Pick-and-Place Path Planning for Robotic Manipulation

Kousar Kousar, Aqsa Shabbir, Kerem Bayramoğlu
Department of Computer Engineering
Bilkent University, Ankara

***Abstract*—This paper presents a novel framework for robotic manipulation within a table-cleaning environment, integrating high-level task planning and advanced motion optimization techniques. Leveraging the Unified Planning library, the framework employs the Planning Domain Definition Language (PDDL) to define hierarchical task dependencies and guides the robot through complex operations, including pick-and-place and wiping. Motion planning is enhanced through the use of state-of-the-art algorithms such as Probability-Smoothing Bi-RRT (PSBi-RRT), Bi-RRT*, RRT*, and Bi-RRT, ensuring efficient and collision-free paths. The framework is implemented in a simulated environment using the RAI platform and the Franka Panda robotic arm, showcasing its ability to handle interdependent tasks effectively.**

I. Introduction

Robotic manipulation tasks, such as table cleaning, is vital for autonomous systems, requiring a blend of symbolic reasoning and motion planning. While motion planning algorithms like RRT and RRT* efficiently generate collision-free paths, they lack task-dependency reasoning. Conversely, Task and Motion Planning (TAMP) frameworks excel at symbolic reasoning but struggle with path optimization and efficiency.

This work presents a hybrid framework that combines the strengths of TAMP and motion planning algorithms to address the challenges of robotic manipulation in a table-cleaning environment. The Unified Planning (UP) library is used to implement PDDL, while PSBi-RRT, Bi-RRT*, RRT* and Bi-RRT provide pathfinding solutions for executing these plans. The framework is designed to handle complex manipulation tasks, such as picking and placing large objects, and wiping table.

As part of the development process, we have successfully implemented the PDDL using UP library. This symbolic layer is integrated in RAI environment, where an existing version of RRT (Bi-RRT in rai) is taken as a basis and it's variants are implemented to enhance pathfinding capabilities. The implementation is done in C++ and using Pybind11, is integrated into rai. The framework is being validated in a simulated table-cleaning scenario using a Franka Panda robotic arm.

II. Related Work

Robotic manipulation and motion planning have been extensively studied, with numerous approaches addressing task and motion challenges. TAMP frameworks combine high-level symbolic reasoning with low-level motion planning to address complex, long-horizon tasks. Notable TAMP frameworks include [1], which utilizes a planner-independent interface to connect task planning with motion planning, and PDDLStream [2], which integrates symbolic planners and black-box samplers via optimistic adaptive planning.

Motion planning techniques such as RRT [3] and its variants, including RRT* [4] and Bidirectional RRT [5], are widely used for generating collision-free paths. However, these algorithms often yield suboptimal paths or require excessive computation time in dense environments. Recent advancements, such as PSBi-RRT [6], address these limitations through dual-target bias sampling, node correction, and θ-cut mechanisms to improve path quality and convergence rates.

This work builds upon these approaches, integrating TAMP with Bi-RRT, RRT*, Bi-RRT* and PSBi-RRT to create a unified framework for robotic manipulation in a table-cleaning environment.

III. TAMP Framework

This work builds on the structured symbolic reasoning of the PDDL and motion planners which are Bi-RRT, RRT*, Bi-RRT*, and PSBi-RRT. Our framework is structured into environment setup, task planner, motion planning, and execution.

A. Environment Setup

The table-cleaning environment shown in Figure 1, is designed to simulate robotic manipulation tasks, incorporating a variety of objects, tools, and constraints. The workspace includes a table as the central area for manipulation, bins for sorting objects, and a robotic arm with interchangeable tools. The objects in the environment such as *tray1*, *tray2*, and a *mug*, require grasping and placing into a designated bin. These objects are handled with the *gripper*. Secondly, the table is wiped using the *sponge* as a tool. Tool usage is constrained; for instance, only the sponge can wipe the table, while only the gripper can grasp objects. Additionally, certain hierarchical constraints are imposed on the task. For example, wiping actions are disallowed until all the objects have been removed from the table and placed in the bin.

