

Autonomous Mobile Service Robot

Madan HT¹, Vaseem Akram², Sumanth Reddy³, Mohan Gowda KN⁴ and Uday Kumar LD⁵
¹⁻⁵REVA University, Bengaluru, India

Email: madanht@reva.edu.in, vaseemak4feb@gmail.com, sumanthavaji12@gmail.com

Abstract—Presented work aims to develop an autonomous mobile service robot which is able to navigate from the source to destination along with G-mapping of the given unstructured environment and it is able to localize itself in the generated map by using SLAM technique. This helps the robot to plan an efficient path to navigate to the goal position along with avoiding the obstacle interference. This is achieved with the help of ROS navigation stack.

Robotic Operating System(ROS) is a framework which has been used to manage the robot's operation, tasks, motion. The navigation of the robot is generally divided into global motion planning and local motion control to navigate and avoid the obstacle present in the generated path.

In simulation 3D simulator GAZEBO has been used to simulate the environment and the robot. Simulated LIDAR is used to create the global map of the unknown environment which is analysed and compared with the actual environment using RVizTool.

Index Terms— ROS, Robotic Operating System, Autonomous Robot, Service Robot.

I. INTRODUCTION

Autonomous navigation means that a vehicle is able to plan its path and execute its plan without human intervention. In some case remote navigation aids are used in the planning process, while at other times the only information available to compute a path is based on input from sensors aboard the vehicle itself. An autonomous robot is one which not only can maintain its own stability as it moves but also can plan its movements. Autonomous robots use navigation aids when possible but can also rely on visual, auditory, and olfactory cues. Once basic position information is gathered in the form of triangulated signals or environmental perception, machine intelligence must be applied to translate some basic motivation (reason for leaving the present position) into a route and motion plan. This plan may have to accommodate the estimated or communicated intentions of other autonomous robots in order to prevent collisions, while considering the dynamics of the robot's own movement envelope.

It's a difficult task to achieve complete navigation autonomy. Perception, mapping, control, planning, and decision making are all part of the most generally used approach, which uses a modular structure for sensing and information processing. This method, however, overlooks the ability to comprehend the environment. For dynamic and noisy situations, a computing model with self-learning capabilities inspired by biological intelligence such as memorising, inferring, and experience updating is required to attain full autonomy in navigation.

People can share housework with smart service robots, reducing the strain on the owner and bringing a lot of convenience to people's life. Smart service robots, particularly for the elderly, provide ease and a lot of joy to their life. Furthermore, the smart home service robot's danger alarm feature can make people's lives at home

safer. As a result, intelligent home service robots are becoming increasingly crucial in people's daily lives. However, certain smart home service robot's functions aren't yet flawless, therefore more research and development is needed to create smart home service robots with more extensive capabilities. SLAM and navigation are used in a variety of contexts, including automotive robotics, factories and production lines, and service robots carrying and delivering things. Its technology is advancing at a rapid pace.

II. IMPLEMENTATION

The implementation of autonomous robot navigation on TurtleBot3 Manipulator uses ROS (Robotic Operating System) as a framework which manages the robotic operation, tasks and motion efficiently. The software framework is mainly divided into two sections known as Navigation and Arm manipulation. Multiple ROS nodes have been implemented to achieve the desired tasks mainly Navigation Stack and MoveIt library for navigation and arm manipulation respectively. The parameters of the navigation stack have been improved through automating the process using python script. Arm manipulation has been implemented to perform the pick-up and drop task. A detail explanation of navigation stack and arm manipulation will be explained in the further sections.

ROS provides developing tools such as 2D visualization tool known as rqt graph and 3D visualization tool known as RViz. These tools make it easy to visualize the state of the robot and the performance of the algorithm to debug faulty behaviour, record sensor data and arm joints data.

Entire project has been simulated in the open source simulation known as GAZEBO, which has ODE (Open Dynamic Engine) physics engine, process 2D and 3D rendering and support logic for sensors and actuators control.

