# ROS Based Autonomous Mobile Manipulator Robot

B. M. Sufiyan Ali(✉), Syeda Mehvish Anwar, M. A. Razaq Khan, and Kaleem Fatima

Department of Electronics & Communication Engineering, Muffakham Jah College of Engineering & Technology, Osmania University, Hyderabad, India
sufiyanali160401@gmail.com

**Abstract.** This paper presents the implementation of an autonomous mobile manipulator robot using Robot Operating System (ROS) as middleware. Jetson nano 2GB is utilized as the onboard computer which runs Ubuntu 18.04 OS in which ROS is installed. ROS navigation stack is setup and configured for the robot navigation. The system utilizes 2D LiDAR to perform navigation with obstacle avoidance. A depth camera is used for extracting the coordinates of the object and MoveIt motion planning framework employs as the manipulation platform to initialize the pick and place operations of the 5 DoF robotic arm mounted on the mobile base. Multiple experiments were conducted to examine usability in order to assess performance. Our experiment results show that the robot can detect and pick an object and navigate to a given destination by avoiding obstacles and place it on the desired location.

**Keywords:** Autonomous mobile manipulator robot · ROS · SLAM · LiDAR · Depth camera · navigation

## 1 Introduction

The key to many emerging robotics applications is mobile manipulation [1]. By combining the benefits of mobile platforms and robotic manipulator arms, these systems lessen their shortcomings. The mobile platform, for instance, increases the arm's workspace while the arm itself offers a variety of operational capabilities. Mobile manipulators provide valuable applications in hospitals, warehouses, offices, construction sites, homes to name a few.

The robot employs a differential drive for locomotion and a 5 degrees of freedom robotic arm is mounted on the mobile base. The open-source Robot Operating System framework aids researchers in creating, reusing, and improving code for various robotics applications [2]. ROS navigation stack is popular for implementing autonomous robot navigation [3] while MoveIt has been widely used for robotic arm manipulation applications. In addition, ROS offers APIs for developing customized packages and interacting with external systems.

The process of SLAM and path planning for navigation requires sensor messages such as Laserscan or pointcloud which are provided by LiDAR or depth camera respectively. Considering the heavy computations of using 3D pointcloud for mapping and


B. Raj et al. (Eds.): ICETE 2023, AER 223, pp. 780–789, 2023.
https://doi.org/10.2991/978-94-6463-252-1_78

**Fig. 1.** Mobile manipulator robot

navigation as well as the cost, RPLiDAR A1M8 was selected. The robot is designed using SolidWorks software and URDF file is generated which contains the information of all the links and joints of the robot parts and sensors [4]. Information of this system is separated into sections. System and equipment details are present in Section 2 System overview. Robot kinematics and ROS navigation stack setup for autonomous navigation is in Section 3 Robot navigation. The implementation of MoveIt using ROS and object detection is in Section 4 Robot manipulation. Results of the test cases are discussed in Section 5 Experiment and results while Section 6 concludes the research of this implementation.

## 2 System Overview

### 2.1 ROS Across Multiple Machines

We used ROS as the main platform for developing the entire software architecture of the robot. ROS supports distributed computations, allowing computations to be relocated at run-time to match the available resources. The laptop was configured to use Jetson nano as the ROS master by setting ROS_MASTER_URI with the IP address of Jetson nano on both the computers. This allowed us to execute hardware nodes corresponding to LiDAR and depth camera on the on-board computer and other nodes corresponding to SLAM, MoveIt and the visualization tools such as Rviz and rqt on the laptop. The bi-directional connectivity between the pair of machines resulted in the improvement of system performance.

### 2.2 Equipment and Sensors

It has a depth camera and a 2D LiDAR installed on the mobile robot body. At the centre of the robot, the LiDAR sensor was mounted and calibrated. The RP-LiDAR ROS wrapper, a supported package, has made it compatible with ROS and ready to use. For quick processing of sensor data, a jetson on board is linked. The sensors and robot are listed below.

