Motion Planning for Robotic Arm Handover Tasks Using Sampling-Based Methods

Seongil Heo[1], Jesse Jenkins[1], Cameron Monson[1], and Corbin Gurnee[1]

***Abstract*—This paper presents a novel approach to coordinated object handover between two robotic arms using the optimal Rapidly-exploring Random Tree (RRT*) algorithm. The primary challenge addressed is the generation of collision-free, optimal trajectories for both robots in a shared workspace while dynamically coordinating their motions to ensure a smooth and stable transfer of an object. Our method employs a dual-arm planning framework in which each arm independently computes its motion plan using RRT*, while incorporating real-time constraints to maintain synchronization during the handover process. Simulation and experimental results on a dual-arm robotic platform demonstrate that our approach achieves efficient and reliable handovers with minimized path cost and reduced planning time, highlighting the potential of RRT*-based methods.**

***Index Terms*—Robot arm, handover, motion planning, RRT*, collision free**

I. Introduction

Recent studies show that object handover is not only essential in human–robot collaboration but also plays a key role in multi-robot coordination. In particular, object transfer between two robot arms is gaining attention, requiring precise grasp planning and trajectory synchronization [1].

In this project, we aim to develop a system in which two robotic arms collaborate to pass an object from one to another efficiently and safely. The objective is to focus on two main aspects: grasp strategy and motion planning. Grasp strategy involves determining suitable graspable parts or poses for secure object acquisition, while motion planning aims to generate smooth and safe trajectories for completing the handover process to the other robotic arm.

II. Related Work

The domain of robot-to-robot (R2R) handover has garnered significant attention, focusing on enhancing coordination, perception, and control mechanisms to facilitate seamless object transfers between autonomous agents.

Sileo et al. [2] introduce a vision-based R2R handover framework that operates without explicit communication. Both the giver and receiver robots are equipped with eye-in-hand depth cameras, employing deep learning techniques for object detection. The physical exchange leverages impedance control and contact force estimation, enabling the receiver to perceive the object's presence and the giver to detect the completion of the handover. This approach emphasizes adaptability in unstructured environments and robustness against pose estimation error.

Costanzo et al. [3] present a comprehensive control strategy applicable to both human-robot and robot-robot handovers. Their method integrates visual servoing for dynamic object tracking and force-based grip modulation to prevent slippage during the transfer. They also use haptic cues, allowing the receiver to signal readiness and the giver to adjust grip force accordingly. The system employs finite-state machines to manage the handover process, ensuring responsiveness to uncertainties such as object mass variations and grasp configurations.

Previous work in robot-to-robot handoff has addressed challenges such as perception, coordination, and real-time control. Our project does not aim to introduce new solutions in these areas, but rather implements a simplified version of the handoff problem using basic motion planning techniques. By studying these foundational concepts in a controlled environment, we aim to better understand the challenges involved in more advanced autonomous handoff systems.

III. Methods

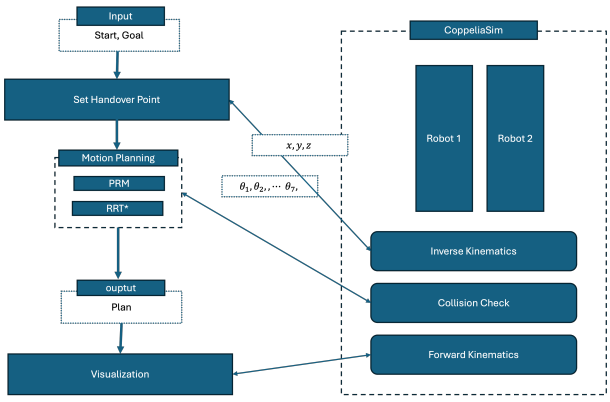


Fig. 1: System Workflow for Motion Planning and Visualization

Trajectory planning algorithms are employed to generate smooth and collision-free motion for both arms. The following key methods are considered.

- **Trajectory Planning:** Joint trajectories will be generated using sampling-based motion planning algorithms such as PRM and RRT*.
- **Grasp Strategy/Handover Coordination:** For simplicity, we assume that when the giver and receiver grippers

[1]Seongil Heo, Jesse Jenkins, Cameron Monson, and Corbin Gurnee are with the Kahlert School of Computing, University of Utah, Salt Lake City, UT 84112 USA. {seongil.heo, u1355879, u1356158, u1261969}@utah.edu

are sufficiently close, the object is handed over. In reality, this process is much more complex, but we simplify to focus attention on motion planning algorithms.

