RBE550: Motion Planning

Motion Planning HW3

Student 1: Harsh Chhajed
Student 2: Pranay Katyal

Question 1

The visibility graph and the Probabilistic Roadmap (PRM) are two different approaches used in robot path planning. Both methods capture the continuous configuration space using a graph structure but employ different strategies for doing so. Below is a comparison of these two methods along with scenarios where one may perform better than the other.

Visibility Graph

Description:
In the visibility graph method, nodes are created at the start and goal locations, as well as at every vertex of the polygonal obstacles. An edge is added between any two nodes if the straight line segment connecting them does not intersect any obstacle (the boundaries of obstacles count as valid connections).

Advantages:

- **Optimality:** In 2D environments, it yields the shortest possible path since it directly connects vertices that are "visible" to each other.
- **Deterministic:** The constructed graph is unique for a given environment.

Ideal Scenario:
The method works particularly well in simple 2D environments with few obstacles (e.g., a robot navigating an open field or park with a few trees) where the number of vertices remains small.

Disadvantages:

- **Scalability:** In environments with many obstacles (or in higher dimensions), the number of vertices and edges increases quadratically, making graph construction computationally expensive.

Probabilistic Roadmap (PRM)

Description:
PRM is a sampling-based approach. It randomly samples points in the configuration space without creating the C-space obstacles, connects nearby samples (if the connecting path is collision-free), and then uses graph search algorithms to find a path between the start and goal. It has 2 phases, a Learning phase, and a Query phase. PRM can be applied to multi query planning problems.

Advantages:

- **Scalability and Flexibility:** It works well in high-dimensional spaces and complex environments with many obstacles.
- **Robustness in Clutter:** It does not require direct line-of-sight connections, making it effective in cluttered environments.

Ideal Scenario:
PRM is advantageous in high-dimensional problems (e.g., robot arm manipulation) or in environments where obstacles are dense, as it does not rely on the explicit geometry of obstacles for graph construction.

Disadvantages:

- **Non-determinism:** The resulting graph and path may vary between runs and are not necessarily optimal.

Summary Comparison

- **Visibility Graph:** Works well in simple, 2D, open environments with few obstacles (e.g., a park with trees), where the optimal (shortest) path is desired.
- **PRM:** Excels in high-dimensional or cluttered environments (e.g., robot arm in a crowded workspace or a drone navigating a complex building) where constructing a complete visibility graph is infeasible.

Question 2

For each of the three manipulators shown in Figure 1, determine the topology and dimension of the manipulator's configuration space.

1. **First Manipulator:**
   Joints: 2 Prismatic (P, P)
   Degrees of Freedom: 2 (each prismatic joint contributes 1 degree of freedom).
   Topology: Since each prismatic joint can be represented by $\mathbb{R}$, the configuration space is
   $$\mathbb{R} \times \mathbb{R} = \mathbb{R}^2.$$
2. **Second Manipulator:**
   Joints: 3 Revolute (R, R, R)
   Degrees of Freedom: 3 (each revolute joint contributes 1 degree of freedom).
   Topology: Each revolute joint corresponds to a circle S^1. Thus, the configuration space is
   $$S^1 \times S^1 \times S^1 = (S^1)^3,$$
   which is topologically a 3-dimensional torus, denoted by T^3.
3. **Third Manipulator:**
   Joints: 2 Revolute + 1 Prismatic (R, P, R)
   Degrees of Freedom: 3 (two angles from the revolute joints and one translation from the prismatic joint).
   Topology: The configuration space is the product of two circles and one real line:
   $$S^1 \times \mathbb{R} \times S^1,$$
   which can also be written as $T^2 \times \mathbb{R}$.

Question 3

Answer the following questions about asymptotically optimal planners.

(a) What is the core idea behind RRT*?

The core idea behind RRT* is to modify the basic RRT algorithm so that it not only finds a feasible path but also continuously improves the solution over time, eventually converging to an optimally low cost path as the number of samples increases. The main modifications to RRT are:

(i) **Best-Parent Selection:** Instead of connecting a new sample to its single nearest neighbor, RRT* considers all vertices within a certain radius (which shrinks as more samples are added) and chooses the vertex that minimizes the total cost from the start to the new sample. This minimizes the path cost incrementally.