B. Task Planner Implementation

Our task planner utilizes the UP framework to execute a hierarchical Task Planning approach. The planner processes the problem defined in the PDDL and generates a sequential

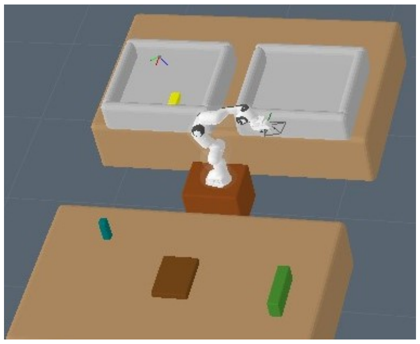
Fig. 1: Environment

symbolic plan that respects task dependencies and satisfies predefined goals. This structured plan comprises actions such as picking, placing, and wiping, which are orchestrated to adhere to logical constraints and dependencies.

1) Domain and Problem Definitions: The task planner operates on a domain and problem definition, each contributing essential elements to the planning process.

1) **Domain:** The domain defines the robot's possible actions, their preconditions, and effects. Three primary actions are implemented:
   - **Pick:** This action allows the robot to grasp an object using a specified tool. Preconditions for the action include the object being present on the table, being ready for picking, and the tool being usable for that object. The effects include removing the object from the table and marking it as held by the robot.
   - **Place:** This action permits the robot to place a held object into a target location, such as a bin. Preconditions require the robot to be holding the object. Effects include placing the object in the bin and updating its status to indicate that it no longer requires processing.
   - **Wipe:** Wiping is used to clean a dirty table. Preconditions include the table being dirty and wiping being allowed. The effects mark the table as clean.
2) **Problem:** The problem file describes the initial state of the environment and the desired goals:
   - Initially, tray1, tray2, and a mug are on the table, which is dirty.
   - The goals specify that all objects must be moved to a bin and the table must be wiped clean.

2) Sequential Dependencies and Intermediate Actions: To handle task dependencies, intermediate actions are incorporated into the planner:

1) **Allow Next Pick (Tray2):** This action ensures tray2 becomes available for picking only after tray1 is successfully placed in the bin.
2) **Allow Next Pick (Mug):** This action ensures the mug becomes available for picking only after tray2 is placed in the bin.
3) **Allow Wiping:** This action permits wiping the table only after all objects have been placed in the bin.

These intermediate actions enforce a logical sequence of tasks, ensuring that actions like wiping occur only after all other prerequisites are satisfied.

3) Plan Execution: The task planner generates a sequential plan that systematically fulfills all goals while respecting dependencies between tasks. The execution begins with the robot picking up tray1 from the table and placing it in the designated bin, satisfying the prerequisite for the next task. Once tray1 is processed, the intermediate action $(allownextpicktray2)$ ensures that tray2 becomes available for picking. The robot then proceeds to pick tray2 from the table and place it in the bin, triggering the intermediate action $(allownextpickmug)$, which enables the mug to be processed. Subsequently, the robot picks the mug, places it in the bin, and invokes the intermediate action $(allowwiping)$. This action permits the table to be cleaned only after all objects are cleared. Finally, the robot uses the sponge to wipe the table, transitioning it from dirty to clean.

C. Motion Planning

Once the symbolic plan is generated, motion planning is used to translate these high-level actions into executable trajectories. This process involves generating collision-free paths for the robot's movements in continuous space. To address varying levels of complexity in the tasks, three motion planning algorithms are developed and compared to existing Bi-RRT(available in rai via ry.PathFinder()) method: RRT*, PSBi-RRT and Bi-RRT*.

1) RRT:* Rapidly-exploring Random Tree* (RRT*) is a sampling-based algorithm that improves upon RRT by optimizing path quality through tree rewiring to minimize the cost to the goal. Its iterative process includes sampling, tree extension, neighbor selection, and path optimization, making it suitable for scenarios requiring asymptotically optimal paths. The pseudocode for RRT* is shown in Algo 1.

