# Chapter 4: Methodology

## 4.1 Introduction

The primary objective of this chapter is to detail the implementation of several motion planning algorithms—RRT, RRT\*, JPlusRRT, IK-RRT, and BIK-RRT—using Python, and to evaluate their performance in both PyBullet and the Jogramop framework. These algorithms were chosen to explore their applicability to integrated grasp and motion planning tasks, particularly in robotic environments with dynamic constraints and complex object configurations.

The experimental process is divided into two phases. The first phase focuses on implementing and testing the algorithms in the PyBullet environment, where their core performance metrics, such as planning time and success rate, were analyzed in simpler scenarios. In the second phase, the RRT\* algorithm was tested within the Jogramop framework, which offers standardized benchmark scenarios designed to challenge the algorithms with more complex and confined environments.

This chapter will cover the research design, the rationale for the algorithms and frameworks used, the experimental setup, the evaluation metrics, and the presentation of the results. By conducting these experiments, we aim to demonstrate how the algorithms perform in practical robotic applications and compare their effectiveness in solving integrated grasp and motion planning problems.

## 4.2 Problem Formulation

The problem addressed in this research involves the integrated grasp and motion planning for a robotic manipulator, with a focus on comparing algorithmic performance in benchmark scenarios.

In this context, we consider a set of potential grasps, where each grasp consists of a grasp pose representing the 6-DoF position and orientation of the gripper relative to the object.

The motion planning component involves searching the robot's configuration space , where each configuration represents a particular set of joint angles for the manipulator. The collision-free subset of the configuration space is denoted as . The objective for the motion planner is to find a path from an initial configuration to a goal configuration , ensuring that the chosen grasp ​ is reached while minimizing path length, avoiding obstacles, and adhering to the robot's kinematic constraints (Rudorfer et al., 2024).

Figure 1 Various scenarios from the Jogramop framework showcasing a robot hand attempting to grasp different objects.

To compare algorithms, we set up benchmark scenarios involving a variety of objects, grasps, and environmental configurations. The goal is to determine how well each algorithm performs in terms of path planning efficiency, and adaptability to dynamic changes in the environment. Performance metrics such as planning time, and success rate will be used to evaluate the effectiveness of each algorithm in real-time simulations.

## 4.3 Research Design

**Research Methodology**

The research follows a **quantitative experimental methodology** aimed to implement multiple motion-planning algorithms in a controlled simulation environment. The experiments were conducted to compare the algorithms based on key metrics such as planning time, success rate, and the exploration of the configuration space. The study is divided into two distinct phases, each designed to test the algorithms under different environmental conditions and complexity levels.

**Phases of Research**

1. **Phase 1**:  
   The first phase involved the implementation and testing of **RRT, RRT\***, **JPlusRRT, IK-RRT, and BIK-RRT** in the **PyBullet environment**. This phase was designed to assess the core performance of each algorithm in relatively simpler scenarios, allowing for a direct comparison of their effectiveness in solving motion planning problems. The focus was on how each algorithm explores the configuration space, generates feasible paths, and handles dynamic constraints.
2. **Phase 2**:  
   In the second phase, the **RRT\*** algorithm was implemented in the **Jogramop framework**, which features 20 standardized benchmark scenarios. These scenarios are more complex and confined than those in Phase 1, providing a more structured environment to evaluate RRT\*'s performance under more challenging conditions. The focus in this phase was on testing the algorithm's robustness and optimality in navigating obstacle-rich environments.

