Multi-Object Pick and Place using Mobile Manipulator

Group 9: Reagan Kan, Ayush Shrivastava

Introduction

The problem is a multi-object pick and place task in indoor 3D houses. The house setup is multi-room. A robot (a mobile manipulator) is placed in a room. It is given (x, y, z) coordinates of a few objects and the locations where each of these objects need to be placed. The robot needs to pick up an object and place it at its target location. Then, from the target location of object 1, the robot needs to go to the pickup location of the second object to pick it up and place it at its target location. This needs to be done for all objects. The robot needs to perform this task in a way such that it minimizes the distance travelled for the entire task. When the robot moves close to the object, it needs to extend its end effector to grab the object and pick it up.

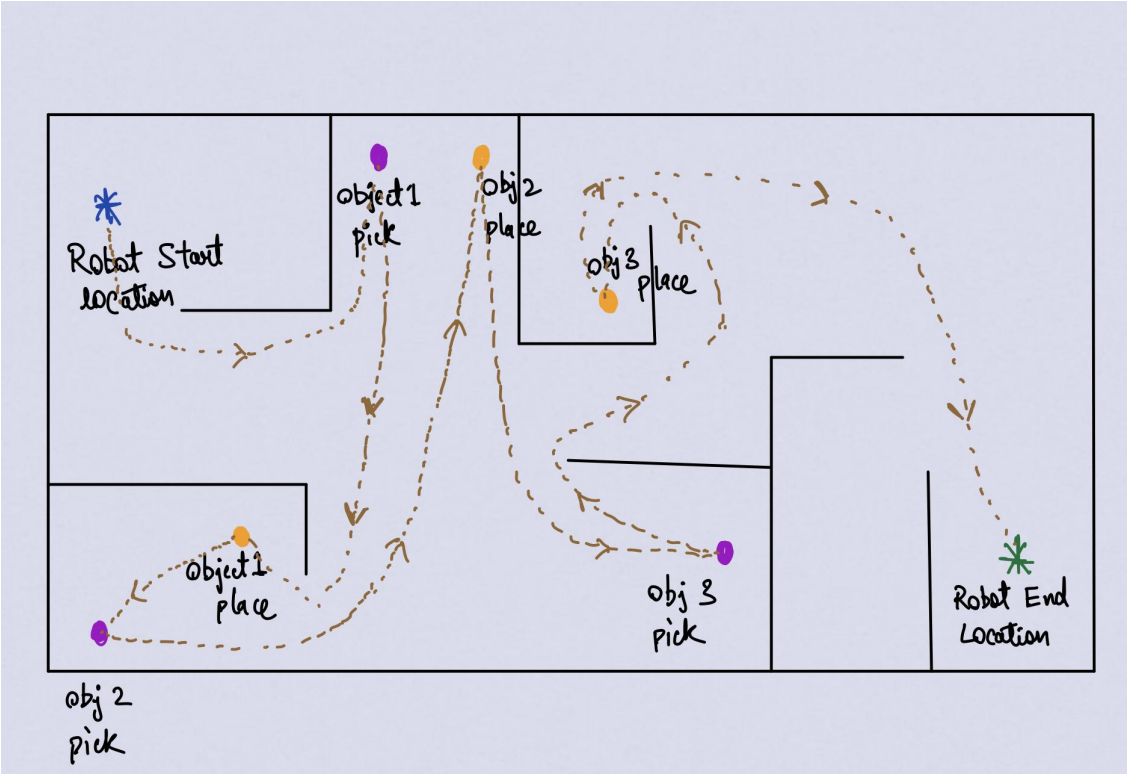


Fig1 : This figure explains the problem setup. Purple dots show the objects to be picked up. Organe locations show their respective drop-off locations. The robot starts at the blue star and needs to end the task by reaching the green star. The robot needs to plan in order to travel an efficient path minimising geodesic distance travelled (shown in brown).

Why is it not trivial?

