setup is developed to demonstrate the fetch-and-carry task from the developed mobile manipulator system. In this setup, the robot navigates through different waypoints of the constructed map while searching for the target box. Once the target box is detected, the robot approaches the

target box and picks it up using Baxter's left arm. Finally, the robot arrives at the goal location and places the box at the goal. The overall control algorithm of the experimental setup is shown in Fig. 13.

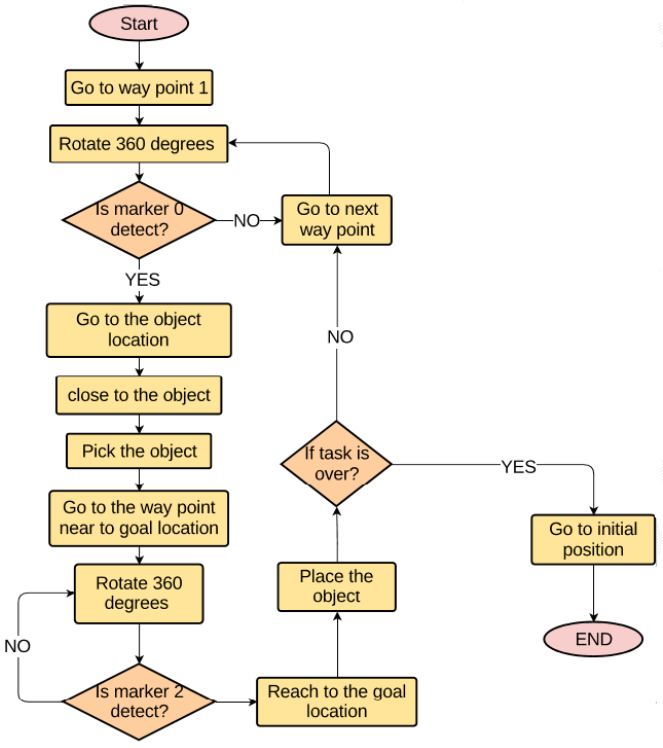
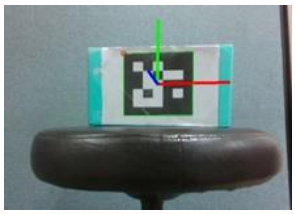
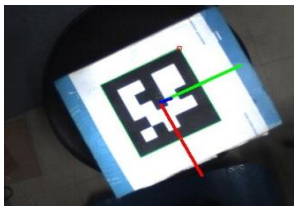


Fig. 13. Mobile manipulator control algorithm

(a)



(b)



(c)

Fig. 14. Aruco marker detection: (a) Box with Aruco markers, (b) ID 0 detection with D435i camera, and (c) ID 2 detection with Baxter left-arm camera.

In this experiment, Aruco marker-based pose estimation technique is used to identify the pose of the target object in both mobile manipulator navigation and object manipulation. A ROS python library for Aruco marker detection and pose estimation is implemented using the Aruco library for augmented reality based on OpenCV [17]. The cv_bridge plug-in is used to implement the OpenCV library with ROS. RealSense D435i camera is used for goal identification, and the left-arm camera of the Baxter with the resolution of 640x800 is used for marker detection in object manipulation. The Aruco marker poses with respect to the camera frames can be obtained from the pose estimation algorithm developed using the OpenCV Aruco pose estimation library. Fig. 14(a) shows the box that is used in the experiment, and the samples of marker detection from both cameras are shown in Fig. 14(b) and Fig. 14 (c). As shown in Fig. 14 (a), there are two Aruco markers placed on the Box. The marker on the front side is used to identify the box during the navigation and the marker on the top is used to estimate the pose during the manipulation.

A. Target Reaching

As explained in section II, the mobile manipulator system is developed with a ROS navigation stack to navigate using a known map. When the robot reaches a pre-defined waypoint in the map, the robot starts to search the Aruco markers by rotating 360 degrees around the self-axis. In this experiment, the target locations are defined with the Aruco marker ID 0, and the goal locations are defined with the ID 2 of the 4X4 Aruco dictionary. Once the marker is detected, the marker pose is calculated with respect to the robot's base frame. Then the angular error β of the marker with respect to the robot base frame is calculated using the tangent of marker x and y coordinates with respect to the base frame given in (6).

$$\beta = \tan(\frac{Fx}{Fy}) \tag{6}$$

The robot is then rotated to satisfy the angular error condition of $-0.05 < \beta < 0.05$ radians. The reason for this initial rotation is to minimize the error produced by the ROS navigation stack when reaching the target. Once the robot satisfies the angular error condition, the target location is calculated with respect to the map frame according to the following method.