- "RP-LiDAR A1M8" This 360-degree 2D laser scanner with a 6-metre range generates 2D point clouds for use in mapping, localisation, and modelling of objects and environments.
- "Intel RealSense D435i" This depth detecting stereo camera served as the sole and primary sensor for gathering visual data about the workplace, which helped with the pick and place operation.
- "Wheel Encoders" This is fastened to the wheels of the mobile robot. A differential drive with two power wheels and one encoder sensor mounted on each serve as the foundation of the robot. Encoders are the source for the position and orientation i.e., the odometry information.

## 3 Robot Navigation

The overview of ROS navigation stack [5] is shown in Fig. 1. The stack expects the robot to publish information about the relationships between co-ordinate frames over time using tf for different applications like, to determine the robot location in the world, to transform sensor messages from sensor link to base link and relate sensor data to a static map (Fig. 2).

Odometry source provides information about velocity of the robot by publishing transform and nav_msgs over ROS.

### 3.1 Robot Kinematics and Odometry

The odometry source node takes feedback from wheels motor encoders and provides current position of robot on odom topic to move_base node. The base controller node accepts the velocity commands from the move_base and is responsible to provide individual wheel velocities to the wheel motors. To write the base controller and odometry source of the robot, it is important to understand the kinematics of differential drive. The robot motion is defined with a linear velocity ($v$) and angular velocity ($\omega$). These velocities must be converted into individual wheel velocities in order to operate in robot joint space. These conversions are carried out using two physical parameters of the robot

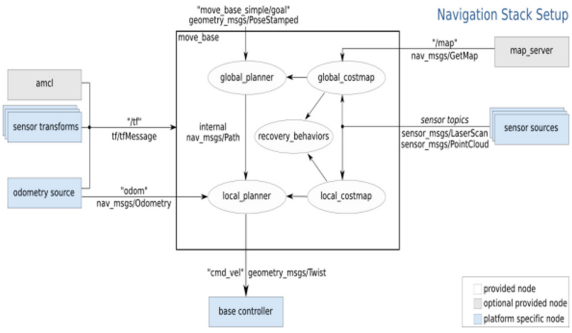


**Fig. 2.** Navigation stack setup

namely, wheel separation length (L) and wheel radius (R). The left wheel velocity ($v_l$) and right wheel velocity ($v_r$) are computed through the following equations.

$$v_t = \frac{2v - \omega \mathrm{L}}{2R}$$

$$v_r = \frac{2v + \omega \mathrm{L}}{2R}$$

For understanding the odometry calculations let us consider a curved trajectory of the robot. Using the difference in wheel encoder pulses ($\Delta$t), wheel radius (R) and total number of pulses per rotation (n), the distance covered by the left and right wheels ($d_l$ and $d_r$ respectively) is calculated, whose mean results in the distance travelled by the robot ($d_c$). These values are used to calculate the position and orientation of the robot. The equations for these odometry calculations are given below.

$$Distance\ d = 2\pi R \frac{\Delta \mathrm{t}}{N}$$

$$\Delta \mathrm{t} = t_l - t_r$$

$$where\ t_l = left\ wheel\ ticks$$

$$t_r = right\ wheel\ ticks$$

$$d_c = \frac{d_l + d_r}{2}$$

$$\varphi + = \frac{d_r - d_l}{L}$$

$$x + = d_c \cos(\varphi)$$

$$y + = d_c \sin(\varnothing)$$

In this way, the odometry source gives the position ($x$, $y$) and orientation ($\varphi$) of the robot with respect to odom frame in the plane.

### 3.2 Mapping and Localization

ROS provides 2D laser-based SLAM (Simultaneous Localization and Mapping) packages in various implementation like Google cartographer, hector slam and gmapping. We selected gmapping due to its easy implementation and accurate mapping. The GMapping algorithm is a laser-based SLAM algorithm for grid mapping [6, 7]. This approach uses a particle filter in which each particle carries an individual map of the environment. The number of particles is reduced using several adaptive techniques to learn the grid maps (Fig. 3).