**Environment Setup**

* **PyBullet Environment**:  
  The PyBullet environment provided a simpler testing ground for initial comparisons between the algorithms. A 6-DOF Franka Panda robotic arm was used, interacting with basic objects and obstacle configurations. This environment allowed for testing in controlled, repeatable conditions, focusing on key performance metrics such as path efficiency and collision avoidance. The flexibility of PyBullet facilitated the testing of various object sizes and obstacle densities, ensuring comprehensive performance evaluation in the first phase.
* **Jogramop Framework**:  
  The Jogramop framework (Rudorfer et al., 2024) was utilized in the second phase to provide a more structured and standardized set of benchmark scenarios. These 20 predefined scenarios simulate real-world challenges, including confined spaces and complex obstacle distributions, making it an ideal environment for testing the more advanced features of RRT\*, such as path optimality and rewiring. The Jogramop framework allows for repeatable testing under uniform conditions, enabling more precise performance comparisons.

**Control Measures**

To ensure a fair comparison between algorithms, several control measures were implemented:

* **Consistent Parameters**: Key parameters such as step size, goal bias, and the number of iterations were kept consistent across all algorithms to eliminate bias in the comparison process. This ensured that any performance differences were due to the inherent capabilities of the algorithms rather than external factors.
* **Repetition of Trials**: Each experiment was repeated 100 times for each scenario, allowing for statistically significant results. Repeated trials helped minimize the effects of outliers and variability in performance.
* **Controlled Environment**: Both the PyBullet and Jogramop environments were designed to minimize external variability. Environmental factors such as object placement, obstacle density, and configuration space boundaries were kept constant throughout the experiments to ensure that all algorithms were tested under identical conditions.

## 4.4 Algorithms and Frameworks

The selection of algorithms and simulation frameworks in this research was guided by the need to explore various aspects of integrated grasp and motion planning, from basic pathfinding to handling complex constraints. Each algorithm and framework was chosen to fulfill specific roles in the overall implementation and testing process.

* **RRT (Rapidly Exploring Random Tree)**:  
  RRT was selected as a **baseline algorithm** for motion planning. As a widely used sampling-based method, it incrementally builds a tree that explores the configuration space quickly. RRT is especially useful for understanding how basic motion planning operates in high-dimensional spaces, making it an ideal starting point for implementing more advanced algorithms.
* **RRT\* (RRT-Star)**:  
  RRT\* was chosen due to its **path optimization capabilities**. Unlike RRT, RRT\* guarantees asymptotic optimality by refining paths as it grows the tree. Its ability to minimize path cost and produce more efficient trajectories makes it essential for tasks where grasping and motion planning must be optimized simultaneously. This algorithm is particularly well-suited for scenarios involving confined spaces or complex object configurations.
* **JPlusRRT, IK-RRT, and BIK-RRT**:  
  These algorithms were selected to address the **more complex aspects of grasp and motion planning**.
  + **JPlusRRT** integrates the Jacobian pseudo-inverse, enabling the robot to handle kinematic constraints more efficiently, which is critical in grasp planning.
  + **IK-RRT** incorporates **inverse kinematics** to ensure that the robot can reach desired grasp configurations while adhering to the robot's joint limits. This makes it suitable for motion planning tasks where the robot’s joint configurations play a key role.
  + **BIK-RRT** further extends the capabilities of IK-RRT by introducing **bidirectional search**, allowing the algorithm to explore the configuration space from both the start and goal configurations. This method enhances the algorithm’s ability to find feasible paths in complex environments, particularly where direct paths are difficult to find.

**Frameworks**

The choice of simulation frameworks—**PyBullet** and **Jogramop**—was motivated by their ability to support the experimental objectives at different stages of the project.

* **PyBullet**:  
  PyBullet was chosen for its **flexibility and ease of use** in implementing and testing motion planning algorithms. It supports real-time physics simulations and provides a wide range of tools for testing robotic systems. For this research, PyBullet was used in the initial phase of testing to implement and verify the basic functionality of the algorithms in simpler scenarios. It allowed for rapid prototyping, debugging, and visualization of the algorithms' behavior in a controlled environment.
* **Jogramop Framework**:  
  The **Jogramop framework** was selected for the second phase of testing due to its **structured benchmark scenarios**. Jogramop offers 20 predefined scenarios, designed specifically to challenge motion-planning algorithms in environments with more complex obstacles and confined spaces. This framework allowed for a more rigorous exploration of the RRT\* algorithm in standardized conditions, making it ideal for testing more advanced capabilities like path optimization and obstacle avoidance. The uniformity of these scenarios ensured consistent testing and helped in comparing the algorithm’s performance across different conditions.

