Project Report: Planning and Decision making for Mobile Manipulator

Tanmay Chinchanikar (tchincha), Badal Arun Pardhi (bpardhi)

1 Abstract:

The aim of this project is to implement a master planner for a non-omnidirectional mobile manipulator that performs sequential pick and place tasks in a known environment. Being familiar with different types of planning algorithms for different purposes, our goal here is not only to integrate both the navigational and manipulator planning algorithms on a moving manipulator but also to test their practicality, efficacy and performance in a standard industry-level simulated environment. We perform navigation and manipulation tasks independently and sequentially due to computational limitations but with minor modifications, both planners can also be implemented in parallel without any loss of performance.

2 Introduction:

Over the last decade, mobile robots have been successfully applied and used in a wide variety of areas such as warehouses, military, industry and security environments. Path planning is one of the most fundamental problems that need to be resolved before mobile robots can navigate and explore autonomously in complex environments. Given a robot and its working environment, the mobile robot searches for an optimal or suboptimal navigation path from the initial state to the target state according to certain performance criteria and it then performs the required manipulation task. In this project, we aim to solve both these problems with state-of-the-art algorithms namely A* for navigation and RRT-connect/RRT-Star for arm manipulation. We also implement these planners in a simulated environment.

3 Method:

As mentioned in the previous section, we are using the variations of A* algorithm for solving the navigation problem of the mobile robot and the RRT-connect algorithm to solve the manipulation problem. The setup for the project is explained below.

3.1 Planning Representations:

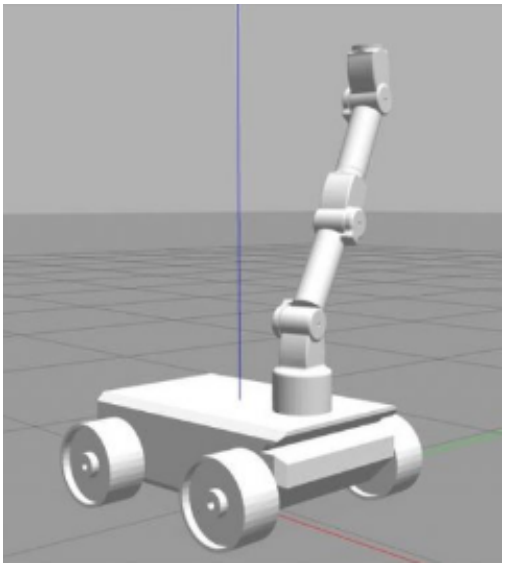
	States	Cost Function	Goal
Navigation	x, y, θ	Time	$x_{goal}, y_{goal}, theta_{goal}$
Manipulation	$\theta_1, \theta_2, \theta_3, \theta_4, \theta_5$	-	Goal configuration

3.2 Simulation Setup:

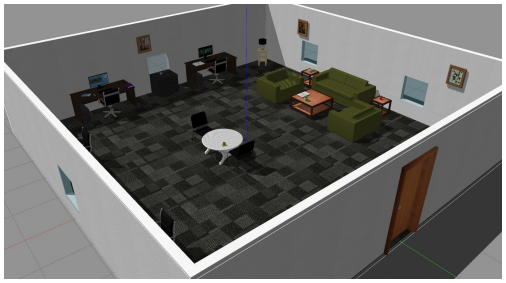
For the purpose of this project, we are using ROS (Melodic/Noetic) to integrate our planner with the mobile manipulator and simulate in a virtual known defined environment.

3.2.1 Mobile Manipulator specifications:

The robot in figure 2 is 4 wheel 5 dof standard manipulator. For the sake of simplicity, we treat the base of the mobile manipulator as a point robot (at the center) and while expanding the states use its dimensions for collision checking as part of planning. Since the dimensional and geometrical aspects of the mobile manipulator are not really a focus of this project, the physical CAD model was directly built from the existing available resources. The URDF files for building the mobile manipulator were imported from 1.
Dimensions of the mobile manipulator:
Length (L) = 0.65 m