(ii) **Rewiring:** After adding a new sample, RRT* checks whether existing nearby vertices can achieve a lower cost by connecting through the new sample. If so, it rewires the tree by updating the parent of those vertices. This local optimization process is key to achieving asymptotic optimality.

(b) How does Informed RRT* improve upon RRT*? Is it strictly better?

Informed RRT* improves upon RRT* by restricting the sampling domain to a subset of the state space that *can* yield a better solution than the best one found so far. Once an initial solution is obtained, an ellipsoidal subset (the *informed set*) is defined, which contains all states that could potentially lead to a path with lower cost. The algorithm then samples only within this region, which has two benefits:

- **Accelerated Convergence:** Focusing sampling on the promising region reduces the number of unnecessary samples in areas that cannot improve the solution, thereby speeding up the convergence toward the optimum.
- **Efficiency:** This targeted approach reduces the overall number of collision checks and nearest-neighbor searches required, thus improving computational efficiency.

Regarding whether Informed RRT* is strictly better than RRT*:

- **Optimality Guarantee:** Both algorithms are asymptotically optimal, meaning that given an infinite number of samples they will converge to an optimal solution.
- **Same Sample Sequence:** If both RRT* and Informed RRT* were provided the exact same sequence of samples, they would ultimately converge to the same optimal solution. Informed RRT* only differs in its sampling strategy—it accelerates convergence by discarding samples from regions that cannot improve the current solution.

Comparison of Solution Quality: Given the same sequence of samples, RRT* and Informed RRT* will ultimately find the same optimal solution, as Informed RRT* only alters the sampling domain after an initial solution is found. However, in rare cases, restricting sampling to the informed set could theoretically exclude samples that might have led to an optimal solution through a different, unexpected path in highly nonconvex or discontinuous cost landscapes. That said, in practical scenarios, Informed RRT* is almost always preferable due to its faster convergence.

Question 4

Suppose five polyhedral bodies float freely in a 3D world and each is capable of both rotating and translating. If these bodies are treated as a single composite robot (and they are not attached to each other), then:

- **Individual Configuration Space:** Each polyhedral body in 3D has a configuration space given by $SE(3)$, which is the space of all rigid body motions. The group $SE(3)$ is homeomorphic to $\mathbb{R}^3 \times SO(3)$ and has dimension 6.
- **Composite Configuration Space:** Since the bodies are not attached to each other, the composite configuration space is the Cartesian product of the individual configuration spaces. Therefore, the composite configuration space is:
$$SE(3) \times SE(3) \times SE(3) \times SE(3) \times SE(3) = [SE(3)]^5.$$
- **Dimension:** The dimension of the composite configuration space is $5 \times 6 = 30$.
- **Topology:** The topology of the composite configuration space is the product topology of five copies of $SE(3)$.

Project Exercise 1: ChainBox State Space and Collision Checking

(a) createChainBoxSpace

The `createChainBoxSpace` function constructs the state space for the chainbox robot.

- The chainbox robot consists of a square base of side length 1 and a 4-link chain (each link of length 1) whose first joint is located at the center of the base.
- The base position is represented in $\mathbb{R}^2$, and its orientation is represented in $SO(2)$.
- Each revolute joint in the chain contributes one degree of freedom and is modeled as an element of $SO(2)$.

Thus, the configuration space is a compound space given by:

$$\mathbb{R}^2 \times SO(2) \times (SO(2))^4.$$

The dimension is $2 + 1 + 4 = 7$. Topologically, it is equivalent to $\mathbb{R}^2 \times (S^1)^5$, where the first S^1 represents the base orientation and the remaining four represent the chain joints.

(b) setupCollisionChecker

The `setupCollisionChecker` function implements collision checking for the chainbox robot:

- **Base Validity:** Checks that the square base (side length 1) stays within the workspace $[-5, 5] \times [-5, 5]$.
- **Obstacle Collision:** Verifies the base does not intersect any obstacles.
- **Chain Self-Collision:** Ensures the chain links do not self-intersect (except at their connecting joints) and do not collide with obstacles.

Below are brief descriptions of the subfunctions/classes used in this collision checker:

- `computeSquareCorners(...)`: Computes the (x,y) coordinates of the square base corners given its center and orientation.
- `checkSegmentIntersection(...)`: Checks if two line segments intersect; used for both obstacle and self-collision detection.
- `checkBoundary(...)`: Confirms that all corners (or endpoints) remain within the specified boundary box.
- `checkLinkCollision(...)`: Given two links (line segments), checks whether they intersect each other or an obstacle.
- `ChainboxRobot` (class): Maintains the chain configuration, link endpoints, and base state. It calls the helper functions above to evaluate collisions.

