RBE550: Motion Planning

Project 3: Boxing with a Chain

Student 1: Dhruv, Agrawal
Student 2: Shreyas, Khobragade

1 Theoretical Questions

1. **Recall the visibility graph method. Similar to the PRM, the visibility graph also captures the continuous space using a graph structure. Compare these two methods. For each method, provide at least one scenario in which it would work well while the other would not. Justify your answer.**

   - The visibility graph method and the Probabilistic Roadmap (PRM) method both capture the continuous space using graph structures, but they have different approaches and are better suited to different types of environments.
   - The visibility graph method finds a collision-free path for a robot by constructing a graph from vertices that represent important points in the environment (e.g., vertices of obstacles and start/end points) that can 'see' each other (i.e., are not obstructed by obstacles), and edges connect these visible points, allowing for the calculation of the shortest path between a starting and ending location by navigating through these visible connections.
   - The idea behind PRM method is to take random samples from the configuration space of the robot, test them for whether they are in the free space, and use a planner to attempt to connect these configurations to other nearby configurations by a collision-free path. The starting and goal configurations are also added, and a graph search algorithm can be applied to the resulting graph to determine a path between the starting and goal locations.
   - Visibility Graph method works well in simple 2D environments composed of convex polygonal obstacles and is mostly used for 2D path planning. Example of this can be a hall or a room occupied by furniture, but our objective is only to navigate in this space. This is because in these types of environments, the visibility graph can efficiently capture all feasible paths since the number of vertices is manageable and the visibility edges can be computed in polynomial time. Also, the graph provides a collision-free path from the start to goal without any need for large number of samples.
   - But, the Visibility Graph method does not work well in higher dimensional spaces or in environments filled with many complex obstacles. Example of this can be a cabinet with many items and our task is to move a certain item using a robot arm. Another example can be a navigating a space filled with complex or non-polygonal structures likes spheres. This is because as soon as we move to higher dimension that 2D, there is an explosion in the number of edges to calculate and this makes the task very computationally expensive. And for objects like spheres with no vertices, generating a visibility graph is even more difficult.
   - The PRM method works very well in high-dimensional spaces, for example in tasks such as motion planning for a robot in 3D or planning for robot arms which has multiple joints. This is because due to random sampling, PRM method can efficiently explore large complex spaces without explicitly representing every obstacle, making it suitable for planning in complex environments.
   - But, PRM method may not work very efficiently in a simple, lower dimensional environment with clear paths. Because it may generate many samples that are unnecessary and might lead to longer computation times due to the random sampling. For these type of scenarios visibility graph would be more efficient as there is no overhead of sampling and connecting random nodes that may not yield any significant advantages.

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

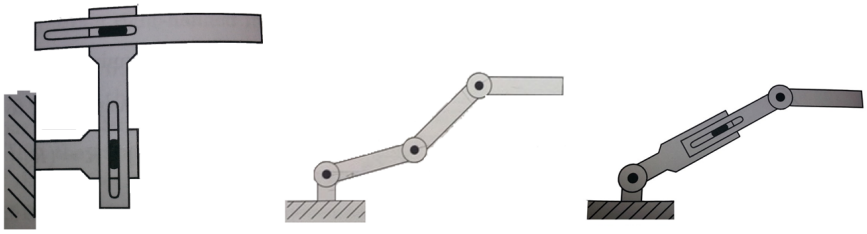


Figure 1: From left to right: a manipulator with two prismatic joints, a manipulator with three revolute joints, and a manipulator with two revolute joints and a prismatic joint.