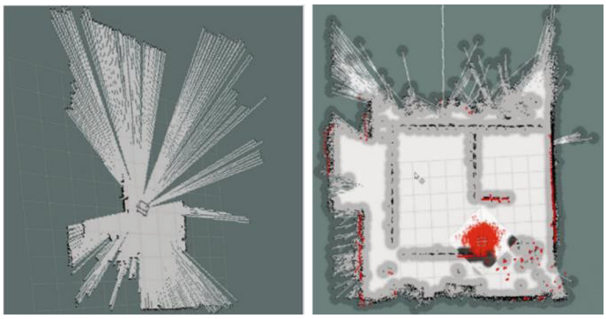

**Fig. 3.** Mapping and localization

Gmapping proposes an approach to compute an accurate proposal distribution taking into account not only the movement of the robot but also the most recent observation. This drastically decreases the uncertainty about the robot's pose in the prediction step of the filter. The "slam_gmapping" package subscribes to laser scan and odometry data to estimate the robot's pose with respect to the odom (initialised at the beginning of the gmappping process) and provides the map to odom link as an output. The 2D occupancy grid map is represented as a regular grid of cells, where the value of each cell encodes a probability of its state as free, occupied, or undefined i.e., unmapped, calculated using a bayesian approach. The static map created is saved and loaded through "map_server" package for further process of localization. The "amcl" package is used for localizing the robot through Adaptive Monte Carlo Localization (AMCL) algorithm [8]. It generates random possible poses of the robot and depending on the readings of laserscan and odometry information, the poses are concentrated in one point when the robot starts moving which indicates that the robot has been localized.

### 3.3 Autonomous Navigation

The move_base node provides a ROS interface for configuring, running, and interacting with the navigation stack on a robot. The main function of this node is to move the robot from its current position to a goal position. The goal is sent to the global planer which calculates a safe path to the goal pose. The path is calculated before the robot starts moving. It calculates the path according to static map and does not take sensor readings into account. The path is then sent to the local planner which provides the velocity commands to move the robot.


**Fig. 4.** DWA planner algorithm

The local planner monitors the odometry and LiDAR data to generate collision free local plan. It can recompute the robot's path on the fly in order to keep the robot from striking obstacles. DWA local planner was employed for this application, whose algorithm is briefed in Fig. 4. A set of parameters are configured through yaml files for running move_base node. Fine tuning of these parameters is essential for maximizing the performance of the navigation stack [9]. The recovery_behaviours node provided by move_base enables the robot to recover from a navigational failure.

## 4 Robot Manipulation

### 4.1 MoveIt!

MoveIt provides fundamental features for manipulation in ROS [10]. It offers a library of robotic capabilities for manipulation, motion planning, control, and mobile manipulation. "move_group" is the primary node of MoveIt that serves as an integrator, pulling all the individual components together to provide a set of ROS actions and services for the user. It is configured using the ROS parameter server from where it gets the URDF for the robot. While giving a native implementation of forward kinematics, MoveIt uses a plugin-based architecture to solve inverse kinematics. MoveIt setup assistant is designed to guide users to import their robot and create a MoveIt package. After importing the robot model, the Default Self-Collision Matrix Generator searches for pairs of links on the robot that can safely be disabled from collision checking, decreasing motion planning processing time. The next step is to add a virtual joint to attach the robot to the world. Since the arm is mounted on the mobile base, base_link was selected as the parent frame, and the base_frame of robotic arm was set as the child link. Following these two planning groups were added, one as arm_group describing all the 5 DoF joints and other as gripper group corresponding to end effector joints. The 'kdl_kinematics_plugin' was selected as the kinematics solver. Two robot poses were added for pick and place positions in order to test the motion of the robotic arm. As the robotic arm is connected to the mobile robot, two wheel joints of the mobile robot were added as passive joints indicating that these joints cannot be controlled directly by MoveIt. Thus, the configuration files for the robot have been generated by the setup assistant to start using MoveIt (Fig. 5).