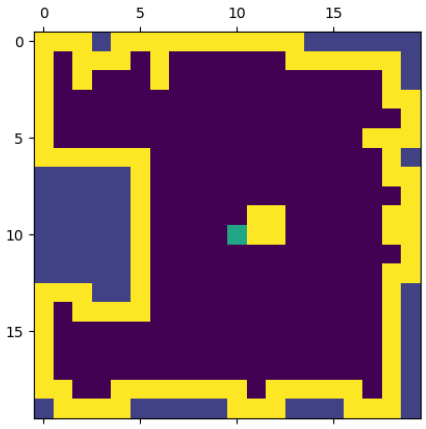


(a) CAD model of Mobile Manipulator



(b) Example of Gazebo World

Figure 1:



(a) Example of Gazebo World

Figure 2: 2D cost map example

Width (W) = 0.6 m
Height (Including manipulator length) = 1.2 m

3.2.2 Virtual Environment:

Both the navigational and manipulation planners are environment independent and can be used in any known environment. We used our own gazebo world to first test the validity of our planner and then implemented the planner on more advanced and complicated worlds. An example of one such complicated environment is shown in figure 1b. The Violet area in the figure represents the free area. The yellow region is the explored obstacle region whereas the blue region is the region that cannot be accessed by the robot. Also, the green cell represents the position of the robot on the map.

3.3 Graphs and Planners:

We are using standard X-connected grid-based cell decomposition graphs for solving our navigation Problem and a Tree based graph for solving our RRT-connect-based manipulation problem. By default, the 3D

Gazebo World does not provide a 2D cost map as well as a 3D octomap/pointcloud that can be directly used as inputs for our planners. Therefore, before moving forward with the planners, we first extract 2D (top view at a certain height) and a 3D cost map that can represent our world in a discrete manner which provides a base for the planning algorithms.

With a resolution ranging from 0.1-0.5 (1 pixel = 100-500 cm) depending on the map size, we use the Plugin from 3 to extract a 2D cost map at a certain height. The default height for creating the 2D cost map is 0 from the ground but can be modified if required. The output from the plugin is the .pgm format output which then needs to be converted to .txt format before being parsed into the code as the standard 1D array representing the 2D map. The final 2D cost map of the example map in figure 1b is shown in figure 2a.
In a similar way, we extract octomap from the gazebo world using OpenMace's method mentioned in 4.

3.3.1 Navigation:

Once we obtain a 2D map of the gazebo world, we implement variations of the A* algorithm to find the optimal path from our initial position to the goal position. Our planner is a generalized version of the A* algorithm which can perform all sorts of variations including a policy search, Anytime A*, Weighted A* and regular A*.

Since our mobile manipulator is not an omnidirectional robot, we consider only 3 motion primitives while transversing. In other words, we perform a constrained 3D A* over a 2D cost map. An example of motion primitive for 2D A* and constrained 3D A* at a particular state is shown in figure 4. Figure 4a shows the motion primitives for regular 2D A* whereas figure 4b and 4c represent motion primitives for 3D A* when the robot is facing North-east and North respectively. The red arrow is the current facing of the robot and green arrows are the possible motion primitives. Along with this constraint, we also add a small modification of penalizing the angular movement of the robot. Every time the robot has to rotate, the cost is 1.5 times more than the cost of going straight. This helps the robot in prioritizing the straight path as much as possible over taking turns repeatedly. Since we do not have variable costs over the map right now, we believe it is not very effective and makes it difficult to visualize the effect of adding penalization over angular movement.

We also use our very own collision checkers for both 2D and constrained 3D A* algorithms. Even though they seem to create absurd paths while visualizing them on the cost map, they make the actual path on the gazebo simulation much better. Since we are using 3D A*, our heuristic cannot be a simple 2D distance-based heuristic. We use backward Dijkstra (2D A* with 0 heuristics) and find the cost obtained from that as our heuristic for the 3D A*. If we are performing 2D A* for finding the path (for comparison), we use Diagonal distance as our heuristic.

3.3.2 Manipulation:

For manipulation planning of 5-dof arm, we use conventional RRT-connect algorithm. We integrate our RRT-connect with the ROS using Moveit Motion Planning Framework. The MoveIt Framework has different pre-built functions that make the custom planner integration easy and convenient. In our case, MoveIt takes care of the self-collisions as well as the feasibility and the validity of arm configuration in the given environment. We follow the process mentioned in 5,6 and 7 to create a plugin that integrates our custom RRT-connect planner with the MoveIt collision checker.

