Generalizable Task Planning

Neel Kulkarni
nkulkar5@stevens.edu

Yi Guo
yguo1@stevens.edu

***Abstract* --** This research addresses the challenge of integrating motion planning and object recognition for a generalized task planning robot for home tasks. The importance of this problem lies in its potential applications in automation, where robots must handle tasks like stacking objects. This solution leverages motion planning and object recognition, simulated in PyBullet, to enhance the robot's ability to plan and preciseness in execution. The results demonstrate significant improvements in task accuracy and execution efficiency. The implementation details, source code, and videos are available on our GitHub: `https://github.com/kulnel2026/generalizable_task_planning`

I. Introduction

The ability for robots to generalize task planning for home tasks holds immense potential for improving efficiency and accuracy in various applications. Generalizing task planning grants robots the ability to adapt their behavior to a wide range of tasks and environments, rather than being limited to predefined actions, specific objects, and analytically defined transition models [1], [2]. Thus, robots capable of generalized task planning can significantly reduce human intervention [3], increase productivity, and handle tasks in hazardous environments.

Despite its potential benefits, several challenges persist. Traditional robotic systems often struggle with the integration of motion planning with obstacles and object recognition due to the complexity of real-world environments. Accurate motion planning requires sophisticated algorithms to navigate dynamic and sometimes unpredictable surroundings that are present in real-world scenarios. The objective of this paper is to create a learning-to-plan method that is able to generalize task planning with unseen objects, aiming to tackle the challenge of natural obstacles during home tasks. The importance of this research lies in developing a system that can integrate motion planning in environments with obstacles with object recognition to create a versatile and adaptive robotic solution.

Utilizing the PyBullet simulation environment, this method has two stages. The first stage extracts object-level segmentation from an RGB-D observation of the scene, which is then used to form predicates describing the scene, leading to the robot searching for skills required for the task. The second stage is a motion planning framework in which the robot plans a path around an obstacle between a start location and a goal location, using data from the RGB-D observation from the scene to plan a path around obstacles to stack different color blocks.

II. Problem Statement

The primary research problem addressed in this paper is the development of a robotic system capable of autonomously planning and executing complex tasks involving the manipulation of objects, specifically focusing on stacking blocks of different colors. The problem encompasses several critical aspects. Motion planning - the robot must be able to plan its movements accurately and efficiently in a dynamic environment and amongst obstacles. Object recognition and segmentation - for effective task execution, the robot must reliably recognize and segment objects from its surroundings. This includes distinguishing objects based on color, shape, and size, even under varying lighting conditions and partial occlusions. System integration - combining motion planning and object recognition into a cohesive requires the system to ensure seamless communication and coordination between different components to perform tasks accurately. By addressing the above aspects, the proposed solution aims to improve the robot's task execution accuracy and efficiency, thereby advancing the state-of-the-art in autonomous robotic systems.

III. Solution

This section presents our solution for developing a generalized task planning robot capable of autonomously stacking cubes of different colors with obstacles present. The proposed solution encompasses two main components: an integrated motion planning network and a robust task planning algorithm which integrates the motion planning algorithm with object segmentation, forming predicates describing the scene, and searching for skills necessary to carry out the plan.

3.1: Motion Planning Framework: RRT-Connect

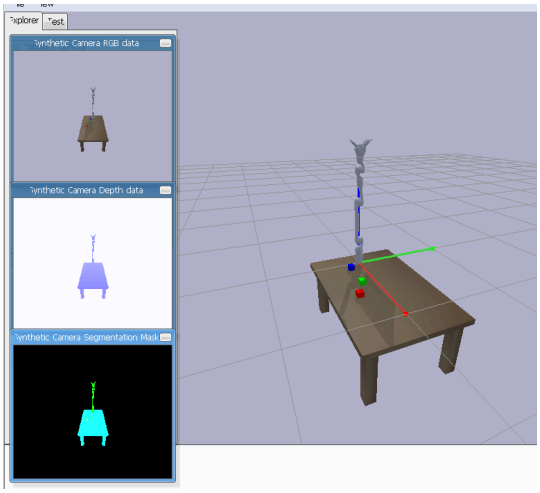


Figure 1: *PyBullet simulation setup. A robotic arm placed on the table. The cubes start on the left side of the table. On the left of the window are cameras; the top capturing RGB, the middle capturing depth, and the bottom segmenting objects based on the RGB-D data.*