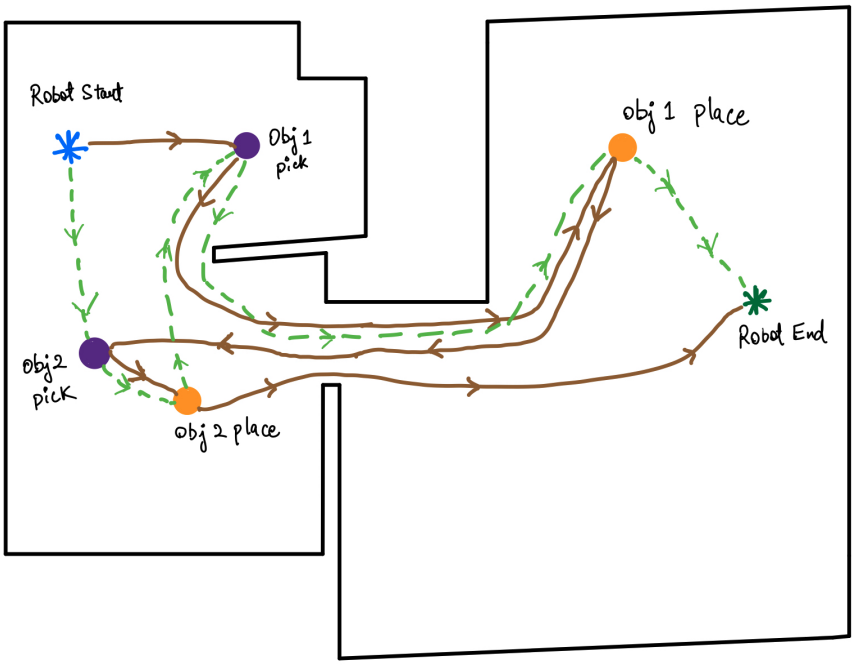


Fig 2: This figure shows the two paths that the robot can take to solve this task. Green path is efficient than the brown path and it requires planning to figure out which path is better.

In order to complete this task in an efficient way, the robot needs to plan using the geodesic distances from all nodes, i.e. robot start and end location, and object pick and place locations. The robot cannot solve this problem in a greedy way i.e. by travelling to the object pick location nearest to it in terms of both euclidean and geodesic distances. This can be understood by looking at the example shown in Fig 2. The robot starts in the top left corner at the blue star. It needs to pick 2 objects shown in purple and place them at orange locations. Object 1 is closer to the robot than Object 2. If the robot solves this task based on which object is closer to the robot, then the robot will follow the path shown in brown. But if we pay attention to where the objects are placed, their pickup locations and where the robot is supposed to go to in the end, we can see that the robot can travel a more efficient path shown in green. In order to figure out this green path, the robot needs to plan based on the geodesic distances to all the places in the house.

Related Work

Hornung et al. [1] successfully applied a modular planning framework to a harder version of the problem: a humanoid robot must move target objects in a cluttered indoor environment and occasionally clear obstacles out of the path. The framework periodically updates a representation of the room using state estimation sensors while planning, and possibly replanning, the trajectory for completing the task. Replanning occurs when unforeseen obstacles block a path, or the end effector drops an object. The authors use a camera for both robot localization, object detection, and grasp verification, but note that any sensor can be swapped in. The framework also includes a placement cost estimation step that samples from possible robot locations to determine potential locations for temporarily placing obstacles.

Batra et al. [2] is a positional white paper which talks about several aspects of the object rearrangement problem. They specify various types of goals, sensors and actuation suites that the task definition can consist of. We are using the GeometricGoal definition of the goal in our project.

Project contribution

This project provides a comparative analysis of three algorithms for planning under the context of a pick-and-place task inside a home environment. The results of this analysis can be useful for the development of robotic home assistants.

While the scope of this project is limited given the time constraints, research in this area is very exciting and leads to development of robots that can clean a cluttered environment. This can be done using learning by demonstrations or giving the end state of the environment using which the robot can reason about which objects need to be placed at their goal locations.

Approach

Methodology

Although the Fetch robot is in a 3D environment, assumptions are made to simplify the problem. Walls and obstacles are rectangular cuboids. When projected onto the 2D plane, they become rectangular polygons. Similarly, the fetch robot can configure its arm such that its widest area is the mobile base, i.e. when projected onto the 2D plane the robot is a circle. Finally, the items to be picked and placed are spheres. Next, we assume that, in addition to knowing the pick and place locations of all items, the robot has full knowledge of the environment in 2D. This means it knows the pose of its own body, all items, and all walls and obstacles vertices.

Now, the problem is a 2D planning problem; given the above information, compute the 2D waypoints that the robot must follow to successfully pick and place all items as efficiently as possible. We define an “efficient” trajectory to be one that requires minimal execution time. However, we do not explicitly optimize for that value and instead look at the minimum 2D

distance traveled. Note, this distance ignores distances traveled through arm movement and mobile base rotation.

Our approach comes in two parts: free location graph construction and item ordering.