- **Inverse Kinematics:** To determine start and goal locations to use in motion planning algorithms, we used optimization methods to compute the inverse kinematics for each arm. We created a cost function to return the distance between the arm gripper and target location, and minimized this function using SciPy.
- **RRT* Implementation:** Our RRT* Implementation extends the basic RRT by, at each sample, first steering a small collision-checked step towards a random point, then gathering all existing nodes within a defined radius and selecting the new nodes parent to be the node that minimizes the cumulative cost of the length, which ensures each insertion is locally optimal. After adding this new node, we rewire the surrounding neighbors by testing whether connecting them through this new node will yield a lower cost, and if it does so, we reattach their connection through this new node. Once a new node falls within a small threshold of the goal, we attach a goal node and extract the least cost back to the start.
- **PRM Implementation:** Our PRM Implementation is pulled from HW 2. It works by effectively creating a web of connected components through randomly selected points. Once this web has been created, it then generates a given path by placing our start and end points within this web and using these limited and finite paths, finding the shortest path within our previously created web form the start to the end point.

IV. Experiments and Results

One of the main decisions we made early on was how we wanted to represent the two arms of the robot we were going to be working with. We decided early to effectively approach the problem as though we were solving two separate motion planning problems, with two different robots, in series. This was due to the difficulty in collision detection with two robots occupying two different spaces. However we did experiment with representing this problem as a single robot with 6 joint angles for our 2D environment (Figure 2) and solved this as a larger single problem with an intermediate objective of reaching a point where two specific points were instead in collision.

A second experiment we briefly worked with was how we were going to choose the specific handover point both our robots were going to be working toward reaching. We spoke about and created la few methods to attempt to select the best point for our specific problem situation we had created as well as briefly attempted to make our own method that would run during our motion planning algorithm that would attempt to select a handover point for our robots, however we eventually decided to settle on just manually selecting a handover point to be used. This was helpful as it allowed us to always have a very definite goal position that we would be attempting to reach, however it also meant that there is another step that is necessary before our algorithms and setup can be done that could possibly be eliminated should we be able to provide extra time to this project.

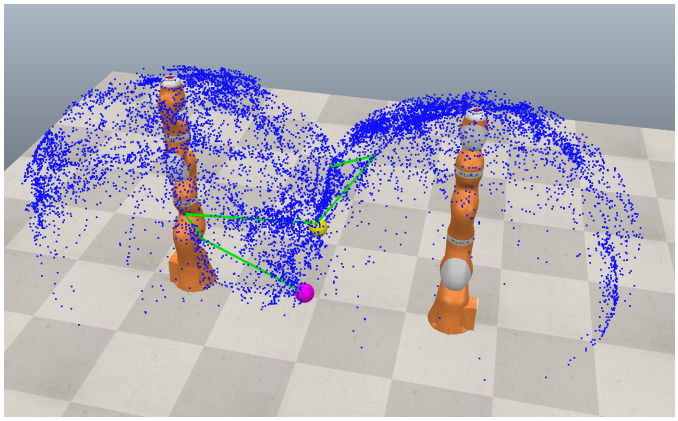


Fig. 2: 7DoF RRT*

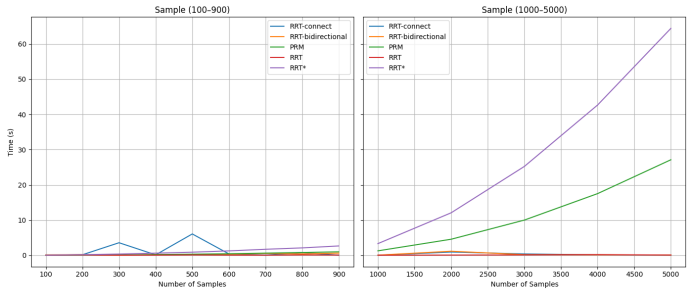
Another experiment we worked with had to do with the way we were going to represent the problem we were going to solve. We had initially wanted to work in a more robust simulator to attempt to make the problem we were going to solve as detailed as possible. The feasibility and difficulty of this quickly made itself known, and we largely scaled back our implementations. We were able to then create and solve a more simple 2D environment with the 3 joint angles and two robots. Once this had been completed we decided that we would like to implement a 3D environment of some sort. This was found to be very difficult, and many solutions were tried and ultimately abandoned before returning to a more simplistic simulator approach that we eventually worked with and were again able to solve as shown in this report.

A final experiment we worked with had to do with our early attempts with the gripper that was at the end of the arms of our two robots in the 3D environment. We had originally intended to create a method in which these grippers were moved to open prior to reaching our start, and to then properly physically transfer the ball between the two arms. However, we eventually that this would be beyond the scope of our motion planning project, and instead would be a good task to attempt to work on in the future if we were able to have extra time.