Rotation angle yaw (ψ) of the robot, can be calculated using the robot orientation, which is published by the move_base node in the form of a quaternion, $q = q_0 + iq_1 + jq_2 + kq_3$ as shown in (7).

$$\psi = \tan^{-1}(\frac{2(q_0q_3 + q_1q_2)}{1-2(q_2^2 + q_3^2)}) \tag{7}$$

For the pose of the robot with respect to the map (x_{rm}, y_{rm}, z_{rm} , ψ) and pose of the marker with respect to the robot base frame (x_b, y_b, z_b), the pose of the marker with respect to the map frame (x_m, y_m, z_m) is given by (8).

$$\begin{bmatrix} x_m \\ y_m \\ z_m \end{bmatrix} = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 & x_{rm} \\ \sin(\psi) & \cos(\psi) & 0 & y_{rm} \\ 0 & 0 & 0 & z_{rm} \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_b - x_t \\ y_b \\ z_b \end{bmatrix} \tag{8}$$

Where x_t is the distance from the object to the robot goal position, it should always be larger than the defined inflation radius of the Navigation Stack to avoid unnecessary recovery behaviors.

Then the calculated x_m and y_m values are sent to the move_base node to plan the path to navigate closer to the object. Once the robot comes closer to the object, the manipulator system is brought closer to the object smoothly by sending velocity commands until the defined forward error conditions (forward error < 0.5 m) and angular error conditions ($-0.05 < \beta < 0.05$) of the marker are satisfied with respect to the base frame. After satisfying the conditions, the manipulation command is sent to the Baxter robot to perform pick-and-place tasks.

B. Object Manipulation

During the mobile manipulator operation, The Baxter's arms follow a predefined pose sequence to perform object handling. The Baxter joint trajectory action server is used to move the arms according to the specified joint angles. The joint trajectory action is a ROS node that provides an action interface for tracking trajectory execution. It passes trajectory goals to the controller and reports success when they have finished executing. When the robot's operation starts, both the arms move to the predetermined navigation pose from its default neutral pose, as shown in Fig. 15(a). The defined navigation pose avoids unnecessary collisions with the obstacles during the system navigation by keeping both arms inside the boundary of the robot's footprint. Therefore, the mobile manipulator control algorithm is programmed to maintain this predefined navigation pose during autonomous navigation.

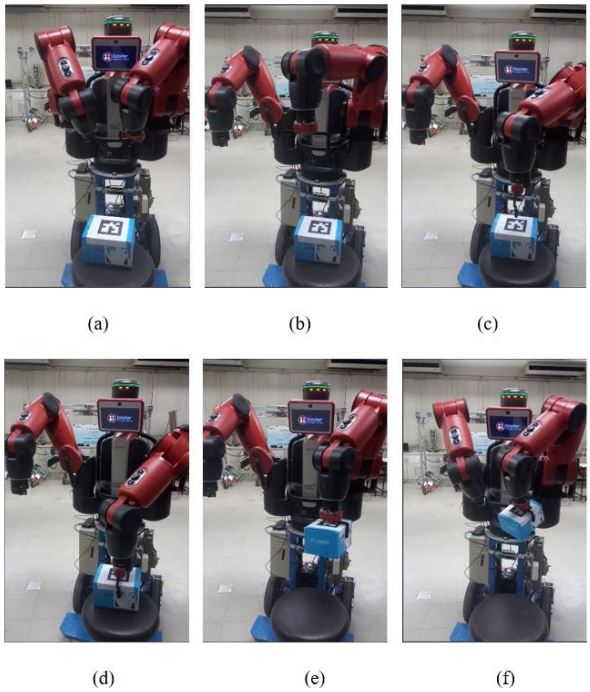
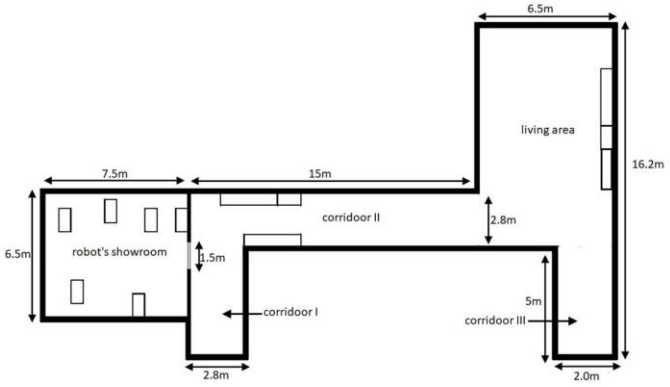


Fig. 15. Baxter performing object picking task: (a) Navigation pose, (b) Searching pose, (c) Reaching to the box, (d) Grab the box using grippers, (e) Picking the box, and (d) Return to the Navigation pose.