Project Exercise 2: Narrow Passage Problem and PRM Benchmarking

(a) makeScenario1

- The environment, start, and goal are provided by the function `makeScenario1`.
- We selected the **RRTConnect planner for this scenario**.
- The solution is visualized and saved as `narrow.txt` (the path matrix) and `narrow.gif` (animation from the visualizer).

Results:

(b) benchScenario1

We benchmarked different PRM sampling strategies (Uniform, Gaussian, Bridge, and Obstacle-based sampling) on Scenario 1.

- **Benchmark Setup:** The benchmarking is performed using OMPL's `Benchmark` tool and PlannerArena.com
- **Observations:** Based on our experiments, **PRM with Obstacles** performed the best in terms of planning time and solution quality. This is likely because the narrow passage environment benefits from consistent and evenly distributed samples.

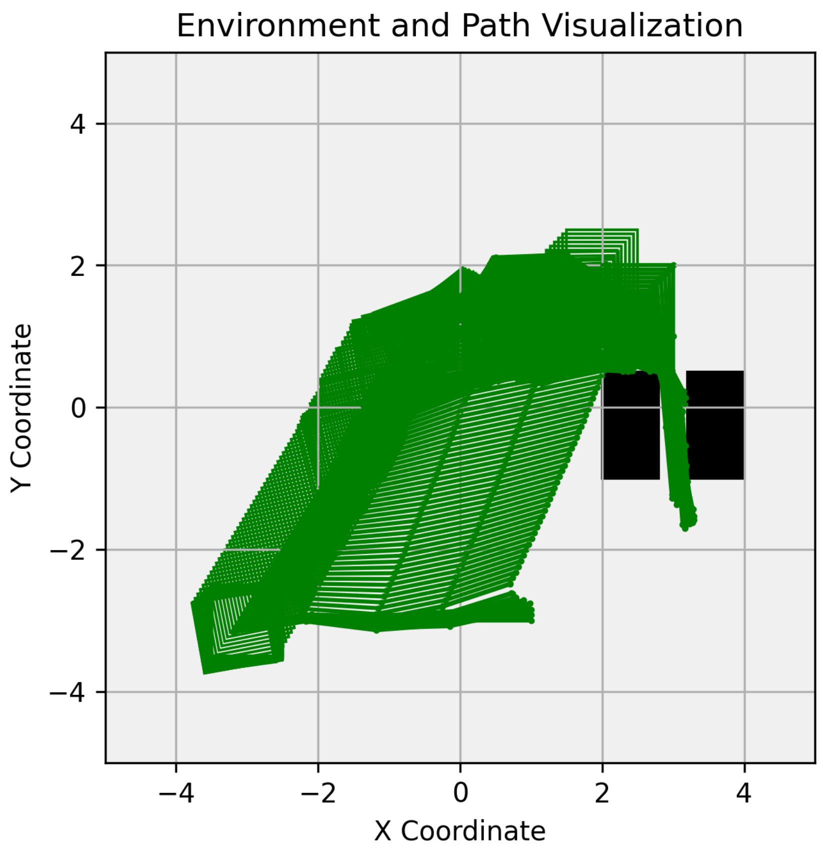
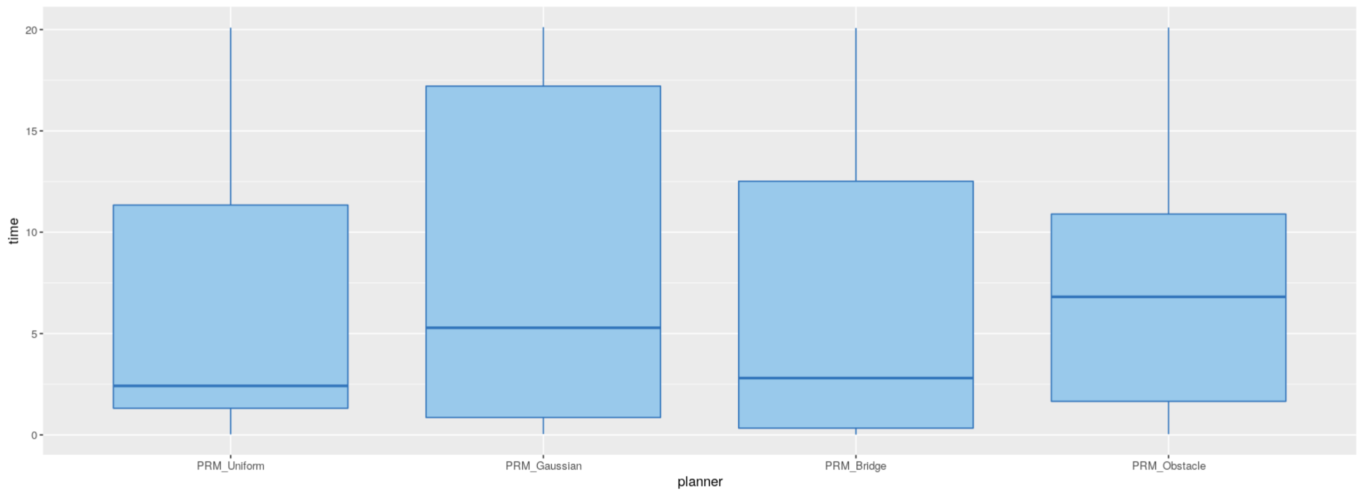
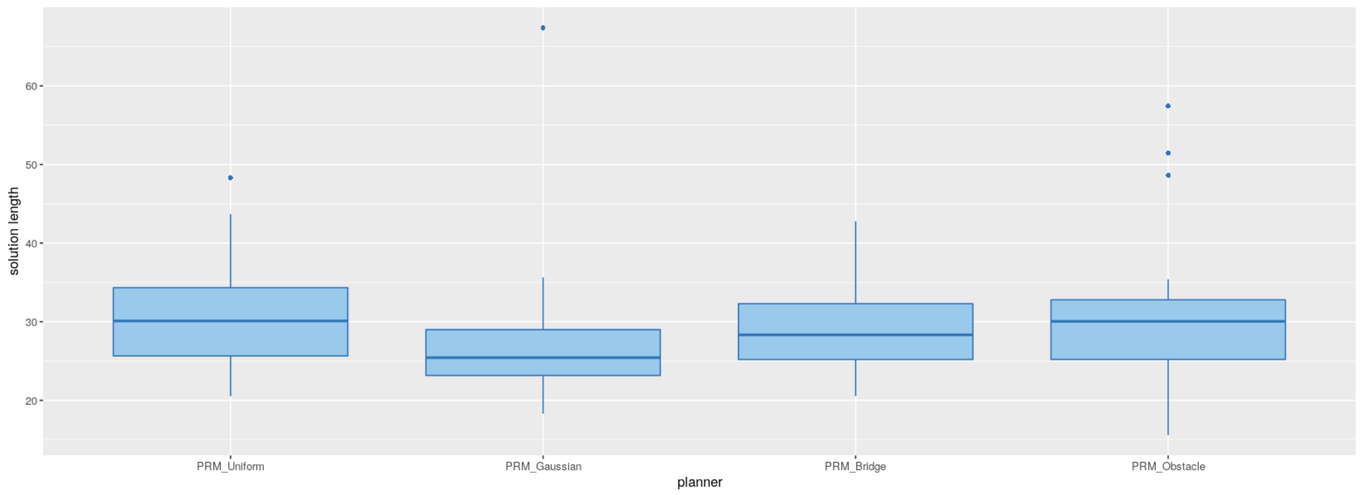


Figure 1: Narrow passage scenario result image.