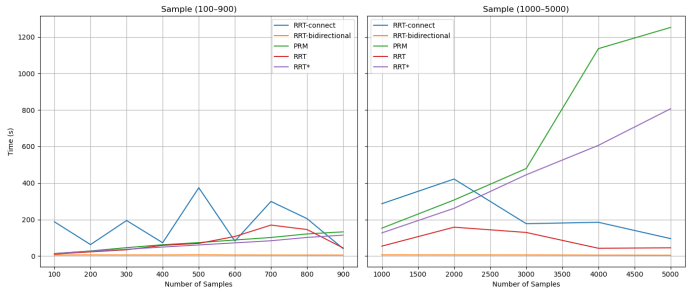
A. Time Comparison in 2D vs. 3D Environments

We evaluated the performance of different sampling-based motion planners across both 2D and 3D environments by measuring their execution times as a function of the number of samples. he results are visualized in Figure 3a and Figure 3b, which correspond to the 2D and 3D environments, respectively. As shown in the charts, all planners demonstrate an increasing trend in runtime with a growing number of samples. For most planners, the growth is approximately linear. This trend is consistent in both 2D and 3D scenarios.

A notable distinction arises when comparing the performance of PRM and RRT* across dimensions. PRM constructs a global roadmap before performing a graph search, whereas



(a) 2D RRT*



(b) 3D RRT*

RRT* incrementally expands the tree and continuously optimizes the path by evaluating and rewiring neighboring nodes. In the smaller 2D environment, PRM outperforms RRT* due to its efficient roadmap-based planning. However, in the more complex 3D environment, PRM suffers from significant computational overhead, resulting in longer runtimes compared to RRT*.

These results highlight a key difference between graph-search-based and optimization-based planning methods. While graph construction methods like PRM are efficient in lower-dimensional spaces, they become computationally expensive when scaled. In contrast, incremental methods like RRT* maintain relatively stable scalability despite their iterative nature. .

B. Component Timing Analysis

In order to gain deeper insights into the computational bottlenecks of sampling-based motion planning, we conducted a component-level timing analysis. This experiment was motivated by discussions in class, where it was emphasized that two operations—collision checking and nearest-neighbor search—typically account for the majority of a planner's runtime.

To validate this observation, we instrumented our planner to measure the time spent in three key components: finding the nearest node to a sampled point, performing collision checking between nodes and edges, and searching neighboring nodes for potential rewiring. These measurements were conducted under varying sample sizes to observe how the cost of each component scales with problem complexity.

Our initial hypothesis was that the neighbor-search and rewiring steps would dominate total computation time. This assumption was based on the fact that these steps involve numerous distance computations, especially as the number of nodes increases.

As shown in Figure 4, the curve representing collision checking nearly overlaps with the total planning time, suggesting that collision detection alone accounts for the vast majority of computation.

We attribute this observation largely to the overhead introduced by interfacing with the simulator. Collision checks required invoking external function calls, which are significantly more expensive than simple in-memory distance computations. This simulation-related latency may have amplified the cost of collision checking beyond what would be observed in a standalone implementation.

Despite this simulator-induced artifact, the findings reinforce a critical insight: collision checking remains a dominant factor in planning efficiency, particularly in real-world or simulator-integrated systems. These results highlight the importance of developing highly optimized collision-checking routines, especially when deploying planners in robotics applications where real-time performance is essential.

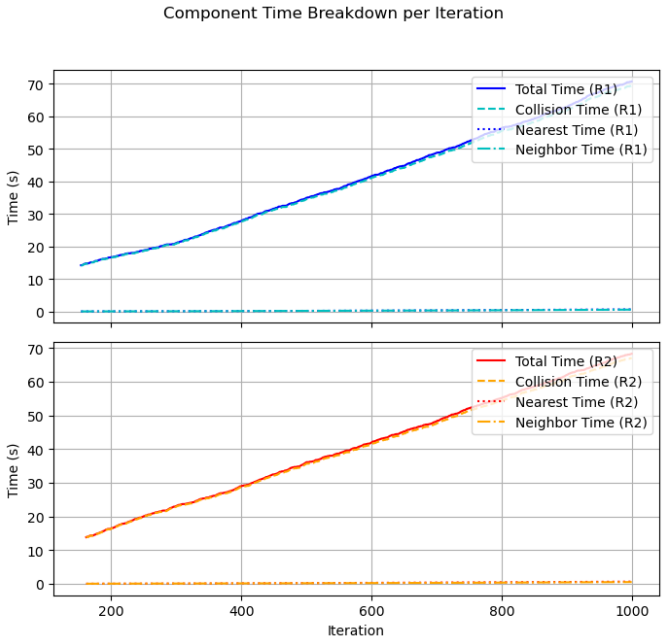


Fig. 4: Result-Time Comparison

C. Total Time vs. Path Length

Figure 5 illustrates the results for both robots. The dashed lines represent the path lengths, which exhibit a steep decline during the initial 200 iterations, indicating rapid early improvement in solution quality. After this phase, the path length stays almost the same, which means adding more samples does not improve the result much. On the other hand, the solid lines show the total planning time, which keeps increasing at a steady rate throughout all 1,000 iterations.