Computer Parts	Specifications
CPU	i5-8265U CPU
GPU	Intel UHD
RAM	8 GB
Storage	100 GB
OS	Ubuntu 20.04 LTS
ROS	Noetic

Fig. 1. Computer Hardware Specification

A. Software structure of navigation system

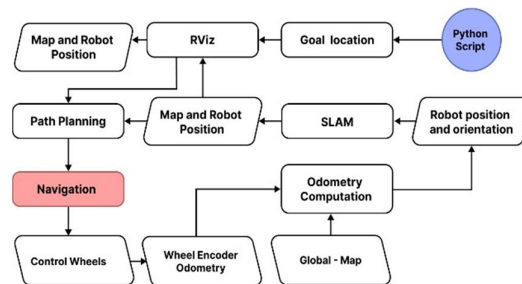


Fig. 2. Software Structure of Navigation System

The flowchart of the navigation system is shown in Figure 2. Initially the global_map is generated by using the LIDAR sensor and SLAM (Simultaneous Localization and Mapping) technique. Generated map will be saved in ROS map_server package which will be further used for navigation purpose in RViz tool (RViz tool is mainly used for analyzing and comparing the robot position and orientation with the real environment). Whenever the goal location is given the system starts to generate a suitable and efficient path of trajectory and send the motor

commands (i.e linear & angular velocity) to the two-wheel motor drivers. The odometry is continuously updated and used by SLAM for localization and mapping (local_map) as the robot is moving.

B. Software Structure of Arm Manipulation

The first is to use an Extensible Markup Language (XML) file to produce a Unified Robot Description Format (URDF) file as a visualization tool for robot modelling. You can use the ROS visualization tool RViz to construct a simple manipulator or custom created parts for your URDF file type. Later the Gazebo, a 3D simulator that can simulate the actual operational environment. Using Simulation Description Format (SDF) files and XML, the Gazebo simulation environment, like URDF, can be simply developed. To control numerous sensors and robots, Gazebo also offers ROS-Control and plugin features. At last MoveIt, It's an integrated library and effective method for manipulators that includes open libraries like the Kinematics and Dynamics Library (KDL) and the Open Motion Planning Library (OMPL), which aid in identifying the manipulator's various functions, such as collision calculation, motion planning, and Pick and place the selected objects in desired location.



Fig. 3. Basic Operation of Arm Manipulation

III. METHODOLOGY

Navigation System: The navigation enables a robot to move from its current position to the desired goal position on the generated map using the map, the robot's encoders, inertial sensor and distance sensor. The procedure for performing navigation is as follows:

1. Sensing
2. Localization/ Pose Estimation
3. Motion Planning

Sensing: On the map, the robot uses the encoder and the inertial sensor (IMU sensor) to update its odometry information, and measures the distance between the sensor's position and the obstacle.

Localization/ Pose Estimation: The current robot's localization/pose estimation is conducted on the global-map using the encoder's wheel rotation amount, the IMU sensor's inertia information, and the distance information from the sensor to the obstacle. In this paper we have used accurate and efficient pose estimating method known as particle filter localization method and Adaptive Monte Carlo Localization (AMCL).

Motion Planning: Motion planning, also known as path planning, produces a trajectory from the current position to the destination position on the map. The path plan developed comprises both global and local path planning for the entire map and smaller areas around the robot. We intend to employ the ROS route planning packages 'move_base' and 'nav_core', which are based on the Dynamic Window Approach (DWA), which is an obstacle avoidance method.

A. Navigation Stack

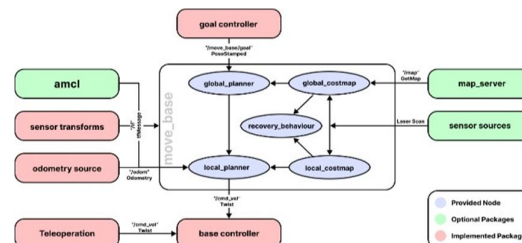


Fig. 4. ROS Navigation Stack Setup

Figure 3 shows the complete structure of navigation stack and it specifies the provided node by ROS, optional provided nodes and the implemented packages to run the stack.

Based on the robot's wheel encoders data "odometry source" package provide the localization information. Goal location is sent through a "goal controller" package for the robot to move from source to desired location. After

receiving the drive commands from the navigation stack “base controller” package is used to drive the robot with individual wheel velocity (i.e linear & angular velocity). To control the robot remotely “teleoperation” package is used to control the robot movement. Using the data from “odometry source” and “sensor source” package SLAM package can create the map of the given environment. “sensorsource” package is used to send the laser scan information to the navigation stack.