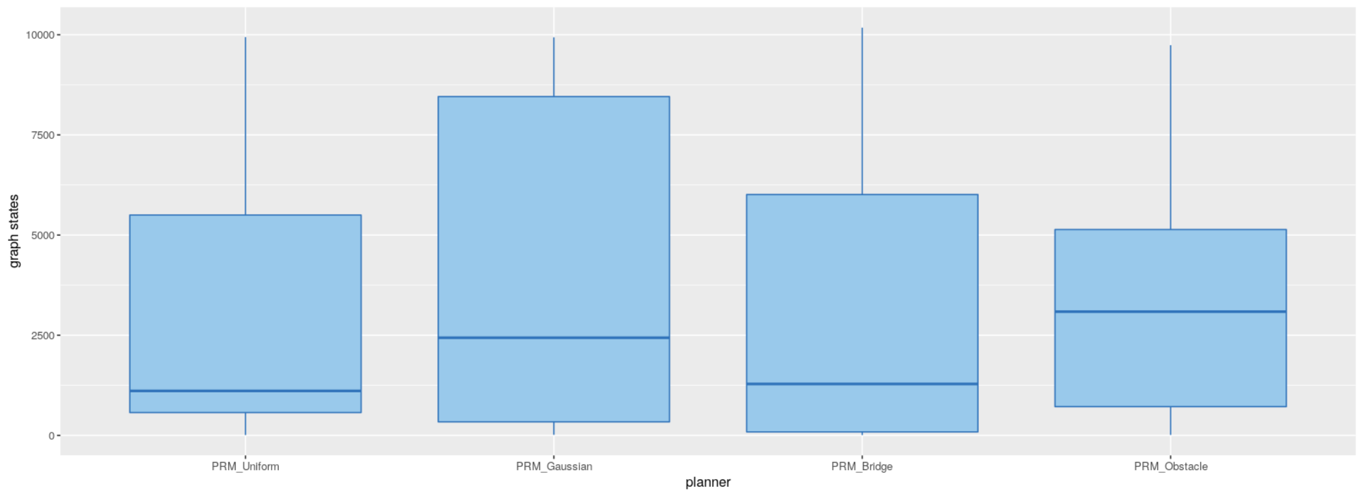
Results:
Benchmarking against Time.



Benchmarking against Solution Length.



Benchmarking against Graph States.



Project Exercise 3: Maximizing Clearance and AO Planner Benchmarking

(a) planScenario2

Scenario 2 involves planning a path that maximizes workspace clearance.

- The environment is provided by `makeScenario2`.
- We use a custom clearance objective, implemented via the `MaximizeMinClearanceObjective`, which approximates clearance by measuring the distance from the base center to the obstacle corners.
- An asymptotically optimal planner (RRT*) is used for planning.

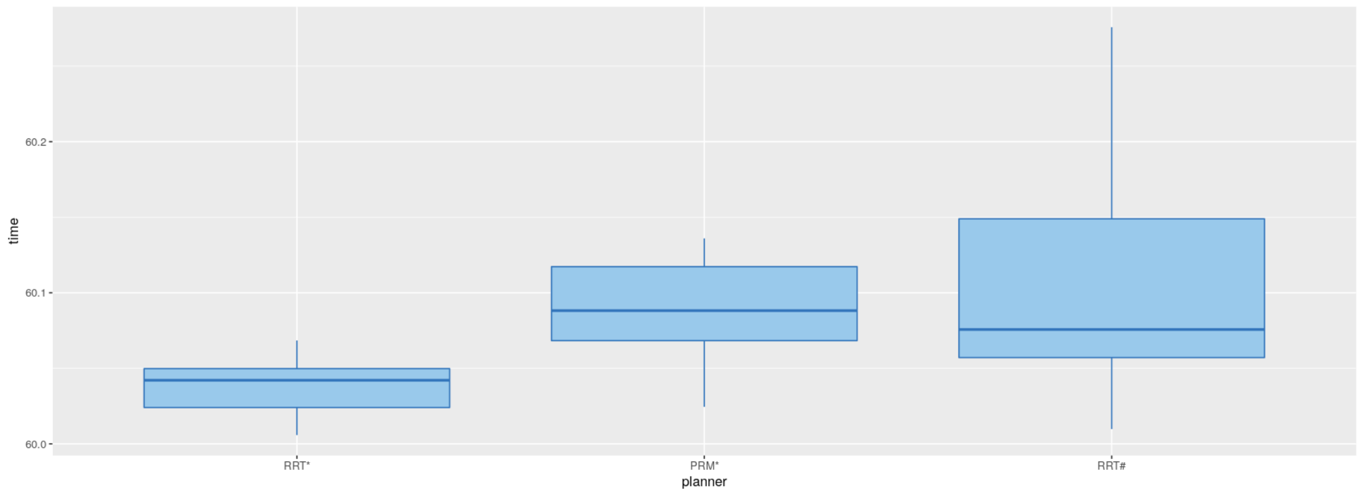
The solution path is post-processed (interpolation and simplification) and saved as `path2.txt` and `clear.gif` (visualization).