**Fig. 5.** Picking an object

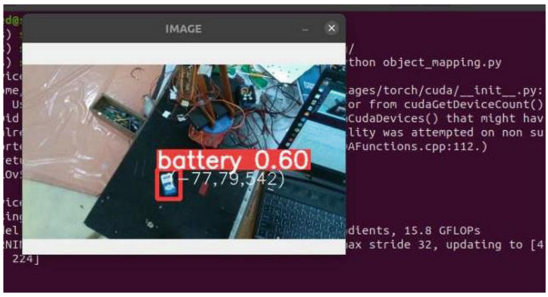


**Fig. 6.** Object detection

### 4.2 Pick and Place Operations

ROS provides OpenCV implementation of SURF algorithm [11] for object detection through "find_object_2d" package [12]. By setting ROS parameter 'subscribe_depth' to true, the 3D mode has been used to get three-dimensional co-ordinates of the object with the help of Intel Realsense D435i. The "realsense2_camera" package provides ROS node for using this camera. Through the GUI provided by the "find_object_2d" package, we marked the object of interest, and saved it for future detection. The detector node detects the object in camera images, estimates depth and orientation, and publish the position of the object over TF (Fig. 6).

We developed a python script to extract the co-ordinates of the object and used Python MoveIt interfaces to communicate with MoveIt. Also, an Arduino Mega micro-controller was used to control the motors connected to the joints of the robotic arm. Arduino script has been developed, creating a node that subscribes to '/joint_states' topic and the angles of all arm joints are provided to corresponding motors to perform pick and place operations.

## 5 Experiment and Results

### 5.1 Experiment for Obstacle Avoidance

In an experiment, the robot was given a 2D nav goal through Rviz leading the robot to generate the path and start moving towards the goal. We tested the obstacle avoidance of the navigation stack by introducing a walking human across the path followed by the robot. The result shows that the robot was able to avoid collision with the human by regenerating the optimal path towards the goal (Table 1 and Fig. 7).

### 5.2 Experiment for Picking the Object

In an experiment, we placed object at different heights from the ground such that the object is visible in the camera frame. The outcome demonstrates that our system successfully completes each scenario, picking the object at various heights and placing it accurately with more success rate around 10cm as tabulated in Table 2.

**Table 1.** Travel time to reach goal with and without obstacle along the path.

| Sl. no | Obstacle distance (m) | Travel time without obstacle (t) in seconds | Travel time with obstacle ($t_o$) in seconds | Avoidance time ($t_o$ - t) in seconds |
|---|---|---|---|---|
| 1 | 3 | 11.9 | 16.1 | 4.2 |
| 2 | 5 | 13.8 | 17.5 | 3.7 |
| 3 | 7 | 15.7 | 19.2 | 3.5 |
| 4 | 10 | 18.2 | 21.8 | 3.6 |
| 5 | 12 | 20.7 | 24.1 | 3.4 |
| 6 | 14 | 23.2 | 26.7 | 3.5 |
| 7 | 16 | 25.7 | 29.2 | 3.5 |

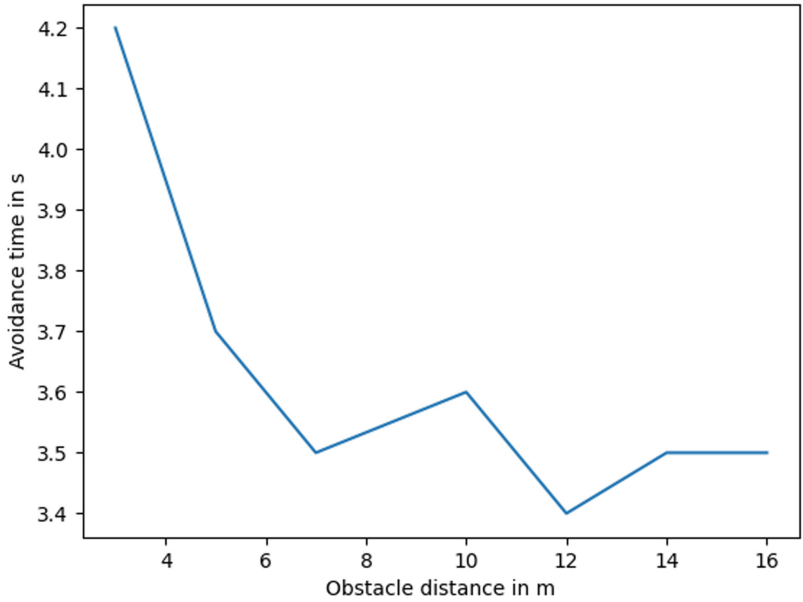


**Fig. 7.** Obstacle distance vs avoidance time

**Table 2.** Performance metrics for robot manipulation.

| Sl. no | Height level (cm) | No. of successful attempts (out of 15) | Success rate (%) |
|---|---|---|---|
| 1 | 5 | 9 | 60 |
| 2 | 8 | 12 | 80 |
| 3 | 10 | 13 | 86.67 |
| 4 | 12 | 13 | 86.67 |
| 5 | 15 | 11 | 73.33 |

