Manipulator Mounted Mobile Robot

Shobuj Paul
B.Tech in Mechanical Engineering
National Institute of Technology Karnataka, Surathkal
New Delhi, India
shobuj.191ch048@nitk.edu.in

***Abstract*—The project is a full software and theoretical framework for the modelling, simulation and control of a differential drive mobile robot mounted with two robotic manipulators.**

***Index Terms*—manipulator, navigation, control, robot**

I. Introduction

Mobile robots and manipulators are some of the most common class of robots used for industrial automation and research. Manipulators mounted on mobile robots could prove to be extremely effective in terms of both speed and dexterity in tasks in factories, construction, labs and medical suites. We have used 3D CAD software to create the model for the robot and imported it to ROS. Then we have utilised freely available packages such as the navigation stack for navigation of the mobile robot and Moveit! motion planning framework for control of the robotic arms.

II. Software Tools Used

- Autodesk Fusion360 was used to CAD modelling of the manipulator and wheeled robot.
- ANSYS Student 2021R1 used for structural analysis of manipulator assembly parts
- For the URDF (Unified Robot Description Format) files, fusion2urdf plugin was used to convert CAD model to stl meshes and generate files for a model.
- Gazebo 11 was used as a simulation environment to test the control of the bot using a physics engine.
- ROS Noetic was used as a software framework for simulation and control of the robot.
- Navigation stack used to create maps using and pass commands to move_base to for collision free motion.
- Moveit! motion planning framework was used to solve for the inverse kinematics of the manipulator and pass joint commands using either C++ nodes or Python scripts.

III. Theory

A. Control and Navigation of Mobile Robot

A differential drive model was used for the bot which was mounted on non-orientable wheels and the difference in wheel velocities is used for the motion of the robot.

$$\dot{x} = v \cos\theta \tag{1}$$

$$\dot{y} = v \sin\theta \tag{2}$$

$$\dot{\theta} = \omega \tag{3}$$

Where v and ω are the inputs for the controller. This is a simplified model of the differential drive bot where only the speed v and angular velocity ω are of concern, but in general the only controllable inputs are the velocities of the wheel which can undergo linear transformation to give the simplified inputs.

$$v = \frac{v_l + v_r}{2} \tag{4}$$

$$\omega = \frac{v_l - v_r}{2L} \tag{5}$$

Kalman Filters along with Adaptive Monte Carlo Localization (AMCL) is used to create the map of the robot enviroment, packages for both of which are provided by the ROS Navigation Stack. The robot can use global for planning in a path using A^* path planning algorithm and local path planners to avoid dynamic obstacles. Goal locations can be passed to move_base node using Python scripts.

B. Kinematics of Manipulator

Simple kinematics and geometry can be used to find out the position of the end-effector given link lengths, but inverse kinematics would need to be estimated due to the high complexity of the system. State-space for the manipulator is the vector of joint angles,

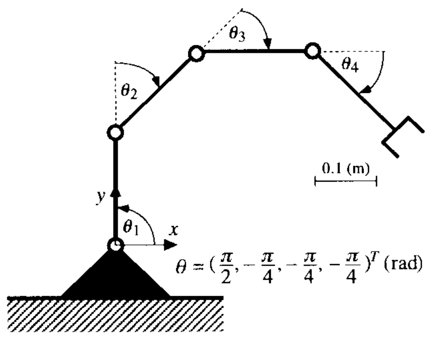
θ_1
θ_2
θ_3
θ_4
x
y
0.1 (m)
$\theta = (\frac{\pi}{2}, -\frac{\pi}{4}, -\frac{\pi}{4}, -\frac{\pi}{4})^T$ (rad)

Fig. 1. Representation of Manipulator

IV. Implementation

A. CAD Design

The model was designed on Autodesk Fusion360 with a parent-child inverted tree structure of the components used

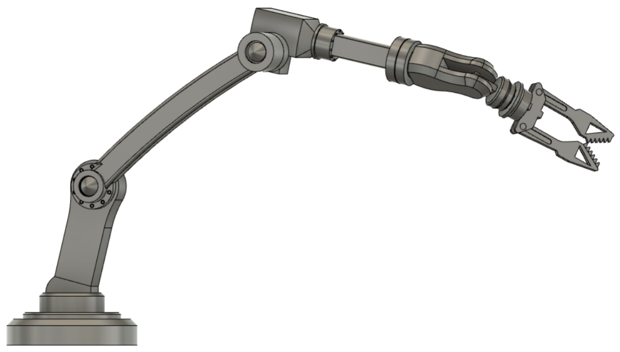

Fig. 2. 3D Model of Robotic Manipulator

for the assembly for proper transmission elements and transformations to be published via the TF package.