(b) benchScenario2

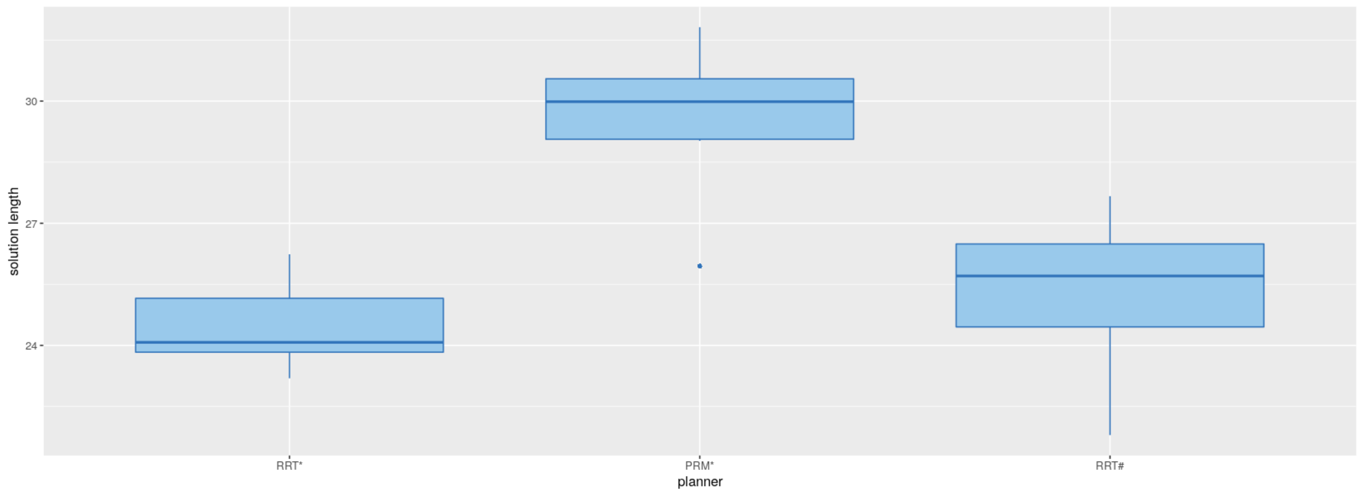
We also benchmarked asymptotically optimal (AO) planners (RRT*, PRM*, and RRT#) on Scenario 2 using the custom clearance objective.

- **Benchmark Setup:** The benchmarking is performed for 60 seconds over 10 runs using OMPL's Benchmark tool.
- **Observations: RRT Connect** produced paths with higher clearance (lower cost according to the clearance objective) and with lower variance across runs. This indicates that RRT-Connect effectively balances exploration and cost minimization in environments with narrow passages.

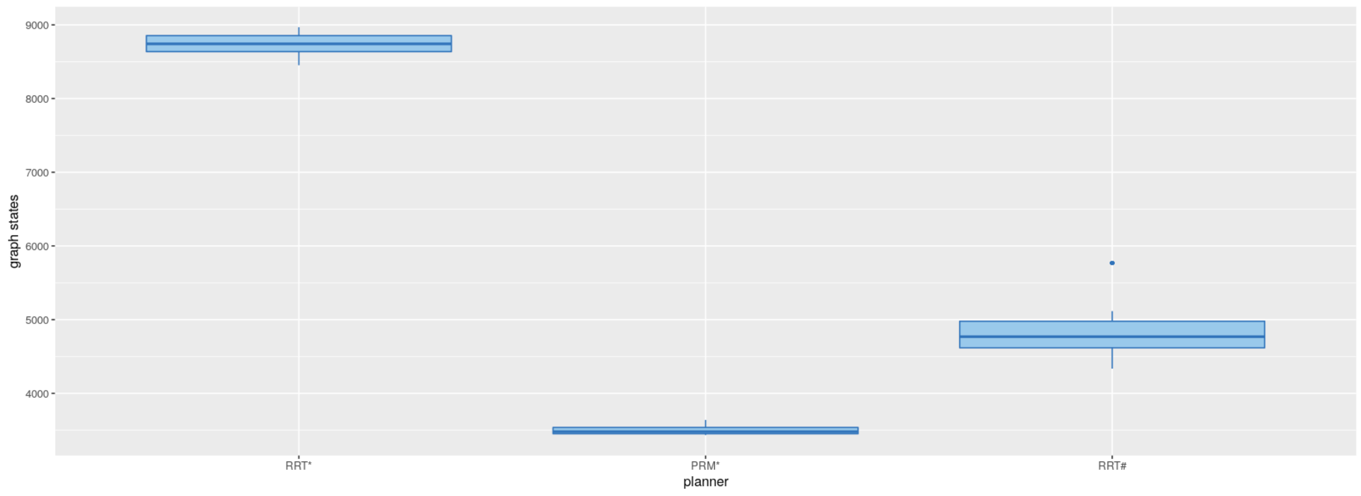
Results:
Benchmarking against Time.