Odometry Package: The robot's odometry data is utilized in local path planning to construct a local path or avoid obstacles by obtaining information such as the robot's current speed.

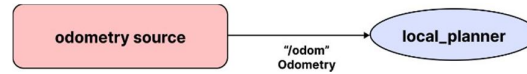


Fig. 5. Odometry Source Package

Coordinate Transformation: The ROS utilizes its coordinate transformations (TF) because the pose of the robot sensor varies based on the robot's hardware setup. This simply defines the sensor's relative x,y, and z coordinates as determined by the robot's odometry.

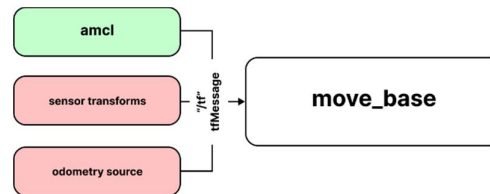


Fig. 6. Coordinate Transform Package

Distance Sensor: This is the measurement of the relative distance from the sensor. The robot's current pose is estimated using this distance sensor, and the robot's mobility is planned using AMCL (adaptive Monte Carlo localization).

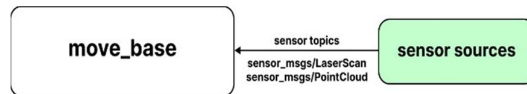


Fig. 7. Sensor Source Package

Map: Navigation uses Occupancy Grid Map which is generated through SLAM technique and saved in map_server package. The generated global-map is then sent to navigation stack whenever it is required.

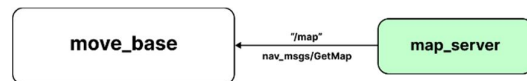


Fig. 8. Map Server Package

Target Coordinates: Target coordinates are specified by the user. Here we have used RViz to set the target coordinates. The target coordinates consist of two-dimensional coordinates (x, y) and direction θ .

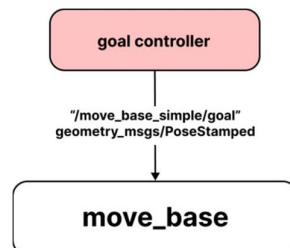


Fig. 9. Goal Controller Package

Velocity Command: The robot can be transferred to the destination coordinate by publishing the velocity command that follows the planned path.

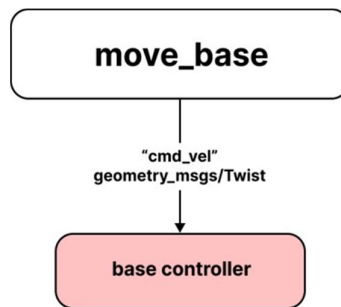


Fig. 10. Base Controller Package

Teleoperation package: Teleop package is used to detect the keyboard key press on remote computer and convert the key press into velocity commands.

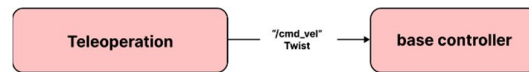


Fig. 3. Teleoperation Package

Arm Manipulation: Manipulators are robots that are used in industries to perform simple tasks. Open Manipulator by ROBOTIS is one of the manipulator the supports ROS, and has the advantage of being able to easily manipulate using the ROS package like RViz and library known as MoveIt.

MoveIt: It is an integrated library for manipulators that includes quick inverse kinematics analysis for motion planning, complex manipulation algorithms, robot hand control, dynamics, controllers, and motion planning, among other things. It's also simple to use even if you don't have much experience with manipulators because the GUI is available to help with the many settings required to utilise MoveIt!