**Algorithm-Specific Justifications**

* **RRT** was selected for its ability to quickly explore large configuration spaces, providing a good baseline for understanding basic motion planning behavior. Its simplicity makes it a valuable starting point for testing and implementing more complex algorithms.
* **RRT\*** was chosen to extend the exploration process by incorporating path optimization. Its ability to find optimal paths, particularly in cluttered environments, makes it an essential component for tasks where minimizing the trajectory’s cost is important.
* **JPlusRRT, IK-RRT, and BIK-RRT** were selected to investigate how grasp and motion planning can be effectively combined. These algorithms address specific challenges in robotic manipulation, such as managing kinematic constraints and finding feasible grasps. Their inclusion in the research allows for a more detailed analysis of how motion planning algorithms can handle real-world complexities in grasp planning.

In summary, the combination of these algorithms and frameworks allowed for a comprehensive exploration of motion planning in both simple and complex environments. PyBullet provided a flexible environment for initial implementation, while the Jogramop framework ensured that the algorithms could be rigorously tested in standardized scenarios.

## 4.5 Experimental Setup

#### 4.5.1 Simulation Environments

The experiments were conducted in two distinct simulation environments to implement and test the algorithms:

* **PyBullet**: In the first phase, the algorithms—RRT, RRT\*, JPlusRRT, IK-RRT, and BIK-RRT—were implemented in PyBullet, a real-time physics simulation engine. A 6-DOF Franka Panda robotic arm was used for motion and grasp planning tasks. The scenarios tested in this environment were relatively simple, focusing on basic object manipulation, obstacle avoidance, and path planning tasks. This setup allowed for flexibility and quick iterations, making it ideal for initial algorithm testing and development.
* **Jogramop Framework**: In the second phase, the RRT\* algorithm was tested using the Jogramop framework, which includes 20 standardized benchmark scenarios. These scenarios are designed to introduce more complex and confined environments, including narrow passages and high obstacle densities. This provided a more structured testing environment where the algorithm’s ability to handle real-world motion-planning challenges could be explored.

#### 4.5.2 Testing Parameters

Several key parameters were adjusted during the experiments to suit the nature of the algorithms and scenarios:

* **Step Size (η)**: The step size, which controls how far the robot moves at each iteration, was adjusted based on the complexity of the scenario. Larger step sizes were used in simpler environments to allow faster path exploration, while smaller step sizes were applied in confined spaces to ensure smooth and precise paths.
* **Goal Bias**: This parameter increases the likelihood of sampling near the goal, improving convergence. In complex, confined spaces, a higher goal bias was set to help the algorithms quickly reach the goal.
* **Rewiring Radius**: For RRT\*, the rewiring radius was adjusted to optimize path cost. In environments with high obstacle density, a larger radius allowed the algorithm to rewire paths for greater optimization.

These parameters were fine-tuned based on the environment's complexity to ensure optimal algorithm performance while maintaining consistency across experiments.

#### 4.5.3 Testing Scenarios

The algorithms were tested in a variety of scenarios, ranging from simple environments to more complex obstacle-rich setups:

* **PyBullet Scenarios**: These scenarios involved grasping single or multiple objects, navigating around basic obstacles like walls or blocks, and planning collision-free paths. These scenarios were designed to test the foundational behavior of each algorithm.
* **Jogramop Scenarios**: The Jogramop framework provided more challenging environments, including narrow corridors, high obstacle densities, and confined spaces. These scenarios simulated real-world constraints where precision and optimization in path planning were critical.