## 6 Conclusion

In order to implement autonomous mobile manipulation in complex environment of mobile robot, this paper presents the usage of ROS navigation stack for autonomous navigation and also explains how to generate a MoveIt package using the setup assistant for easy implementation of complex motion planning of robotic manipulator. Using the concept of grid occupancy, a static map has been created utilizing lidar data and the robot has been localized using the Adaptive Monte Carlo Localization algorithm. The main challenge was in tuning the local planner for smoother path planning. However, this task was simplified by the 'rqt_reconfigure' plugin provided by ROS, which allowed the dynamic reconfiguration of the parameters. Three dimensional pointcloud has been successfully utilized to get the pose of the detected object for executing pick and place functions. Finally, the experimental findings demonstrate that the suggested approach is capable of successfully completing the autonomous mobile manipulation task in challenging environments.

## References

1. Sereinig, Martin & Werth, Wolfgang & Faller, Lisa-Marie. (2020). A review of the challenges in mobile manipulation: systems design and RoboCup challenges. e & i Elektrotechnik und Informationstechnik. 137. 1-12. https://doi.org/10.1007/s00502-020-00823-8.
2. Quigley, Morgan & Conley, Ken & Gerkey, Brian & Faust, Josh & Foote, Tully & Leibs, Jeremy & Wheeler, Rob & Ng, Andrew. (2009). ROS: an open-source Robot Operating System. ICRA Workshop on Open Source Software. 3.
3. S. Gatesichapakorn, J. Takamatsu and M. Ruchanurucks, "ROS based Autonomous Mobile Robot Navigation using 2D LiDAR and RGB-D Camera," 2019 First International Symposium on Instrumentation, Control, Artificial Intelligence, and Robotics (ICA-SYMP), 2019, pp. 151-154, https://doi.org/10.1109/ICA-SYMP.2019.8645984.
4. Maddalena Feder, Andrea Giusti, Renato Vidoni, "An approach for automatic generation of the URDF file of modular robots from modules designed using SolidWorks," Procedia Computer Science, Volume 200, 2022, Pages 858-864, ISSN 1877-0509.
5. Fabro, João & Guimarães, Rodrigo & Oliveira, André & Becker, Thiago & Brenner, Vinícius. (2016). ROS Navigation: Concepts and Tutorial. https://doi.org/10.1007/978-3-319-26054-9_6.

6. G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with RaoBlackwellized particle filters," IEEE Transactions on Robotics, vol. 23, no. 1, pp. 34-46, 2007.
7. A. Doucet, J. F. G. de Freitas, K. Murphy, and S. Russel, "Rao-Blackwellized particle filtering for dynamic bayesian networks," in Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI), pp. 176-183, Stanford, CA, USA, June 2000.
8. Sagarnil Das, "Robot localization in a mapped environment using Adaptive Monte Carlo algorithm," International Journal of Scientific & Engineering Research Volume 9, Issue 10, October-2018, ISSN 2229-5518.
9. Zheng, Kaiyu. (2017). ROS Navigation Tuning Guide. https://doi.org/10.1007/978-3-030-75472-3.
10. Chitta, Sachin. (2016). MoveIt!: An Introduction. https://doi.org/10.1007/978-3-319-26054-9_1.
11. Oyallon, Edouard & Rabin, Julien. (2015). An Analysis of the SURF Method. Image Processing On Line. 5. 176-218. https://doi.org/10.5201/ipol.2015.69.
12. M. S. Ruiz, A. M. P. Vargas and V. R. Cano, "Detection and tracking of a landing platform for aerial robotics applications," 2018 IEEE 2nd Colombian Conference on Robotics and Automation (CCRA), 2018, pp. 1-6, https://doi.org/10.1109/CCRA.2018.8588112.