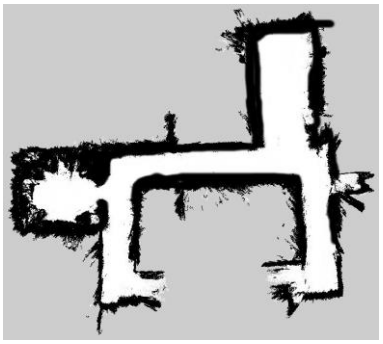
When performing the object picking task, once the robot reaches the object location, the arms move to the predefined searching pose as shown in Fig. 15(b).As mentioned before, the pose estimation of the box is done based on the Aruco marker, located at the top of the target box, which has the average dimensions of 20cm x 15cm x 12cm. The Baxter's left-arm camera is used to identify the Aruco marker. Then using the OpenCV Aruco pose estimation library, the 6D pose of the Aruco marker $P_{ar}=(X_{ar},Y_{ar},Z_{ar},roll_{ar},pitch_{ar},yaw_{ar})^T$ is calculated. Then, the Aruco marker pose with respect to Baxter's base frame is calculated using the ROS tf2 package. Next, the inverse kinematics of Baxter's left arm is calculated using Baxter's Solve_Position_IK_Request ROS service. It is an inbuilt ROS service of the Baxter robot to calculate the inverse kinematics. Based on the calculated joint angles the object is picked as shown in Fig. 15(c) and Fig.15 (d). Finally, both the arms reach back to the navigation pose and prepare for the navigation as Fig.15 (e) and Fig. (f). After that, the robot navigates to the goal location, and using a similar method, Baxter places the object on the goal location.

IV. EXPERIMENT AND RESULTS

The developed system was validated with two experiments. The first experiment was conducted to analyze the performance of the mobile manipulator navigation while the second experiment is conducted to validate the overall system performance by performing fetch-and-carry tasks. All the experiments were performed inside the first floor of the main engineering building of Chiang Mai University. The 2D map of the floor area is shown in Fig. 16(a), and the generated 2D map of the floor is shown in Fig. 16(b).



(a)



(b)

Fig. 16. 2D Map of the experimental area: (a) Detailed 2D map of the experimental area, and (b) Generated occupancy grid map.

A. Performance Analysis of the Navigation System

The performance of the system navigation is analyzed by a

navigation task to the pre-defined waypoints of the generated map, as shown in Fig. 17 (b). The experiment is conducted to evaluate the accuracy of navigation of the system along with the given map. This experiment includes 15 trials, and the entire set of data is recorded while the robot navigates from the waypoint O to waypoint A, waypoint A to waypoint B, and waypoint B to waypoint O as shown in Fig. 17. The Cartesian coordinates of the waypoint O, A, and B are (0, 0), (22.5, 2.0), and (22.5, 9.0) respectively.

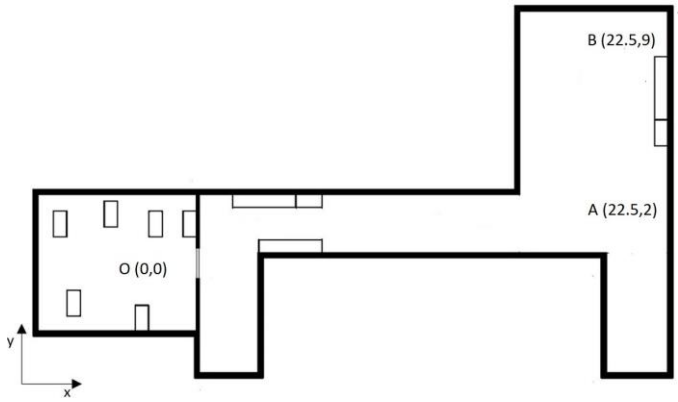


Fig. 17. Waypoints locations

To analyze the navigation performance of the developed system, the local planner and the robot's actual path are observed at each trial. Root Mean Square Deviation (RMSD) is used to validate the navigation performance of the system. In this experiment, RMSD value between the local planner and the robot's actual path $(RMSD)_{LR}$ is calculated to analyze the performance of the developed mobile robot controller. If the local position, robot position, and the number of data points are given by the (X_l,Y_l), (X_r,Y_r), and n; the $(RMSD)_{LR}$ can be calculated as shown in (9).