The first step creates a free location graph where the nodes are 2D locations of interest (free locations, robot start/end locations, and item pick/place locations) and edges connect nodes that are visible to each other on the 2D map. Furthermore, the edges are weighted by the 2D euclidean distance between nodes.

There are many algorithms that could be leveraged for constructing this graph, such as Generalized Voronoi Diagrams (GVD), Visibility Graphs (VG), Modified Cell Decomposition (CD), and Rapidly-exploring Random Trees (RRT). We compare the effectiveness of the latter three algorithms because GVD provides free locations that are optimally far from obstacles which leads to slower robot paths.

Typically, CD returns a graph of cells with edges connecting adjacent cells. Since the nodes in the free location graph represent specific 2D coordinates for free locations, we modify the CD approach in the following ways. After a vertical line sweep to identify the cells, we consider the graph nodes to be the midpoints of the cell boundaries.

To determine the shortest item ordering, we store in a lookup table the shortest path between several pairs of points in the graph (robot start location + item pick location, item pick location + item place location, item place location + robot end location). This allows for the computation of the total distance for each item ordering.

Implementation

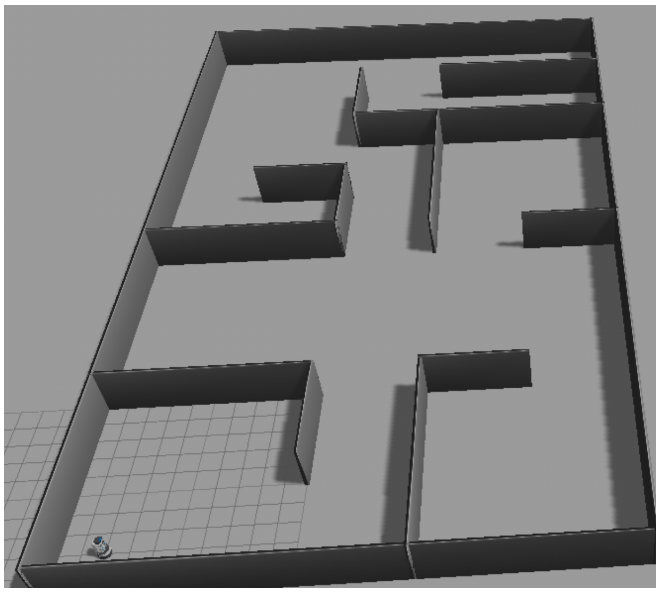


Fig 3: This is the top-down view of the house that was built in Gazebo and was used as a testbed for our experiments.

Since the class operated remotely this semester, we decided to simulate the robot using ROS and Gazebo rather than conduct real robot experiments. We constructed a Gazebo model for the 3D house with a sufficiently complex interior as shown in Fig 3. Our launch script loads the house and a Fetch robot into the purely geometric Gazebo simulator, e.g. there is no friction. Finally, we set up a publisher script to move the robot given a path of 2D waypoints. At any point in time, the robot is either rotating towards a target angle or translating towards a target 2D point. The script subscribes to the robot pose and periodically checks if it has reached its target. This setup allows us to provide paths generated by different algorithms on different arrangements of robot and item start/end locations.

We initially wanted to use MoveIt to simulate the robot arm grabbing the items as the robot moved between pick and place locations. However, we did not have enough time to fully figure out how. We include a video of the

For generating the robot free locations, we leveraged existing Python code for RRT and implemented VG and CD ourselves. Computations for the shortest path distance between two nodes were handled by Dijkstra's algorithm implementation from the Python NetworkX library.

Results

We tested the algorithms on 2-item pick-and-place tasks. While the same house model is used in each configuration, the robot start/end locations, along with the item pick/place locations are different between configurations. Below are the configurations. Red is the robot start location. Green is the robot goal location. Pick locations are blue and place locations are purple.