#### 4.5.4 Repetitions and Trials

To ensure the reliability of the results, each algorithm was tested across **100 trials per scenario**. This repetition helped minimize the impact of variability or outliers in the data. During each trial, the system logged key metrics such as planning time and success rate, and the results were averaged to provide a clear comparison of performance across different algorithms and environments.

## 4.6. Metrics for Evaluation

The performance of the algorithms was evaluated using two key metrics:

* **Planning Time**: This reflects the algorithm's computational efficiency. A shorter planning time indicates faster computation of a collision-free path.
* **Success Rate**: This measures the reliability of the algorithm in successfully finding a path to the goal. A high success rate indicates robustness in handling dynamic and obstacle-rich environments.

These metrics were chosen because they directly reflect the real-world performance of grasp and motion planning algorithms. Planning time assesses the computational cost, while the success rate measures the practical applicability of the algorithm in robotic tasks.

**4.7 Experimental Results**

* **Presentation of Results**:
  + Present the key results from both the PyBullet environment and Jogramop framework.
  + Use tables, figures, or graphs to illustrate differences in planning time, success rate, and exploration efficiency between the algorithms.
* **Result Comparison**: Compare the results of RRT, RRT\*, JPlusRRT, IK-RRT, and BIK-RRT in each scenario.
  + Highlight where each algorithm performed best or worst.
  + Discuss specific scenarios where certain algorithms outperformed others.

**4.8 Discussion of Findings**

* **Analysis of Results**:
  + Analyze the experimental results in detail.
  + Discuss how each algorithm's performance aligns with its theoretical strengths and weaknesses.
  + Highlight trends in planning time and success rates across different scenarios.
* **Algorithm Strengths and Weaknesses**:
  + For each algorithm, discuss where it excels and where it struggles (e.g., RRT\* performs well in complex environments but requires more computational resources).
* **Impact of Environment**: Discuss how the environment (e.g., PyBullet vs. Jogramop) impacted the results.
  + Was RRT\* more suitable for complex benchmark scenarios?
  + How did IK-RRT and BIK-RRT handle grasp constraints in the simpler environment?

## 4.9. Data Collection

Data was collected for each scenario by logging the planning time and success rate after every trial. The results were stored using automated logging scripts within the simulation framework, ensuring accurate tracking of each run. Each trial was repeated 100 times to minimize the effect of outliers, and any anomalous results were carefully reviewed and validated for consistency.

Table 1 Compare of average time after 100 trials of RRT Star algorithm in 20 different scenarios

## 4.9. Challenges and Adaptations

During the research, several significant challenges arose, particularly in setting up the simulation environments and implementing the algorithms. Designing environments that closely mirrored real-world tasks was both time-consuming and technically demanding, as it required the careful tuning of parameters and extensive testing to ensure the accuracy and robustness of the simulations. Furthermore, the implementation of all the algorithms within the Jogramop framework turned out to be more complex than initially anticipated. JPlusRRT, in particular, posed difficulties due to its intricate planning and optimization requirements, demanding careful attention to both memory efficiency and computational complexity.

The original goal was to implement four distinct algorithms in the Jogramop framework. However, due to time constraints and the complexity of the task, only the RRT\* algorithm was fully implemented and tested. The adaptation of this plan allowed for a more focused comparison between RRT\*, JPlusRRT, and IK-RRT, the latter two of which had already been integrated into the framework. This shift in focus highlighted both the strengths and limitations of the RRT\* algorithm.

One of the most challenging aspects of the project was running RRT\* across 20 different scenarios, each tested 100 times. This involved substantial computational effort and highlighted the high computational cost of RRT\*, especially when scaling to more complex environments. The algorithm required careful configuration, tuning of parameters such as step size and goal direction probability, and constant monitoring to ensure the results were consistent. Balancing this level of computational demand with the practical limitations of hardware and time was a significant hurdle in the project, and it underscored the importance of computational efficiency in algorithm design and simulation testing.