- **For the first manipulator,**
  Let the axis frames for each joint be defined as, positive z-axis pointing in upward direction, positive y-axis pointing to the right direction and positive x-axis pointing out of the plane.
  The first joint joint can translate in the z-axis, and the second joint in the y-axis.
  Thus each joint can traverse in a real line space.
  Combining the two real lines, we get a real 2D plane.
  Thus the **Topolgy** is $\mathbf{R^2}$.
  Since there are two independent motion axes, i.e z-axis and y-axis, the **dimension of C-space** is **2**.
- **For the second manipulator,**
  Let the axis frames for each joint be defined as, positive z-axis pointing out of the plane, positive y-axis pointing in the upward direction and positive x-axis pointing to the right.
  The all joints rotate about the z-axis.
  The topology of a revolute joint is S^1.
  Combining the three topologies, we get the **Topology** as $\mathbf{S^1 \times S^1 \times S^1}$
  Since the the three joints can be rotated independently and all have a single degree of freedom ,i.e rotation about z-axis, the **dimension of C-space** is **3**.
- **For the third manipulator,**
  Let the axis frames ah each joint be defined as, positive z-axis pointing out of the plane, positive x-axis pointing perpendicular to the second link in the plane and positive x-axis along the second link.
  The the first and third joints rotate about the z-axis.
  The second joint translates along the x-direction.
  The topology of a revolute joint is S^1.
  The topolgy of a prismatic joint is R.
  Combining the topologies, we get the **Topology** as $\mathbf{S^1 \times S^1 \times R}$
  Since the the three joints can be controlled independently and all have a single degree of freedom ,i.e rotation about z-axis or translation along x-axis, the **dimension of C-space** is **3**.

3. **Answer the following questions about asymptotically optimal planners:**

- **What is the core idea behind RRT*? That is, explain what modifications are done to RRT in order to make it asymptotically optimal. What methods are changed, and what changes are they?** RRT* is an asymptotically optimal planner which means that if we give a long enough time time probability 1, then the path that we will get will be optimal.
  Main Idea: Every time a new node is added to the tree, instead of just adding the node, check if that is actually the best way to reach that node from the start node.
  If there exists a better way from the existing tree to connect that node, then a re-wiring is done to reach the new node such that the path cost is optimal. Modifications done to RRT:

  – The **Extend** function of RRT finds the nearest node to the random sample in the tree. From the nearest node, it then steers towards the random node to get a new node.
    This new node is added to the tree by after checking if the path between nearest node and new node is collision free.
  – The **Extend** function in RRT* takes most of the things from RRT till the point of finding the new node to the tree.
    Then instead of adding the node directly to the tree, a radius is calculated using the equation $r_n =$

$min\left\{\left(\frac{\gamma}{\zeta_d}\frac{logn}{n}\right)^{1/d}, \eta\right\}$ and all nodes lying inside the radius are nearest neighbours and are taken into consideration for calculating the potential optimal cost path from the start node to the new node.
Once the collision free optimal cost path has been found, the new node is added to the tree which has the nearest node as its parent which leads to optimal cost path.
Then all other nearest neighbours are checked for optimality from the start node through the new node, and if any optimal path is found to that node, then the path is re-wired for that node.
These changes make RRT* asymptotically optimal.

- **How does Informed RRT* improve upon RRT*. Is Informed RRT* strictly better than RRT* ? e.g., are there any cases where RRT* can find a better solution given the same sequence of samples?**
  Improvements of Informed RRT* over RRT*:
  - **Goal biasing:** Informed RRT* uses a bounding region around the goal using the L_2 Norm as a heuristic. This means that is focuses its exploration within a specific area that is more likely to yield optimal paths. This leads to faster convergence towards the optimal path.
  - **Cost-based pruning:** Informed RRT* evaluates the cost of the paths during exploration and prunes away nodes that are likely to result in sub-optimal paths, thus reducing unnecessary computations.
  - **Sampling Efficiency:** The sampling strategy in Informed RRT* is designed to concentrate more on areas that can lead to optimal solutions, whereas RRT* samples uniformly, which may not be as efficient in complex spaces.

  Informed RRT* is essentially RRT* until the first solution is found.
  After finding a solution, Informed RRT* takes the path cost as a measure and creates a PHS around the start and goal node.
  It discards all nodes outside the PHS, which means that if any solution is found which goes through those nodes, will have a higher path cost.
  It only samples nodes within the PHS.
  This means that cost of any path found within the PHS will be less than or equal to the original solution path cost.
  Whenever a new path is found which is better than the previous path, a PHS is constructed having a more stricter bound, thus improving path optimality.
  RRT* on the other hand computes for the same optimality but taking into consideration the whole space.