In general, for manipulation planner (Variants of RRT) we only modify the collision checker as mentioned above and use our same exact self-developed planners from assignment 2 keeping the other changes as minimal as possible considering the time constraint. In the future, we hope to try different samplers and modifications to our algorithm.

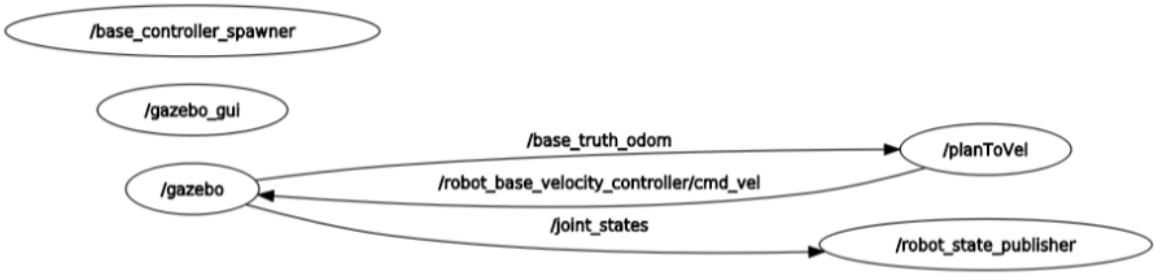
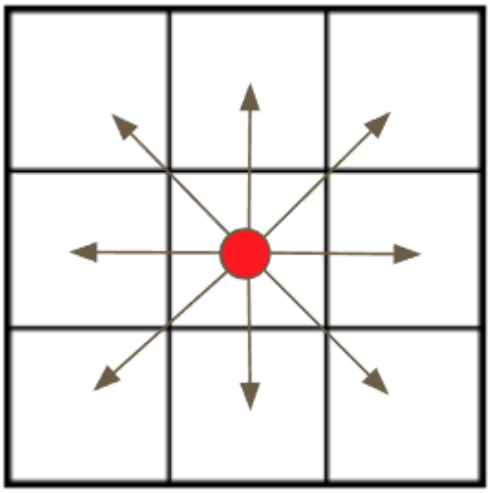
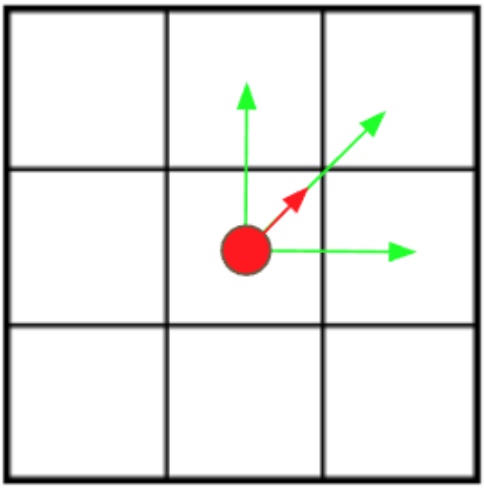


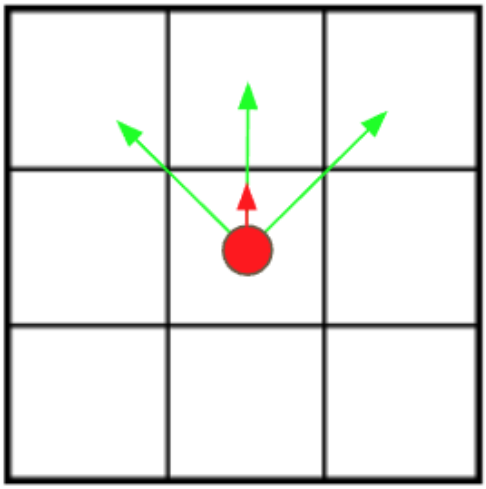
Figure 3: ROS Infrastructure



(a) Motion Primitives for 2D A*



(b) Motion Primitives for 3D A*With theta 45^0



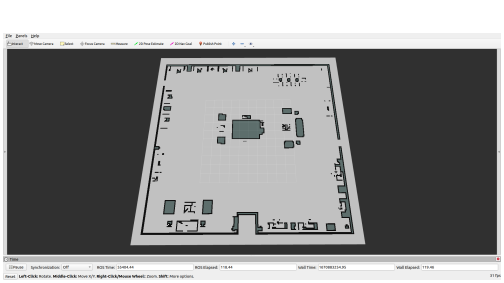
(c) Motion Primitives for 3D A* With theta 90^0