The manipulator consists of 6 degrees of freedom actuated by servo motors. Only revolute joints were used in the creation of the model, which translated to being continuous joints in the URDF xacro file. The gripper was designed with to pick up and sustain heavy structural loading.

The URDF package is then generated using fusion2urdf plugin and then moved to the ROS workspace. There a few changes are made to the default files, such as separation of Gazebo file elements and defining physical parameters like material and friction. A inertia free dummy link is introduced to avoid conflicts with KDL, which was made parent to the base_link and would be acting as the move_base for the robot.

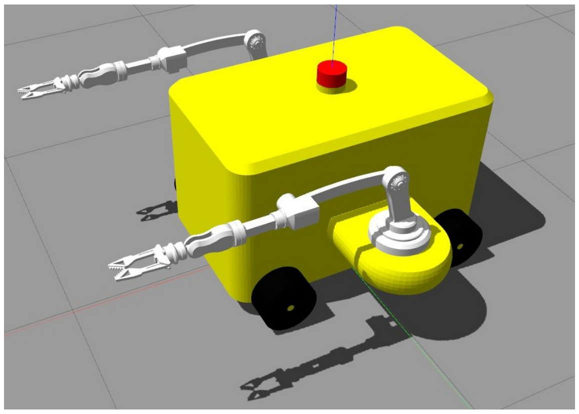

Fig. 3. Complete Model of the Robot

The entire assembly consists of base_link as the body, with manipulators mounted on the sides on wing-like extrusions. The whole system is mounted on 4 non-orientable wheels for differential drive kinematics.

A lidar-like sensor model is also attached on top of the robot for simulating a range sensor in ROS.

B. File Structure and Version Control

The project repository is hosted on GitHub and Git was used for version control of the ROS packages. The main branch consists of three description packages consisting of the URDF of the mobile robot, manipulator and joint URDF of the entire model. It also consists of configuration and navigation packages for motion and control of the robot.

Each package consists of folders separating the different aspects of the robot, URDF files, Gazebo files, YAML files and so on. Currently only empty world is used for the simulation but if required world files along with relevant launch files can be added.

C. Manipulator Control using Moveit!

Manipulator control is estabilished using Moveit! packages. At first the Moveit! Setup Assistant is run which takes the URDF xacro file of the robot as input for the robot description. Then basic physical properties such as collision matrix and virtual joint are defined. Then the move_groups for the manipulator is defined, which includes all joints except those associated with the gripper for the first group and the second as the gripper links themselves. Basic robot poses for default positioning are defined corresponding to the move_groups as well as the end effector for the robot model. Finally joint position controllers are set up for the robot arm, which control the joints of the robot arm given a goal location for the end effector.

Motion planning can be tested using RViz GUI by providing random goal locations and checking the validity of the planned paths. These goal locations can be provided to the move_group node using Python and C++ API for control of the robot in Gazebo.

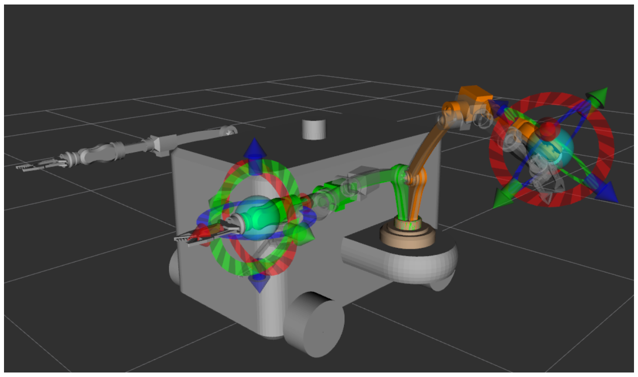

Fig. 4. Moveit Demo in RViz

References

- Ioan A. Sucan and Sachin Chitta, ”MoveIt”, [Online] Available at moveit.ros.org
- Open Source ROS Community, ”ROS Wiki” [Online] Available at http://wiki.ros.org/
- Choset, H., Lynch, K. M., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L., & Thrun, S. (2005). Principles of Robot Motion: Theory, Algorithms, and Implementations. MIT Press.

Project Repository

https://github.com/Shobuj-Paul/Mobile-Manipulator