Algorithm 1: RRT*

```
Input: Start state $q_{\text{start}}$, Goal state $q_{\text{goal}}$, Maximum iterations
       $max\_iter$, Step size $\Delta q$, Radius $r$
Output: Optimal path connecting $q_{\text{start}}$ to $q_{\text{goal}}$ (if found)
1  Initialize tree $T_{\text{start}} \leftarrow$ InitializeTree($q_{\text{start}}$);
2  for $i \leftarrow 1$ to $max\_iter$ do
3      $q_{\text{rand}} \leftarrow$ SampleRandomState();
4      $q_{\text{new\_node}} \leftarrow$ ExtendTree($T_{\text{start}}, q_{\text{rand}}, \Delta q$);
5      if $isColliding(q_{new\_node}) == False$ then
6          Neighbors $\leftarrow$ FindNeighbors($T_{\text{start}}, q_{\text{new\_node}}, r$);
7          $q_{\text{best}} \leftarrow$ SelectBestParent($Neighbors, q_{\text{new\_node}}$);
8          SetParent($q_{\text{new\_node}}, q_{\text{best}}$);
9          RewireNeighbors($q_{\text{new\_node}}, Neighbors$);
10     end
11 end
12 if $Distance(q_{new\_node}, q_{new\_goal}) < \Delta q$ then
13     return ReconstructPath($T_{\text{start}}, q_{\text{goal}}$);
14 end
15 return Failure;
```

2) Bi-RRT:* Bidirectional Rapidly-exploring Random Tree* (Bi-RRT*) extends RRT* by growing two trees, one from the start and one from the goal, to accelerate convergence. It refines path quality through rewiring and alternates tree extensions to connect them efficiently. The pseudocode is shown in Algo 2.

Algorithm 2: Bi-RRT*

```
Input: Start state $q_{\text{start}}$, Goal state $q_{\text{goal}}$, Maximum iterations
       $max\_iter$, Step size $\Delta q$, Radius $r$
Output: Optimal path connecting $q_{\text{start}}$ to $q_{\text{goal}}$ (if found)
1  Initialize trees $T_{\text{start}} \leftarrow$ InitializeTree($q_{\text{start}}$) and
   $T_{\text{goal}} \leftarrow$ InitializeTree($q_{\text{goal}}$);
2  for $i \leftarrow 1$ to $max\_iter$ do
3      $q_{\text{rand}} \leftarrow$ SampleRandomState();
4      $q_{\text{new\_node}} \leftarrow$ ExtendTree($T_{\text{start}}, q_{\text{rand}}, \Delta q$);
5      if isColliding($q_{new\_node}$) == False then
6          Neighbors $\leftarrow$ FindNeighbors($T_{\text{start}}, q_{\text{new\_node}}, r$);
7          $q_{\text{best}} \leftarrow$ SelectBestParent($Neighbors, q_{\text{new\_node}}$);
8          SetParent($q_{\text{new\_node}}, q_{\text{best}}$);
9          RewireNeighbors($q_{\text{new\_node}}, Neighbors$);
10     end
11     Swap $T_{\text{start}}$ and $T_{\text{goal}}$;
12 end
13 if Distance($q_{new\_node}, q_{new\_goal}$) $< \Delta q$ then
14     return ReconstructPath($T_{\text{start}}, T_{\text{goal}}$);
15 end
16 return Failure;
```

3) PSBi-RRT: The Probability-Smoothing Bidirectional Rapidly-exploring Random Tree (PSBi-RRT) enhances Bi-RRT with elliptical sampling and dual-target bias for efficient pathfinding in cluttered environments. It reduces exploration with restricted sampling, employs θ-cut optimization to smooth paths, and alternates bidirectional tree growth for faster convergence. The algorithm can be seen in the submitted folder.