$$RMSD_{LR} = \sqrt{\frac{1}{n}\sum_{1}^{n}(X_l - X_r)^2 + \frac{1}{n}\sum_{1}^{n}(Y_l - Y_r)^2} \qquad (9)$$

In addition to that, the goal reaching accuracy was also calculated by measuring the final robot position of the robot once the robot completes the full path O→A→B→O. The robot's final position and the orientation is measured based on the odometry data provided by the Intel RealSense T265 camera.

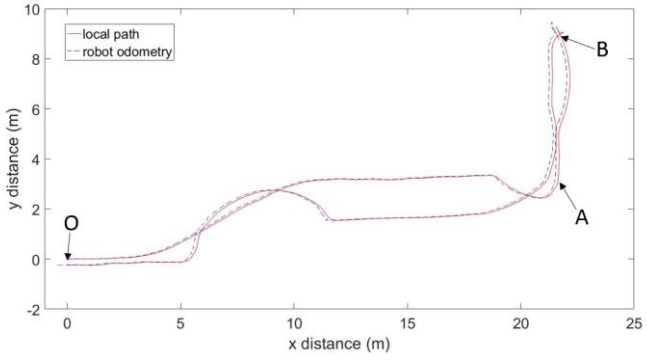


Fig. 18. Local path and the robot's actual path recorded in the second trial

Fig. 18 shows the local path and the robot's actual paths during trial 2. When carefully observed, it was seen that the robot followed the local planner without having any considerable deviation. However, when comparing the robot's actual path and the local planner, there is a slight deviation in the robot's actual path from the local planner when navigating from point A to point B. The main reason for that is the floor area of point A to point B is an empty area compared to the floor area of path OA. Therefore, RGB-D data provided by the depth camera was not enough to perform precise loop closure in an empty area. One possible explanation would be that there are no objects for the depth camera to detect, to tie the submaps together and perform loop closure. However, the odometry provided by the T265 tracking camera reduces this error in the empty environment.

According to the position and orientation errors measured at the final goal location O, it was observed that the robot did not always stop at the exact goal location O (0, 0) after completing the path O→A→B→O. The same was observed in the robot's path shown in Fig. 18. The position and orientation errors can also be reduced by minimizing the goal tolerances and the footprint size. However, reducing the goal tolerance to a lower value creates unwanted oscillations when reaching the goal points. In addition to that, the localization errors affect the accuracy of reaching the goal. However, when compared with the size of the robot, the position errors and orientation errors of the existing setup is acceptable. Table II shows the overall navigation performance of the system during the experiment.

TABLE II
OVERALL NAVIGATION PERFORMANCE OF THE SYSTEM

Performance measurement	Value
Average $(RMSD)_{LR}$	0.2 m
Minimum distance from the final goal point O	0.15m
Maximum distance from the final goal point O	0.41m
Maximum angular error at the final goal point O	0.02 rad
Minimum angular error at the final goal point O	0.15 rad
Average measured xy goal tolerance	0.21m
Average measured yaw goal tolerance	0.11 rad
Number of successful trials	15
Number of unsuccessful trials	0

B. Overall System Performance

The developed system was validated by performing fetch-and-carry tasks inside the same floor area as shown in Fig. 16. The selected floor consists of a three corridors, a room with objects, and a vacant living-area. In this experiment, the robot's main tasks were to select an object in a random position on the map, move it to the desired target, place it in the target position and return it to the starting position. Fig. 19 shows the initial position of the robot (O), goal location (G), and the object locations (P, Q, R, S) where the object is placed during the experiment. This test consists of 15 experiments, and in each experiment, the location of the object is randomly selected from one of the locations (P, Q, R, S) mentioned above. The 2D coordinates of the P, Q, R, S, and G with respect to the initial position O (0, 0) are (14.5, 3.0), (20.0, 7.5), (22.0, 9.5), (24.5, 8.5), (0.0, -1.5) respectively.