Benchmarking against Solution Length.



Benchmarking against Graph States.



Conclusion

In this project, we developed and optimized a motion planning framework for a BoxChain robot with a 4-link chain. We constructed a custom compound state space and implemented collision checking for both the robot base and its chain links. For a narrow passage scenario, RRTConnect was selected as the most efficient planner for this case.

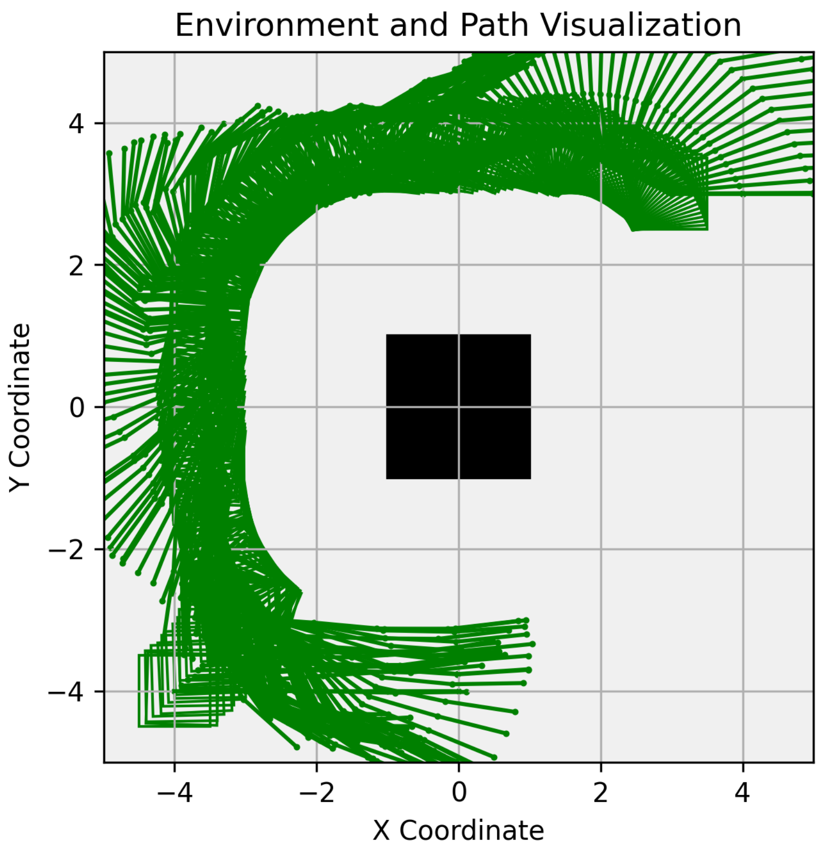
We also compared different PRMs and observed that when time is the parameter, PRM Obstacles outperformed the other PRM variants. Whereas PRM Bridges consistently performed better in terms of solution length. But in terms of graph states PRM Obstacles outperformed it yet again, yielding PRM Obstacles as the ultimate sampling strategy.

While in a clearance-optimization scenario, RRT* (and its benchmarked counterparts) were used to check maximize the clearance from obstacles. Our benchmarking experiments (placeholders for detailed results and visualizations) demonstrate the effectiveness of these planners under different environment conditions.

We Notices that RRT* performed better in terms of Time, whereas PRM* performed better in terms of graph states, or number of samples, but RRTConnect gave the most stable performance and performed consistently well.

BONUS

the clearance function although we were unsure how it was supposed to be implemented, we were able to get a Good clearance, by tuning the parameters. which hopefully is to the satisfaction!


Environment and Path Visualization
Y Coordinate
X Coordinate