D. Execution in the RAI Environment

The execution phase of the framework is represented in the RAI simulation environment, employing a Franka Panda robot equipped with a gripper to perform a sequence of task actions on objects situated on a table. The implementation integrates both discrete and continuous analysis, leveraging advanced motion planning algorithms. The goal of the motion planner is to identify a collision-free trajectory between an initial configuration q_{init} and a goal configuration q_{goal}. Each primitive operator is executed in the RAI environment by:

1) Mapping symbolic actions (e.g., pick-and-place or wiping) to corresponding motion primitives.
2) Using IK to compute joint configurations for grasping, placing, and manipulating objects.
3) Employing Bi-RRT, RRT*, Bi-RRT* and PSBI-RRT to validate and refine the motion paths.

IV. Results and Discussion

A. Task Plan Execution

The execution of PDDL-based symbolic task planning in our framework encountered certain challenges, particularly in scenarios where task dependencies and geometric constraints were not fully accounted for during plan generation. Our final state is depicted in Figure 2. While the PDDL model successfully produced sequential plans that satisfied symbolic preconditions and effects, the integration of these plans with low-level motion planning revealed issues with misclassifications and task misalignment. These challenges underscored the complexities of merging high-level symbolic reasoning with the practical requirements of motion feasibility.

One major issue identified during execution was the planner's inability to dynamically consider geometric constraints. For example, symbolic plans assumed that tasks like wiping the table could proceed immediately after objects were placed in the bin. However, in practice, residual obstructions or improper object placement on the table surface led to failures during the motion execution phase. Additionally, dependencies between tasks, such as clearing the table before allowing wiping, were sometimes violated due to insufficient reasoning about the spatial configuration of objects.

Another contributing factor was the lack of coordination between symbolic and motion planning. While the PDDL planner generated logically valid sequences, unforeseen constraints during execution, such as collision risks or unreachable grasp poses, disrupted the sequential flow. For instance, placing objects in the bin with incorrect orientations caused subsequent tasks to become infeasible, highlighting the need for a tighter coupling between symbolic and geometric reasoning.

To address these issues, several corrective measures were implemented. Geometric constraints were explicitly incorporated into the symbolic planner by adding preconditions and effects that ensured task feasibility. For example, actions like wiping the table were augmented with conditions verifying that the table surface was unobstructed. These refinements significantly improved the framework's performance, ensuring that all task dependencies and geometric constraints were respected. Our final plan that was executed is shown below:

1) *pick*(tray1, table1, l_{gripper})
2) *place*(tray1, bin1, l_{gripper})
3) *allow next pick(tray2)*
4) *pick*(tray2, table1, l_{gripper})
5) *place*(tray2, bin1, l_{gripper})
6) *allow next pick(mug)*
7) *pick*(mug, table1, l_{gripper})
8) *place*(mug, bin1, l_{gripper})
9) *allow wipping*
10) *wipe*(table1, sponge)

B. Motion Execution

The analysis shown in the Table I highlights distinct performance patterns among the methods as the distance between start pose and goal pose changes. When a solution is found, the RRT* consistently produces the shortest average path length, indicating its efficiency in generating optimal routes. Conversely, PSBi-RRT often results in the longest average path length, suggesting it prioritizes finding paths over optimizing them. However, PSBi-RRT excels in finding successful paths, achieving the highest success ratio across most scenarios, even as distances increase. In contrast, RRT* exhibits the lowest success ratio, struggling particularly as the distance grows more challenging.