Figure 4: Motion Primitives for A*

4 Experiments/ Experimental Analysis:

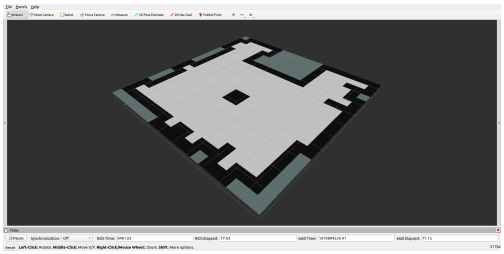
For Experimental analysis, we consider 1 small environment map with low resolution and 1 large environment map with higher resolution. Our aim was to study the path obtained by our constrained 3D A* against our regular 2D A* and see if the paths are actually viable and feasible in both cases. The virtual Gazebo environments and their cost maps are shown in Figure 5. With a resolution of 0.1, the dimensions of the map in figure 5a are 220*220 pixels.

(a) Gazebo Environment:Office Big



(b) Office big cost map in Rviz

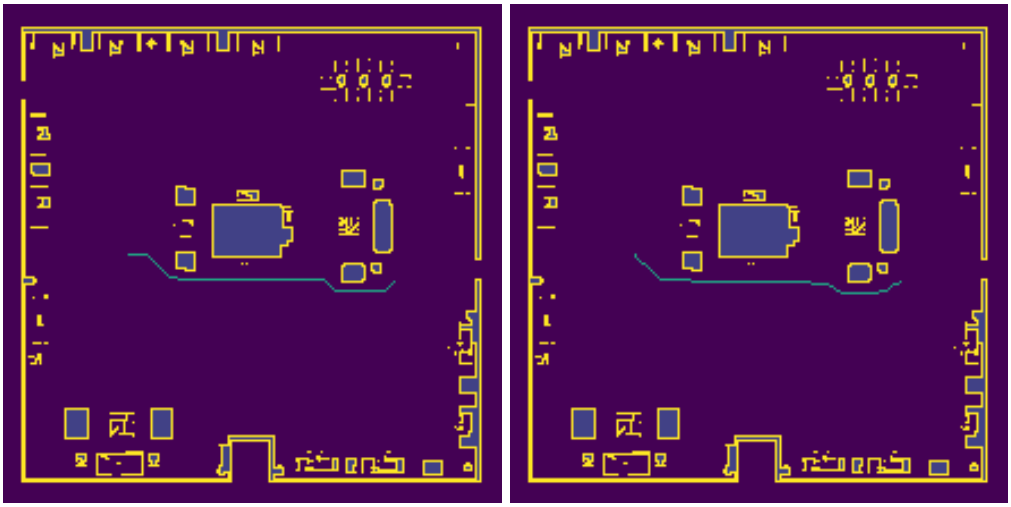
(c) Gazebo Environment:Office Small



(d) Office small cost map in Rviz

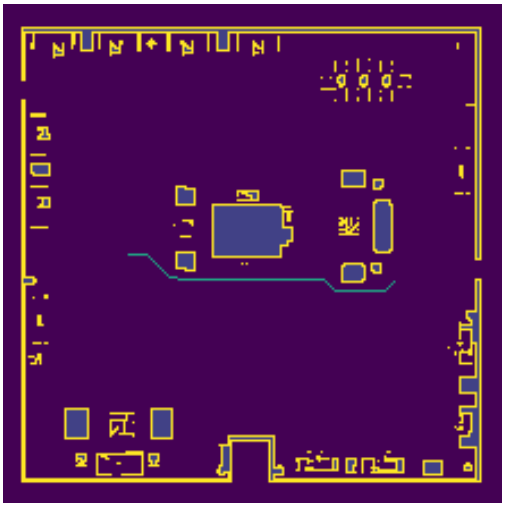
Figure 5: Gazebo Environments and their cost maps

	2d A*	3d A*
Time with Heuristic (Micro Sec)	2039	104066
Cost	127	128.5
States Expanded	1302	6383
Path Length	118	119
Time without Heuristic (Micro Sec)	51124	354573
Cost	127	128.5
States Expanded	20607	1550410
Path Length	118	119