In our project, the robot arm must grab a cube from the left side and move it to the other side of a table, navigating around obstacles in the middle of the scene, as shown in *Fig. 1*. Motion planning is critical for enabling the robot to navigate and manipulate such objects within its environment. Our solution utilizes the MoveIt framework for motion planning. MoveIt is a widely-used open-source software for mobile manipulation, integrating various tools for motion-planning, manipulation, 3D perception, kinematics, control, and navigation. MoveIt is used to plan the robot's path while reaching for the block and around the wall, leveraging the RRT-Connect algorithm, known for its efficiency in rapidly exploring search spaces, to generate feasible paths for the arm.

RRT-Connect (Rapidly-exploring Random Tree - Connect) is a variant of the RRT (Rapidly-exploring Random Tree) algorithm designed to improve the efficiency and performance of motion planning tasks. Unlike standard RRT, which grows from a single tree from start configuration, RRT-Connect simultaneously grows into two trees; one from the start and another from the goal, as illustrated in *Fig. 2*. In this case, the initial point is the location at which the robot picks up the cube. The goal point is a predefined point on the other side of the table. These trees attempt to connect with each other, significantly increasing the likelihood of finding a valid path in less time.

RRT-Connect best fits the task at hand in comparison to other algorithms. Standard RRT, while good for initial, simple exploration, is less efficient in complex environments seen in real-world scenarios compared to RRT-Connect. RRT* (RRT-Star), while effective in high-dimensional spaces and complex environments, comes with a higher computational cost and slower convergence time, which does not suit the speed needed to react to dynamic situations in real-world environments. Overall, we chose the planning algorithm, RRT-Connect, because it balances the strengths and weaknesses of these algorithms to best suit high-dimensional, dynamic situations.

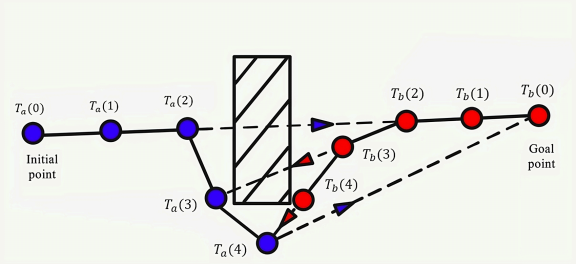


Figure 2: *Figure describing RRT-Connect. T_a (blue) and T_b grow from the initial point and goal point, respectively. The trees expand until they meet. The immediate nodes represent intermediate points in the trees with $T_a(0)$ being the initial point and $T_b(0)$ being the goal point. An obstacle is represented as a shaded rectangle, which the trees navigate around, the dashed line indicating the connection between the trees.*

3.2: Task Planning

For effective task execution, our solution integrates a multi-step plan. This process involves several key steps.

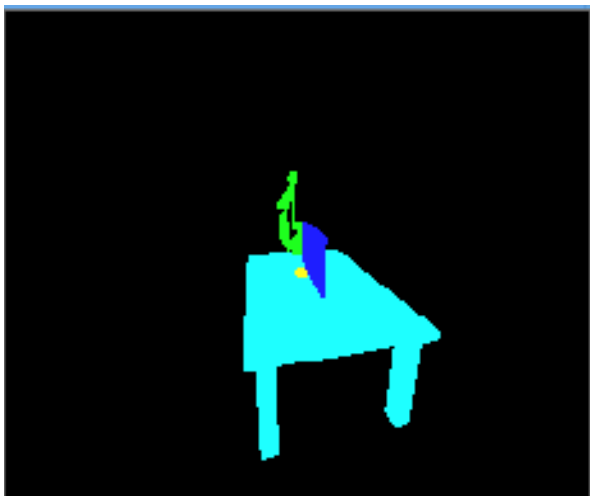


Figure 3: *Scene captured by the Object Segmentation Camera. The robotic arm is highlighted green, the cube is highlighted yellow, the wall is highlighted dark blue, and the table is highlighted light blue.*