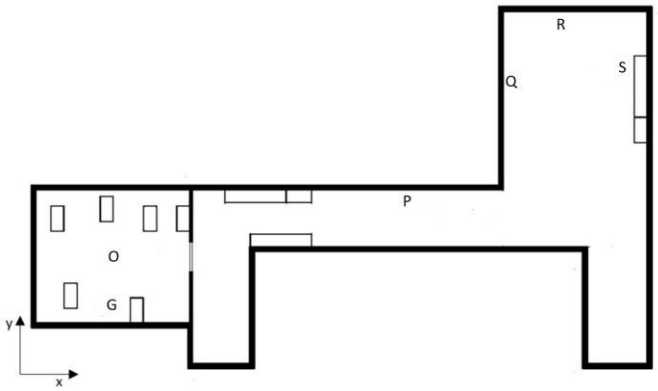


Fig. 19. Waypoints locations

The robot completed all 15 trials during the experiment. Fig. 20 shows the operation sequence during the fetch-and-carry task performed when the object is placed at the location S. In this experiment, the robot navigates to the pre-defined waypoints, as shown in the sequence number 1-8 in Fig. 20. The robot then searches for the Aruco marker ID 0, located at the front side of the box. The searching scenario is shown in sequence number 9-11 in Fig. 20. Once the marker is detected, the robot reaches the object, as shown in the sequence 12-13 in Fig. 20. Sequence 14-16 shows the box manipulation scenario and sequence 17-25 shows the navigation process to the goal location from the object location. In the goal location, the Aruco marker ID 2 is placed to identify the exact goal location. According to the Aruco pose estimation, the robot accurately reaches the goal location (sequence 27-29) as in Fig. 20. The robot then places the object on the table at the goal location (sequence 30-31). Finally, the robot returns to the initial position, as in the sequence 32-35.

V. Conclusion and Future Works

This paper demonstrates the development of a mobile manipulator capable of performing fetch-and-carry tasks. The developed system can successfully self-navigate in an indoor environment and manipulate an object. When considering the overall performance of the developed system, it can be seen that the system performs well within a simple environment with the help of the existing 3D vision system. Therefore, this can be introduced as a reasonable and straightforward solution for basic mobile manipulation tasks in various industrial and domestic environments. This system can be used to minimize human intervention in the workplace when carrying out fetch-and-carry tasks especially during the Covid-19 pandemic. The Navigation system is developed with the Intel RealSense camera setup, as explained in section III. The localization is done based on the RTAB-Map package using the RGB-D data provided by the of the D435i camera. Therefore, poor lighting conditions may sometimes cause a problem for the localization.

However, in good lighting conditions, the system performed well without causing any significant issues. This system functions well in an object-dense environment. As explained in section IV, in an empty area, the system cannot perform a precise localization due to the lack of data provided by the 3D camera to the RTAB-Map node. However, the

Fig. 20. Fetch-and-carry operation sequence of the robot

odometry provided by the RealSense T265 tracking camera reduces this error. Nevertheless, the maximum range of the RealSense D435i camera is approximately 10m. Therefore, the developed system may not perform well in an empty area more extensive than a 10m x 10m area.

Obstacle detection is also carried out by the RealSense depth camera, located at a distance of 0.57m from the base frame. As the overall robot setup's height is only 1.5m, the obstacles locate with a height of more than 1m above from the base frame cannot be identified. Also, due to the limitations of the obstacle detection, the navigation stack is developed with a larger global inflation radius, and it will limit the system navigation to space which is less than 1.5m. Introducing laser scanners will improve system navigation within a small area. However, the developed system can still avoid obstacles safely.

A fundamental object manipulation technique is used in this experimental setup. Therefore, the system can be further developed by introducing complex object manipulation techniques such as grasp planning, redundancy avoidance, and complex geometry manipulation. Currently, the system is developed to operate either the arm or base exclusively. Introducing a close kinematic chain for this system will improve the system's efficiency, allowing it to perform more sophisticated tasks.

Finally, the system has limited capabilities to deal with unexpected failures. Significantly, the system cannot deal with dynamic obstacles present on the left, right, and backside of the robots. It may cause unnecessary collisions with the robot. Also, the robot is unable to restart from the recovery state if it gets stuck during navigation. These issues can be solved by adding a multiple-camera setup, a complex state machine technique and a machine learning technique to carry out task execution by introducing possible failures.