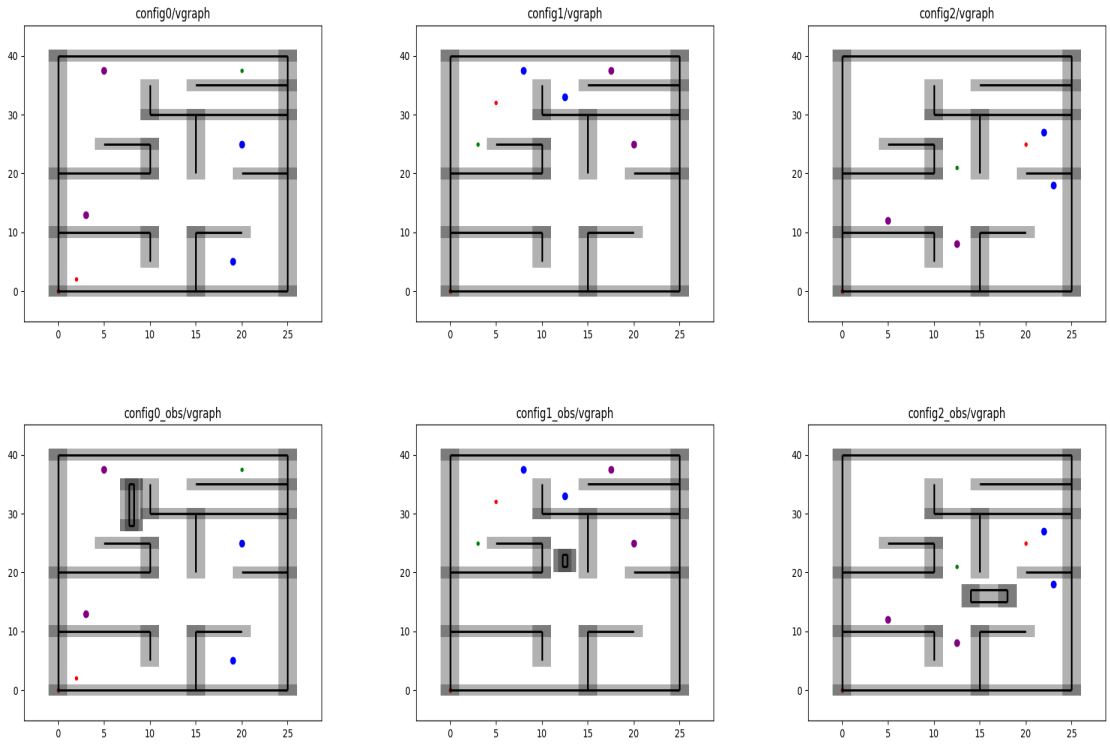


Fig 4: All six configurations used for evaluating approaches. Robot start/end locations are red/green. Item pick/place locations are blue and purple. Black lines are the walls and gray boxes are the expanded walls accounting for robot width.

Initially, we planned on evaluating the algorithms using the pick and place success rate and the execution time. Unfortunately, time constraints prevented us from executing all paths in simulation. We include a link to a few videos that we did manage to capture. Videos link: https://drive.google.com/drive/folders/1AJPYObzbhfAtBktkKVcS-cUC1sYBO6uj?usp=sharing.

We instead evaluate the algorithms using the average 2D euclidean distance across the six configurations. Below are the distances for the generated plans.

Distance (meters)	VG	CD	RRT
C0	135.966	141.576	128.650
C0 + obstacle	136.786	144.406	132.737
C1	128.371	117.75	110.263
C1 + obstacle	97.728	120.754	111.983
C2	72.596	75.356	72.289
C2 + obstacle	76.423	79.493	72.664
Average	107.978	113.222	104.764

Table 1: Total path distances for each approach-configuration pair.

Discussion

Initially, we expected VG to give the shortest paths. However, on average the RRT was the shortest. We think this is because RRT has many more nodes in the graph than VG, which allows more precise paths.

For future work, we would like to compare across more configurations and actually simulate the robot trajectories in Gazebo. Also, we would like to incorporate the arm movement in the simulation with MoveIt. Another direction would be to investigate other non-graph methods for this planning task.

Meta-learning

In the process of writing the code for visibility graphs and cell decomposition, we reviewed the finer details of the algorithms. For example, when working with visibility graphs, it is important to expand the convex hull of obstacles to account for the robot width. We learned how to compute many geometric properties, such as how to check for intersection of two line segments. Naturally, we gained experience with using Python libraries such as matplotlib, networkx.

Through working on the robot simulation in Gazebo, we gained familiarity with specific aspects of ROS. We built off what we learned from Homeworks 1 and 2 about ROS (setting up a ros package, adding scripts, etc.) and learned how to build a model, configure launch script to load

custom world with custom models, publish to gazebo with rospy, subscribe to topics other than odometry.