Initially, a point cloud RGB-D image of the scene is captured, providing both color (RGB) and depth (D) information. The raw images are processed to segment and identify objects. This is achieved using deep learning models such as convolutional neural networks (CNNs). The goal is to accurately identify and separate individual objects, as illustrated in *Fig. 3*, even when they are partially occluded or in varying lighting conditions. From the segmented regions, key features such as color, shape, and size are extracted. These features are essential for differentiating between objects and for the subsequent task planning stages. Once the objects in the scene are recognized and segmented, the next step is to generate a list of parameterized skills that the robot can use to interact with these objects. A parameterized skill is represented as a set of four functions: $s = (L_P, L_E, \pi, f_T)$. L_P represents the set of logical preconditions required to execute the skill, L_E represents the set of expected logical effects, π represents a visuomotor policy (Reinforcement learning algorithm), f_T represents the termination condition of the policy. The robot formulates a task plan based on the identified objects and their positions, using a breadth-first search algorithm to search for a sequence of skills to transform the start state into the goal state.

Algorithm 1 Task Planning

Input: RGB Image *I*
Output: Path *P*
1. Extract Point Clouds *C* from RGB-D Image *I*
2. Perform Object Segmentation on *C* to generate segmentation masks $M = \{m1, m2, ..., mn\}$
3. Generate Object-Level Features $F = \{f1, f2, ..., fn\}$ from *M*
4. For each feature *fi* in *F* do
5. Generate Predicates *pi* in the list $q = \{p1, p2, ...\}$ representing the initial state of the object
6. Define Task Goal g as a set of predicates $g = \{p1, p2, ...\}$
7. Initialize Task Planner with Start State *q* and Goal State *g*
8. Search for a sequence of skills $S = \{S1, S2, ..\}$ that achieves goal g
9. If task planning is successful then
10. a. Initialize Motion Planner with Start State *q* and Goal State *g*
11. b. Plan a path *P* around obstacles using object-level features *F*
12. c. Execute the sequence of skills *S* along the path *P*
13. Return Path *P*

IV. Results and Analysis

This section provides numerical results to support the proposed simulation workflow. The results were obtained through simulations conducted in the PyBullet environment with the integration of MoveIt for motion planning. The primary objective of the simulation was to evaluate the performance of the robot in navigating a path around a variety of objects to get the cube to the other side. There were two simulations conducted: one obstacle and multiple obstacles blocking the path of the arm in stacking the cubes.

4.1: Simulation Setup

The simulation was run for a total of 25 iterations to ensure statistical significance. The parameters for the simulation were set as follows:

Number of Simulations (NUM_SIMS): 25
Robot Model: Kinova Gen3
Environment: PyBullet
Motion Planner: RRT-Connect
Objects: Cubes of different colors and wall obstacles
Initial Conditions: Robot in home position, cubes placed at random positions on one side of the table

The single obstacle setup is simple. Referencing *Fig. 4*, the table is divided into two sides by a wall in the middle. The arm must plan a trajectory around the wall to get to the other side.

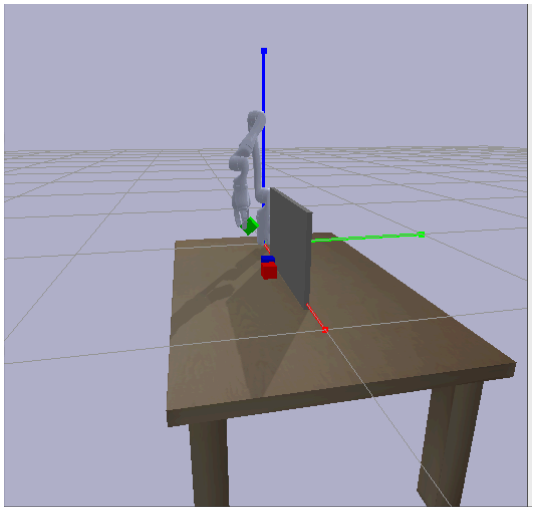


Figure 4: *Single dividing wall between the left and right sides of the table. A robotic arm is shown planning a path around the wall, having picked up a block.*

4.2: Table 1 - Single Obstacle Success Rate and Planning Time

Metric	Value
Success Rate (%)	84
Avg. Planning Time	6.245 seconds

The multiple obstacle setup involves a combination of different obstacles. As it goes, the objective is the same: to get the arm from one side to the other. However, instead of a singular wall in the middle dividing the two, there are multiple walls blocking above and the sides of the wall, as shown in *Fig. 5*. Now, not only does the robot have to maneuver around the single wall, but it also has to plan a path to consider the constraints on the sides and on top of the original wall, requiring a significantly more precisely planned trajectory.