ACKNOWLEDGEMENT

This research is supported by the Department of Mechanical Engineering at Chiang Mai University in Thailand.

REFERENCES

[1] R.Bostelman, T.Hong, and J.Marvel, “performance measurement of mobile manipulators,” in proc. Multisensor, Multisource Information Fusion: Architecture, Algorithms, and Applications, Baltimore, United States, 2015.

[2] “Collaborative Robot Series : PR2 from Willow Garage 2013,”robotiq.com,[online].Available:https://blog.robotiq.com/bid/65419/Collaborative-Robot-Series-PR2-from-Willow-Garage, Accessed on: Mar 31, 2021.

[3] Mobile manipulato rRB-1,”robotnik.eu,[online].Available: https://www.robotnik.eu / manipulators /rb-one/ , Accessed on: Mar 31, 2021.

[4] ”TIAGo, the best robotic partner for research,” pal-robotics, 2015. [online].Available:http://blog.pal-robotics.com/tiago-your-best-robot-for-research/,Accessed on: Mar 31, 2021.

[5] “rob@work3,”rob@work,2018,[online].Available:https://www.care-o-bot.de/en/rob-work.html. Accessed on: Mar 31, 2021.

[6] “Intel RealSense SDK 2.0 – Intel RealSense Depth and Tracking cameras,” Intel® RealSenseTM Depth and Tracking Cameras. [Online]. Available: https:// www. intelrealsense.com/sdk-2/, Accessed on: Mar. 05, 2021.

[7] IntelRealSense, “IntelRealSense/realsense-ros,” GitHub, Dec. 11, 2019. [Online]. Available: https:// github. Com /IntelRealSens /realsense-ros. Accessed on: Feb. 24, 2021.

[8] T.L Harman, and C. Fairchild, “Introduction to Baxter,” Aug.02, 2016. [Online].Available:https://sce.uhcl.Edu/Harman/A_CRS_ROS_SeminarDay3/UNIT3_2BaxterHW&SW/3_2_2BAXTER_Introduction_2_08_2016.pdf.

[9] “rosserial - ROS Wiki,” wiki.ros.org. [Online]. Available: http://wiki.ros.org/rosserial, Accessed on: Mar. 01, 2021.

[10] “rtabmap_ros - ROS Wiki,” wiki.ros.org. [Online]. Available: http://wiki.ros.org/rtabmap_ros,Accessed on: Mar. 02, 2021.

[11] “REP 105 -- Coordinate Frames for Mobile Platforms", wiki.ros.org, [Online]. Available: https://www.ros.org/reps/rep-0105.html. Accessed on: Mar. 01, 2021.

[12] “move_base - ROS Wiki,” wiki.ros.org. [Online]. Available: http://wiki.ros .org/move_base, Accessed on: Mar. 01, 2021.

[13] “navfn - ROS Wiki,” wiki.ros.org. [Online]. Available: http://wiki.ros .org/navfn, Accessed on: Feb. 27, 2021.

[14] E.W.Dijkastra,”A note on two problems in connection with graphs,”Numerische mathematik, vol.1, no.1, pp.269-271, 1959.

[15] “Wiki.ros.org. (2020). teb_local_planner” - ROS Wiki. [online] Available at: http://wiki.ros.org/teb local planner, Access on: Feb. 27, 2021.

[16] C. Roesmann, W. Feiten, T. Woesch, F. Hoffmann and T. Bertram,” Trajectory modification considering dynamic constraints of autonomous robots,” ROBOTIK 2012; 7th German Conference on Robotics, Munich, Germany, 2012, pp. 1-6.

[17] Doxygen. "Aruco Marker Detection." OpenCV. [Online]. Available: https://docs.opencv.org/master/d9/d6a/group__aruco.html, Accessed on: March. 4, 2020.

Isira Naotunna received the B.Sc. degree in Department of Mechanical Engineering from University of Moratuwa, Sri Lanka, in 2016, and currently pursuing M.Eng. degree in Department of Mechanical Engineering from Chiang Mai University, Thailand. His current research interests and publications are in the areas of robotics, and mechatronics Systems

Theeraphong Wongratanaphisan received the M.S. and Ph.D. degrees in Mechanical Engineering from Lehigh University, Bethlehem, PA, USA, in 1996, and 2001, respectively. He is currently an Associate Professor at Department of Mechanical Engineering, Chiang Mai University, Thailand. Her current research interests and publications are in the areas of mechatronics, robotics, and artificial intelligence.