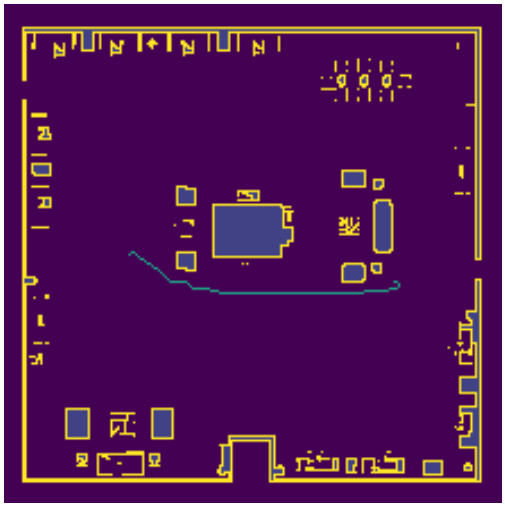


(a) 2D A*→R:(55,110,225^0) G:(172,122,45^0)

(b) 3D A*→R:(55,110,225^0) G:(172,122,45^0)



(c) 2D A*→R: (172,122,45^0)G:(55,110,225^0)



(d) 3D A*→R: (172,122,45^0)G:(55,110,225^0)

Figure 6: 2D vs 3D A* Path generation Example

Figure 6 represents the example plans from one point on the map to another while considering the initial and final heading angles of the mobile robot. The constructed paths are cost optimal and represent practical and feasible paths for the robot by considering the collision check. Figures 6a and 6c represent the forward and return paths planned by the 2D path planner which do not consider any angular orientations and can freely move in an 8-connected grid whereas figures 6b and 6d represent paths from the 3D planner that do consider the limitations of the vehicle and return a path that the robot can actually travel. From the return path figures, it can also be observed that 2D A* bluntly returns the same path whereas the 3D path considers the heading and plans accordingly. The numerical results and difference between the 2 methods are mentioned in the table above.

5 Conclusions:

Through this project we were successfully able to implement both a navigational planner and a manipulation planner. We were able to fulfill our base objective of solving these two problems separately and deploying them in a simulation environment such as Gazebo. Due to limited resources, time, and the number of contributors, our experiments are majorly focused on exploring the effectiveness of our navigation planner. Apart from 2D and 3D A-Star, we also experimented with weighted and anytime A-Star, however, due to smaller map size we did not see much difference in the time of execution. From the results for the manipulation planner, we observed that RRT-Connect was able to find a solution relatively faster (0.3 seconds) in contrast to RRT-Star which was taking 3 to 5 seconds to come up with the solution. We believe the reason for this difference lies in the configuration space within which the robot arm was moving. Since most of the space near the execution location was empty, RRT-connect was sampling a significantly lesser number of nodes and than RRT-Star. From the results of navigation, we observed that 3D A-star generally explored more nodes and took a relatively longer time to execute owing to its more complex collision checking and primitive sampling approach. In a practical setting, it is reasonable to adopt trajectories generated by the 3D A-Star planner since it does not require the robot to turn in place and provides a relatively smoother path.

Demo video: `https://tinyurl.com/mobileManipulator`

6 Work Breakdowns:

The work was distributed evenly between the two teammates. While one of us was more focused on developing the planner, the other was responsible for the integration of the planner with the simulation environment. Debugging the code was done collaboratively when it comes to both manipulation and navigation planners.

References

[1] How to Build a Simulated Mobile Robot Base Using ROS https://automaticaddison.com/how-to-build-a-simulated-mobile-robot-base-using-ros/

[2] Gazebo models and worlds collection https://github.com/chaolmu/gazebo_models_worlds_collection

[3] gazebo 2D plugin https://github.com/marinaKollmitz/gazebo_ros_2Dmap_plugin

[4] Generate Octomap from Gazebo world https://github.com/heronsystems/OpenMACE/wiki/Generate-Octomap-from-Gazebo-world

[5] Creating MoveIt Plugins https://github.com/ros-planning/moveit_tutorials/blob/master/doc/creating_moveit_plugins/plugin_tutorial.rst

[6] Developing a Planning Request Adapter http://wiki.ros.org/industrial_trajectory_filters/Tutorials/PlanningRequestAdapterTutorial

[7] Collision checking in robot arm using MoveIt! https://subscription.packtpub.com/book/hardware-&-creative/9781783551798/10/ch10lvl1sec74/collision-checking-in-robot-arm-using-moveit