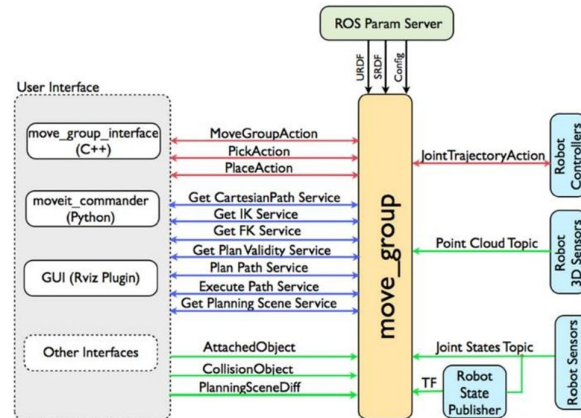


Fig. 11. Teleoperation Package

Using ROS actions and services, the 'move_group' node can exchange commands with the user. MoveIt! has created a service that allows more users to communicate with the 'move_group' node via the 'move_group' interface in C++ or the 'move_commander' interface in Python, as well as the 'MotionPlanning plugin to RViz'.

The URDF, Semantic Robot Description Format (SRDF), and MoveIt! configuration all send information about the robot to the 'move_group' node. The URDF that you developed will be used, whereas SRDF and MoveIt! will not. MoveIt!'s setup Assistant will be used to create the configuration.

Through ROS topics and actions, the 'move_group' node offers status and control of the robot and its environment. The controller uses the 'FollowTrajectoryAction' interface to tell the user about the robot status, while the joint state uses the 'sensor msg/JointStates' message, the transformation information uses the tf library, and the joint state uses the 'sensor msg/JointStates' message. The 'planning scenario' also provides the user with information about the environment in which the robot is operating as well as the robot's status.

The 'move_group' provides a plugin function for extensibility, and an open source library allows the user to apply numerous functions (control, path creation, dynamics, etc.) to their robot. The MoveIt! plugin is a fantastic library that has already been proven by a large number of people, and a number of newly developed open source libraries will be available soon. Those libraries include Open Motion Planning Library (OMPL), Kinematic and Dynamic Library (KDL), and A Flexible Collision Library (FCL).

TurtleBot3 Manipulator: TurtleBot3 is a ROS-based mobile robot which is compact, inexpensive, and programmable used in education, research, recreation, and product prototyping. The objective of TurtleBot3 is to drastically reduce the size of the platform and cut the price without compromising performance or quality, which allows possible upgrades. Depending on how you reconstruct the mechanical components and employ optional parts like the controller and sensor, the TurtleBot3 can be customised in a variety of different ways.

The TurtleBot3 uses SLAM, Navigation, and Manipulation as its fundamental technologies, making it ideal for home service robots. The TurtleBot can construct a map and drive around the room utilizing SLAM (simultaneous localization and mapping) algorithms. It may also be operated remotely via a laptop, joypad, or Android phone. By adding a manipulator like OpenMANIPULATOR to the TurtleBot3, it may also be used as a mobile manipulator capable of manipulating an object. The TurtleBot3 Waffle and WafflePi are both compatible with the OpenMANIPULATOR.

IV. SIMULATION RESULTS

Launching the robot in simulation world. Loading TurtleBot3 OpenManipulator into the house.

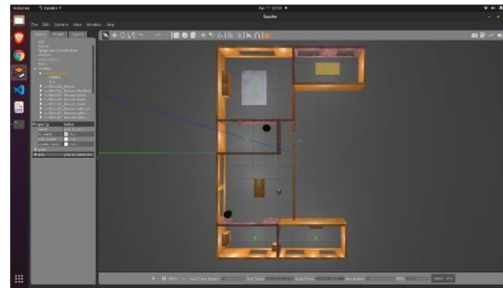


Fig. 12. TurtleBot3 Open Manipulator Launched in Environment

When the robot moves, the map is drawn using the robot's odometry, tf information, and sensor scan information. The RViz contains this information. The map is saved in the directory where 'map_saver' is currently running. The map is two-dimensional Occupancy Grid Map (OGM) where white is the free area in which the robot can move, black is the occupied area in which the robot cannot move, and grey is the unknown area.

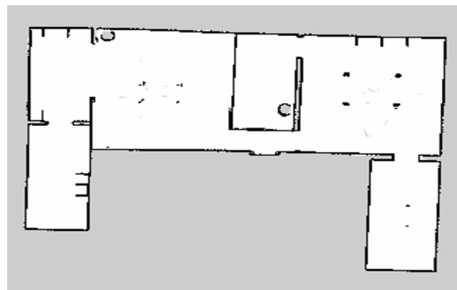


Fig. 13. Global Map Generated Using SLAM Technique

Now in RViz menu, navigation goal is selected which appears as large red arrow. This specifies the goal destination to the robot. The root of the arrow is the 'x' and 'y' position of the robot, and the orientation pointed by the arrow is the 'theta' direction of the robot. Now the robot is navigating to the place where the object will be placed.