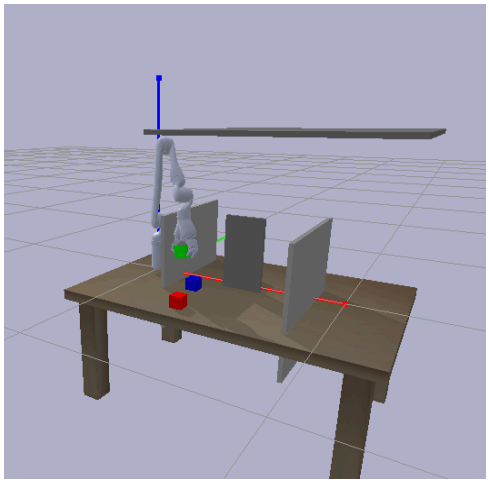


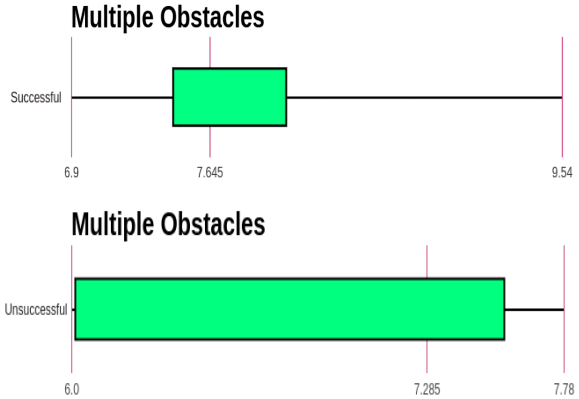
Figure 5: *Multiple obstacles, including a roof and multiple walls, blocking the path between the robot arm and the other side.*

4.3: Table 2: Multiple Obstacles Success Rate and Planning Time

Metric	Value
Success Rate (%)	60
Avg. Planning Time	7.5448 seconds

Most of the failures that occurred with single obstacles were not significantly notable as they were mostly glitches in the PyBullet environment. It was interesting to note, however, the failures for multiple obstacles. In the case of multiple obstacles, we observed that most of the time, the arm would take the longer route around the walls, rather than the shorter, but more precise, route between the walls. Because the route was longer, the block would have a higher chance of slipping out of the fingers than when the robot chose the shorter path, leading to

most of the failures. This correlates with the respective planning times for successful and unsuccessful attempts. Referencing the box plots below, successful attempts, on average, required higher planning time due to the fact that a more precise plan through the walls were planned. The failure times for unsuccessful attempts had a range of planning times. Lower planning time failures occurred from the block slipping out of the fingers from taking the longer route which required less precise planning. Higher planning time failures occurred if the block would accidentally hit the wall due to the space being tight. For future research, we can tune the motion planning algorithm to bias shorter, but more precise paths.



V. Conclusions

This research presents a comprehensive approach to integrating motion planning and object recognition for a generalized task planning robot, focusing on stacking cubes of different colors in the presence of obstacles. The solution leverages the PyBullet simulation environment and MoveIt for motion planning, employing the RRT-Connect algorithm for its efficiency in complex environments. In conclusion, this research marks a significant step towards developing generalized task planning robots capable of handling complex tasks in dynamic and obstacle-rich environments. The integration of advanced motion planning and object recognition techniques offers a scalable and efficient solution, paving the way for more autonomous and versatile robotic systems in home and industrial settings.

References

[1] M. Merlin, “Generalizable Task Planning,” Apr. 2024.
[2] C. Wang, D. Xu, and L. Fei-Fei, “Generalizable Task Planning through Representation Pretraining,” May 2022.
[3] S. Mukherjee, C. Paxton, A. Mousavian, A. Fishman, M. Likhachev, and D. Fox, “Reactive Long Horizon Task Execution via Visual Skill and Precondition Models,” Jul. 2021.