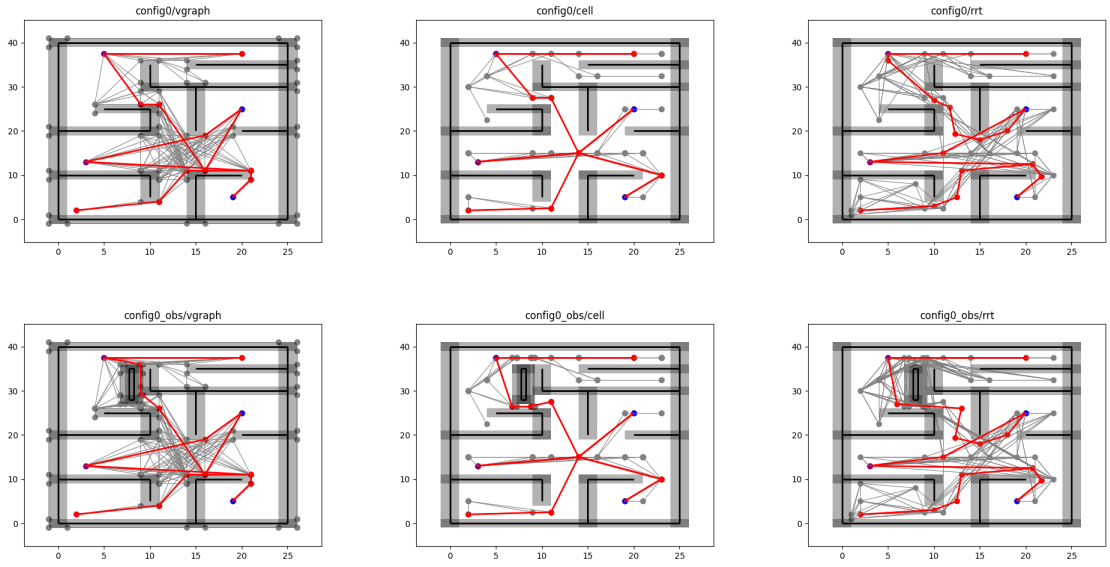
References

[1] A. Hornung, S. Böttcher, J. Schlagenhauf, C. Dornhege, A. Hertle and M. Bennewitz, "Mobile manipulation in cluttered environments with humanoids: Integrated perception, task planning, and action execution," *2014 IEEE-RAS International Conference on Humanoid Robots*, Madrid, Spain, 2014, pp. 773-778. doi: 10.1109/HUMANOIDS.2014.7041451. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7041451&isnumber=7041308

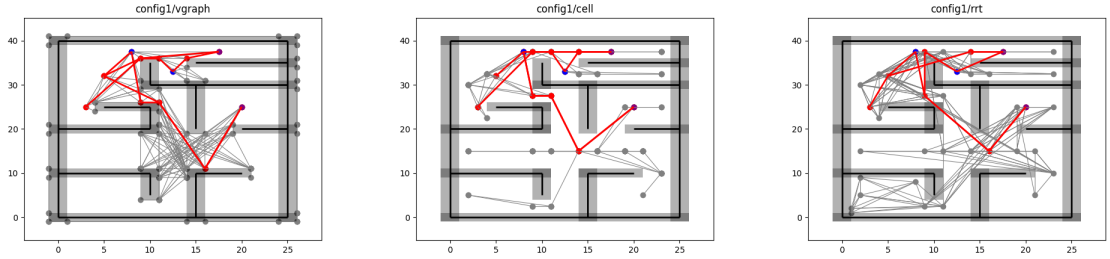
[2] Batra, Dhruv, Angel X. Chang, Sonia Chernova, Andrew J. Davison, Jia Deng, Vladlen Koltun, Sergey Levine et al. "Rearrangement: A challenge for embodied ai." arXiv preprint arXiv:2011.01975 (2020). URL: https://arxiv.org/pdf/2011.01975.pdf

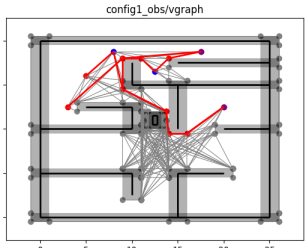
Appendix

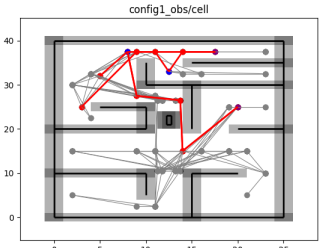
Configuration 0

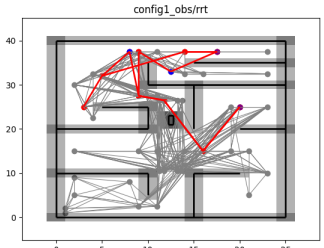


Configuration 1









Configuration 2