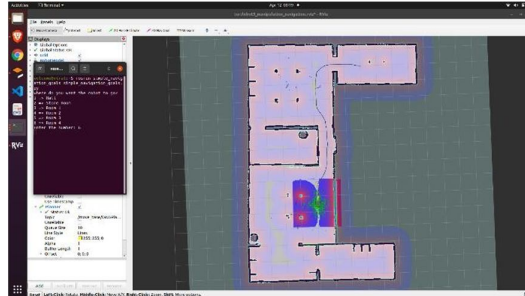


Fig. 14. Robot Navigation

Now the robot is controlled for detecting the object and will pick-up the targeted object using OpenManipulator grippers which uses move_group package that provides a GUI to control the manipulator.

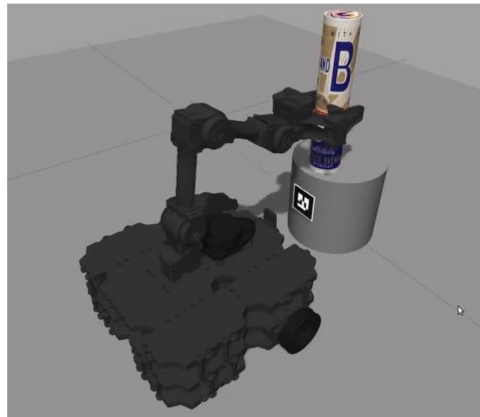


Fig. 15. Robot Picking Up the Object

V. CONCLUSION

In this paper we have described the implementation of ROS Navigation Stack and arm manipulation on a TurtleBot3-OpenManipulator. We have given a detailed information and specification for software requirements. In terms of simulation we have described navigation stack package to achieve navigation and move_group package to control arm manipulation.

An important component needs to be improved is to automate the arm for object detection on its own after reaching the goal position.

REFERENCE

- [1] Hafiq Anas and Ong Wee Hang, (2021) An implementation of ROS Autonomous Navigation on Parallax Eddie platform, Computer Vision and Pattern Recognition, arxiv.org, 10-11.
- [2] Zhang Zhaohui, Mei Xuesong, Bian Xu, Cai Hanghang, Ti Jian, (2016), Development of an Intelligent Interaction Service Robot Using ROS, IEEE Explore, 1739-1740.
- [3] Safdar Zaman, Wolfgang Slany, Gerald Steinbauer, (2011) ROS-based Mapping, Localization and Autonomous Navigation using a Pioneer 3-DX Robot and their Relevant Issues, IEEE Explore, 2-5.

- [4] Rajesh Kannan Megalingam, Chinta Ravi Teja, Sarath Sreekanth, Akhil Raj, (2018), ROS based Autonomous Indoor-Navigation Simulation Using SLAM Algorithm, International Journal of Pure and Applied Mathematics, 200-203.
- [5] David Portugal, Andre Arauj and Micael S Couceiro, (2019), A Guide for 3D Mapping with Low-Cost Sensors Using ROS, Robot Operating System (ROS), 3–23.
- [6] Swas Oajsalee, Suradet Tantrairatn, Sorada Khaengkarn, (2019), Study of ROS Based Localization and Mapping for Closed Area Survey, IEEEExplore, 25-27.
- [7] Ruchik Mishra, (2018), ROS Based Service Robot Platform, 4th International Conference on Control, Automation and Robotics, 56-58.
- [8] Sang Young Park and Gui Hyung Lee, (2017), Mapping and Localization of Cooperative Robots by ROS and SLAM in Unknown Working Area, IEEEExplore, 858- 861.
- [9] Jiansheng Peng, Hemin Ye, Qiwen He , Yong Qin, Zhenwu Wan, Junxu Lu, (2021), Design of Smart Home Service Robot Based on ROS, Hindawi Mobile Information Systems, 4- 8.