Distance	**Bi-RRT**		**RRT***		**Bi-RRT***		**PSBI**	
	Avg Path	**Succ Paths**	**Avg Path**	**Succ Paths**	**Avg Path**	**Succ Paths**	**Avg Path**	**Succ Paths**
3.13	3.588	100	3.238	100	3.565	100	3.661	100
3.16	3.607	100	3.268	99	3.599	100	3.706	100
3.20	3.635	80	3.386	1	3.600	78	3.835	100
3.23	3.627	7	0.000	0	3.708	4	4.002	99
3.27	0.000	0	0.000	0	0.000	0	4.339	95
3.31	0.000	0	0.000	0	0.000	0	4.406	83
3.35	0.000	0	0.000	0	0.000	0	5.485	10

TABLE I: Comparison of Results for Different Methods

An alternative analysis could examine the trade-offs between success ratio and computational effort, as indicated by average evaluations. For example, the Star and Single Star methods consistently require significantly higher evaluations compared to other methods, regardless of the scenario. This is because these methods take additional steps to optimize the path length, ensuring that the algorithm performs refinement processes. As a result, they run until the selected maximum number of iterations, which is 1000 in this example. While this approach helps achieve shorter paths when solutions are found, it comes at the cost of computational efficiency and success ratios. Notably, the success ratios of both Bi-RRT* and RRT* methods degrade significantly as the distance increases, with RRT* failing completely beyond a certain threshold.

These findings suggest a clear trade-off between path optimization and robustness. While PSBi-RRT sacrifices path length optimization for higher success ratios, RRT* and the Star methods prioritize path quality but are more computationally intensive and less reliable in challenging scenarios. Balancing these factors could involve tuning the maximum iterations or introducing adaptive strategies to manage computational resources and enhance reliability.

In [6], where the PSBi-RRT algorithm is originally defined, it is demonstrated that PSBi-RRT outperforms Bi-RRT in terms of both success ratio and efficiency. However, this is not reflected in our results, where PSBi-RRT does not consistently outperform other methods like RRT* in certain aspects. A possible reason for this discrepancy is that the initial PSBi-RRT design was specifically tailored for mobile robots, with robot dynamics based on two-wheel robot models. This specialization may not translate effectively to the context of our work. Additionally, the authors of [1] mention tuning the algorithm's parameters based on experimental results for their specific environments. In contrast, our implementation directly adopted the parameter values from their work without further optimization. This suggests that performance may be further improved by fine-tuning the parameters to better align with the specific requirements of our environment and tasks.

V. Conclusion & Future Work

The project successfully integrates TAMP with advanced motion planning algorithms, including Bi-RRT, PSBi-RRT, Bi-RRT*, and RRT*, to address the complexities of robotic manipulation in a table-cleaning environment. The framework

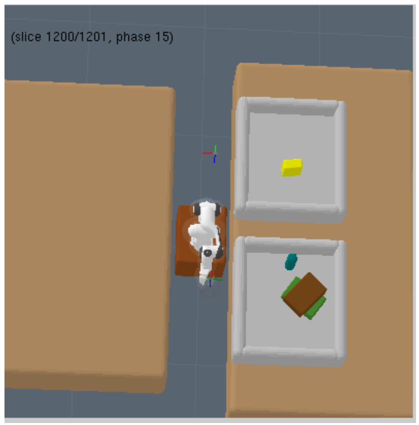

Fig. 2: Final State of Our Environment

effectively combines symbolic reasoning and motion optimization to handle interdependent tasks like pick-and-place operations and wiping while adhering to hierarchical constraints. Future work will focus on extending the framework to dynamic environments where objects and obstacles may change in real-time. Enhancements include real-time updates to symbolic plans and motion trajectories, vision-based perception for object recognition in unstructured settings, and reinforcement learning for adaptive motion planning.

References

[1] S. Srivastava, E. Fang, and Riano, "Combined task and motion planning through an extensible planner-independent interface layer,"

[2] C. R. Garrett and Lozano-Pérez, "Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning,"

[3] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning,"

[4] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning,"

[5] J. Wang, B. Li, and M. Q.-H. Meng, "Kinematic constrained bi-directional rrt with efficient branch pruning for robot path planning,"

[6] G. Ma, Y. Duan, M. Li, Z. Xie, and J. Zhu, "A probability smoothing bi-rrt path planning algorithm for indoor robot,"