# Methodology

1. Introduction

This section describes the methodology used to compare four path planning algorithms—RRT\*, IKRRT, IKRRTOptimized, and JPlusRRT—in terms of path length, planning time, and success rate. The comparison aims to identify the strengths and weaknesses of each algorithm in different robotic path planning scenarios.

# 2. Experimental Setup

## 2.1. Simulation Environment

## 2.2. Robot and Environment Configuration

The robot used in the experiments is a 6-DOF manipulator with a gripper.

# 3. Algorithms Implementation

## 3.1. Description of Algorithms

### RRT\*:

The RRT\* (Rapidly exploring Random Tree Star) algorithm is an incremental sampling-based method used in motion planning, which is designed to ensure both probabilistic completeness and asymptotic optimality. This algorithm incrementally builds a tree of feasible trajectories rooted at the initial state, exploring the Configuration space in a manner that tends towards the goal configuration while optimizing the path. The following details highlight the key features and process of the RRT\* algorithm as discussed in the referenced paper by Karaman and Frazzoli (Karaman and Frazzoli, 2011).

#### Key Features of RRT\*

1. **Asymptotic Optimality**: Unlike its predecessor, RRT, the RRT\* algorithm is asymptotically optimal. This means that as the number of samples approaches infinity, the solution path converges to the optimal path.
2. **Probabilistic Completeness**: RRT\* retains the property of probabilistic completeness, ensuring that it will find a solution if one exists given sufficient time and samples.
3. **Incremental and Anytime Nature**: RRT\* is an incremental algorithm, meaning it can provide a feasible solution quickly and then improve the solution over time as more samples are added.

#### Algorithm Steps

1. **Initialization**: The algorithm starts with an initial state and initializes an empty tree.
2. **Sampling**: At each iteration, a random sample from the state space is generated.
3. **Nearest Neighbor Search**: The nearest vertex in the tree to the random sample is identified.
4. **Steering**: A new node is generated by moving from the nearest neighbor towards the random sample, constrained by a step size (η).
5. **Collision Checking**: The new node is added to the tree only if the path from the nearest neighbor to the new node is collision-free.
6. **Cost Calculation and Rewiring**:
   * **Cost Calculation**: For each new node added, the algorithm calculates the cost from the start node to this new node.
   * **Rewiring**: The algorithm attempts to connect the new node to nearby nodes (within a radius). If the new connection offers a lower cost path to these nearby nodes, the tree is rewired to reflect this improved path.
7. **Goal Checking**: The algorithm checks if the newly added node brings the tree closer to the goal and potentially connects it.

#### Important Parameters

* **γ (Gamma)**: A scaling factor used to determine the radius for rewiring connections. It ensures that the rewiring process considers a sufficient number of nearby nodes.
* **η (Eta)**: The maximum step size that the algorithm can move towards the random sample in one iteration.

IKRRT: A variant of RRT that incorporates inverse kinematics to handle high-dimensional configuration spaces.

IKRRTOptimized: An enhanced version of IKRRT with additional optimization techniques for better path quality.

JPlusRRT: An innovative RRT variant that combines Jacobian-based steering with RRT principles for improved performance.

## 3.2. Integration with the Robot

Each algorithm was adapted to work with the 6-DOF manipulator. Modifications were made to ensure compatibility with the robot’s kinematics and dynamics, and to handle the specific constraints of the simulation environment.

# 4. Evaluation Metrics

## 4.1. Path Length

Path length is measured as the sum of the Euclidean distances between consecutive waypoints in the path. It is a crucial metric as shorter paths are generally more efficient.

## 4.2. Planning Time

Planning time is recorded as the time taken from the start of the algorithm until a valid path is found. Timing functions in Python (e.g., time.time()) were used to measure this metric.

## 4.3. Success Rate

Success rate is calculated as the percentage of trials in which an algorithm successfully finds a valid path within a predefined time limit (e.g., 60 seconds).

# 5. Experimental Procedure

## 5.1. Repeated Trials

Each algorithm was tested in 1000 trials to ensure statistical significance. Each trial involves finding a path from a given start to a goal position with varying obstacle configurations.

## 5.2. Varying Conditions

Experiments were conducted under varying conditions, including different start and goal positions and different numbers of obstacles in different environments. This helps in understanding the robustness and adaptability of each algorithm.

## 5.3. Data Collection

Data on path length, planning time, and success rate were collected during each trial. Automated scripts were used to log these metrics for subsequent analysis.

# 6. Data Analysis

## 6.1. Statistical Analysis

The collected data were analyzed using statistical methods such as mean, standard deviation, and ANOVA to compare the performance of the algorithms. These methods help in identifying significant differences in performance metrics.

## 6.2. Visualization

Results were visualized using bar charts and line graphs to compare the path length, planning time, and success rate of each algorithm. These visualizations provide a clear and concise way to present the findings.

## 7. Summary

The methodology outlined above ensures a comprehensive and systematic comparison of the four path planning algorithms. By evaluating path length, planning time, and success rate under varied conditions, the study aims to provide insights into the efficiency and effectiveness of each algorithm in robotic path planning.