These observations highlight a critical aspect of sampling-based optimization: while the early stages contribute significantly to path quality, continued iterations incur linear time costs without proportional benefit. This suggests that, in practice, RRT* should not be allowed to run until a fixed number of samples is exhausted. Instead, implementing a well-defined stopping criterion—such as a convergence threshold on path improvement—can yield substantial efficiency gains without sacrificing solution quality.

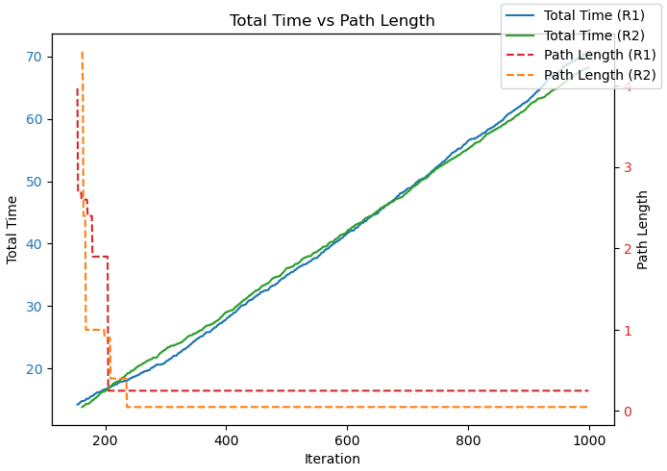


Fig. 5: Time and Path Length Comparison

V. Analysis

- **RRT*:** The repeated best-parent selection of RRT* not only preserves the probabilistic completeness present within RRT, but also drives the solution cost downward over time which gives us an anytime planner that converges towards a true optimal path. In the context of our problem, which would require a more smooth and energy efficient path, RRT* excels by producing progressively shorter and safer motions under tight computation budgets. This results in the collision time exceeding the Nearest Neighbor time by a significant amount as shown in Figure 4
- **PRM:** PRM in specific is beneficial in creating our paths within our handoff method because it is capable of being flexible in its start and end positions. With our handoff in specific, we are attempting to have a variable start and end position with our different handoff locations. This allows the memory from PRM to be useful and quickly generate or solve multiple different problems from similar initial and eventual goal positions.

VI. Discussion

In this project we learned the process of starting with an idea and transforming it into a project, from planning to implementation. We adapted to our knowledge and abilities in order to adjust our project for unforeseen issues.

We learned how to implement RRT* and how it compares to similar algorithms such as RRT-Connect, Bi-directional RRT, and PRM. We also discovered that representing our robot as two different arms with separate states allows us to reduce the number of collision checks in the planning process. Finally, we learned how to design using modular programming, which allowed us to reuse code for different algorithms and applications.

If we were to extend this work, we would like to implement a learnable handover point. This would allow the robot to hand over any object without requiring a human-defined setup. Additionally, we would like to implement an optimization-based algorithm that could significantly reduce the computational cost of sample-based methods.

Furthermore, several directions were explored conceptually during the project but could not be fully implemented due to time limitations. A key consideration was the comparison between different state representations—using either separate trees for each robot or a combined tree for both—which remains incomplete. Trajectory smoothness posed an ongoing challenge as a result of the sampling-based nature of RRT*, highlighting the need for future integration of post-processing techniques. Although gripper control was not included in the current implementation, it remains a critical element for real-world object handover and is planned for future development.

References

[1] H. Duan, Y. Yang, D. Li, and P. Wang, "Human–robot object handover: Recent progress and future direction," *Biomimetic Intelligence and Robotics*, vol. 4, no. 1, p. 100145, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2667379724000032

[2] M. Sileo, M. Nigro, D. D. Bloisi, and F. Pierri, "Vision based robot-to-robot object handover," in *2021 20th International Conference on Advanced Robotics (ICAR)*, 2021, pp. 664–669.

[3] M. Costanzo, G. De Maria, and C. Natale, "Handover control for human-robot and robot-robot collaboration," *Frontiers in Robotics and AI*, vol. Volume 8 - 2021, 2021. [Online]. Available: https://www.frontiersin.org/journals/robotics-and-ai/articles/10.3389/frobt.2021.672995