4. **Suppose five polyhedral bodies float freely in a 3D world. They are each capable of rotating and translating. If these are treated as "one" composite robot, what is the topology of the resulting configuration space (assume that the bodies are not attached to each other)? What is the dimension of the composite configuration space?**
   For the calculation of the topology of the composite robot, we need to look at the topology of a single polyhedral body:
   A single body can translate and rotate freely in the 3D space: this means that it has 3 dimensions of translation (x,y,z) and 3 dimensions of rotation (w_x, w_y, w_z).
   Thus it has a 6 dimensional C-space.
   There are 5 polyhedral bodies in the space all of which have a 6 dimensional C-space.
   The total configuration space of the composite robot is thus the Cartesian product of all the individual C-spaces.
   Dimension of the total C-space is, $5 \times 6 = 30$.
   The topology of an individual body is $R^3 \times SO(3)$.
   Thus the overall topology is :

$$\mathbf{R^3 \times SO(3) \times R^3 \times SO(3) \times R^3 \times SO(3) \times R^3 \times SO(3) \times R^3 \times SO(3)}$$

2 Programming Exercises

1. **Complete the following functions in plan.cpp:**

   (a) **createChainBoxSpace: Implements the state space for the chainbox. Take the following into account:**
   - **The chainbox robot has a square base where each side is 1 and a 4 link chain each link size 1, with the first joint at the center of the box. See included gifs for a visualization.**
   - **The box center of the robot must remain within a [5,-5], boundary at all times.**
   - **For guidance on custom state spaces, refer to this tutorial.**

- **In your report provide the topological space of this robot.**

(b) **setupCollisionChecker: Implements the collision checking for the chainbox robot. Take the following into account:**

- **The robot must remain within a [5,-5] at all times. The example gifs violate this, on purpose to ensure you provide.**
- **The chain should not self-intersect with itself or the box (except for the first link)**
- **Youcan heavily leverage and modify KinematicChain.h to achieve this collision checking. Oneapproachistoimplementeverything in textttKinematicChain.h and the setupCollisionChecker will be only 1 line.**

2. **Complete the following functions in plan.cpp:**

(a) **planScenario1: Solve the narrow passage problem in Scenario 1.**

- **The environment obstacles, start, and goal are already provided to you in makeScenario1.**
- **Choose the most efficient planner from rrt, prm, or rrtconnect. to solve this problem. In your report, explain which planner you selected and why you believe it is the fastest for this scenario.**
  Figures 2, 3 and 4 show workspace images for different planners explored:

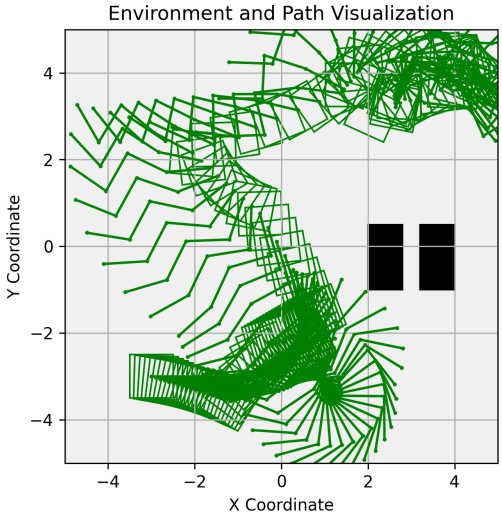


Figure 2: Path of RRT planner

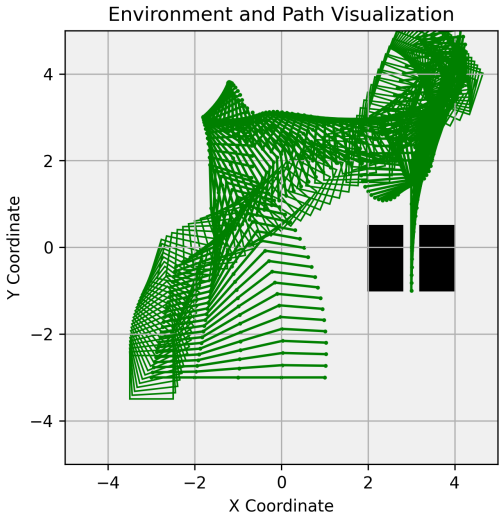


Figure 3: Path of PRM planner

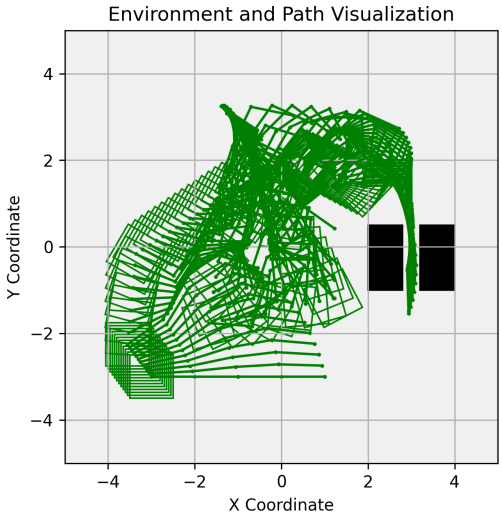


Figure 4: Path of RRTConnect planner

The table below provides the information about each planner:

	RRTConnect	**PRM**	**RRT**
States	212	272	10867
Time (s)	0.074213	0.409	10
Interpolation steps	100	100	100

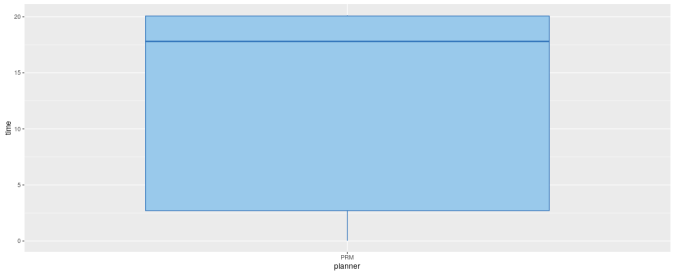
Table 1: Caption

From the above images and the table, we can see that **RRTConnect is the best approach** as it takes least time to come to a solution.
This is logically also true, since RRTConnect grows trees both near the start location and goal location and tries to connect them, which means that near the goal, it is more focused on finding the valid points in the narrow region.
Since RRT samples uniformly, it is not able focus on the narrow region near the goal and thus could not get to the goal location in the given time.
PRM was able to find the path to the goal, but it took much more time than RRTConnect to find the solution.
This is because PRM first makes a road map and for that it needs a densely sampled region at the narrow strip, which lead to a longer planning time.

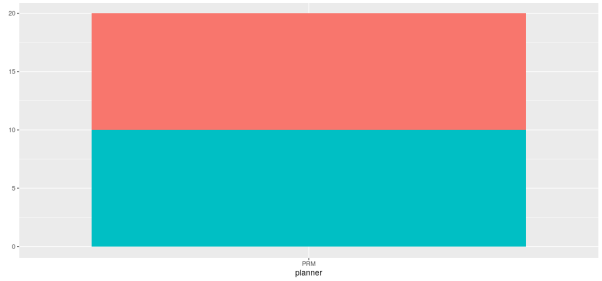
- **Use the visualizer to plot the solution, and submit the generated files as narrow.txt and narrow.gif.**

(b) **benchScenario1: Benchmark different PRM sampling strategies for Scenario1:**

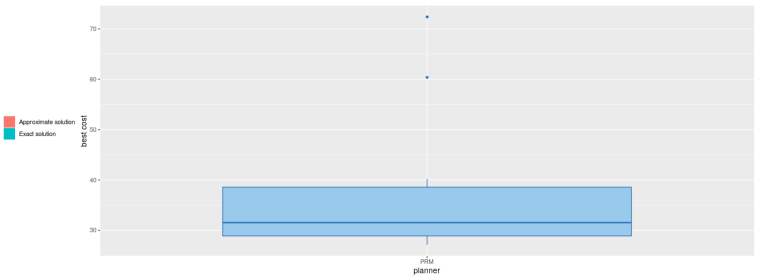
- **Benchmark (Uniform, Gaussian, Bridge, and Obstacle)-based sampling for scenario1.**
- **The example in this benchmarking demo could help.**
- **In your report, explain which sampling strategy performed the best and why. Include supporting figures from PlannerArena.**
  The Figures 9, 6, 7 and 8 show the different solution metrics for the samplers:



(a) Time

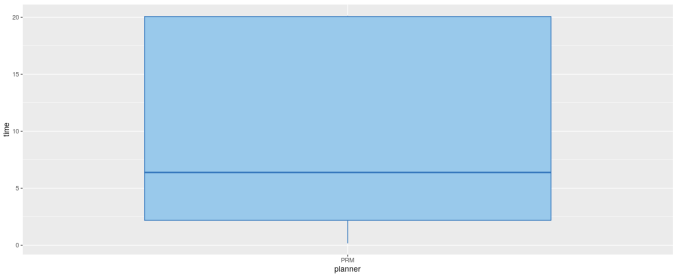


(b) Correct Solution

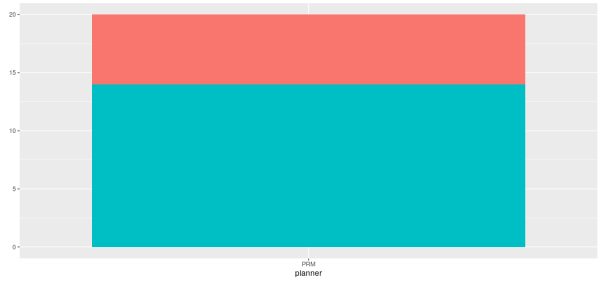


(c) Best Cost

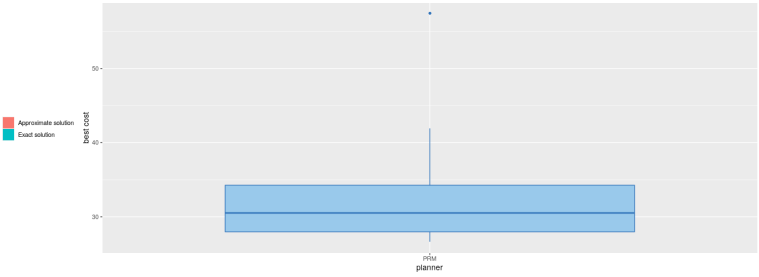
Figure 5: Uniform Sampler



(a) Time

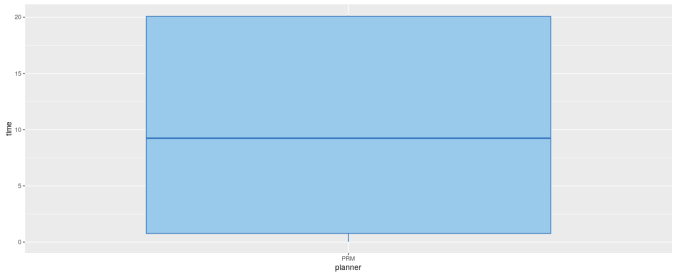


(b) Correct Solution

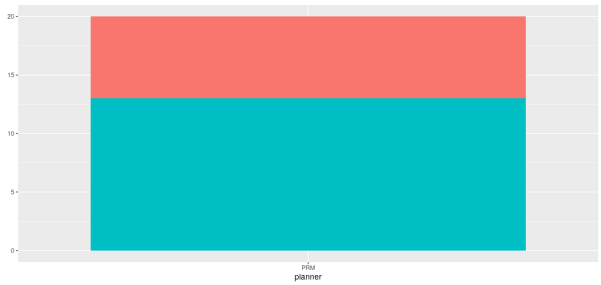


(c) Best Cost

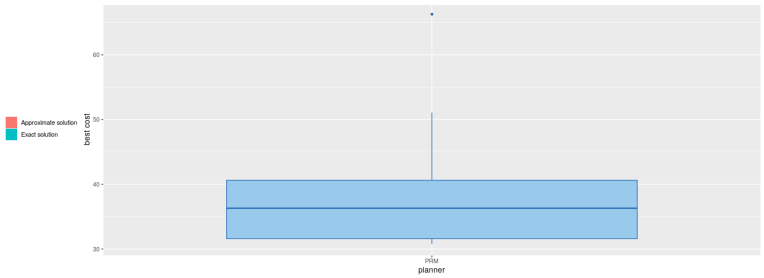
Figure 6: Gaussian Sampler



(a) Time

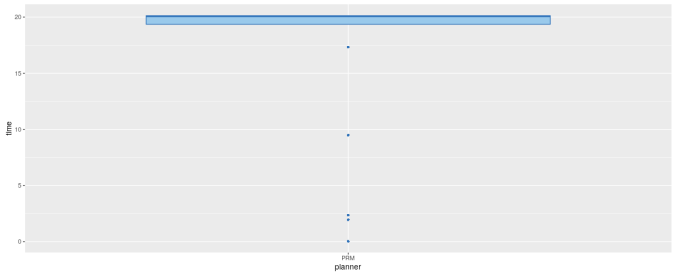


(b) Correct Solution

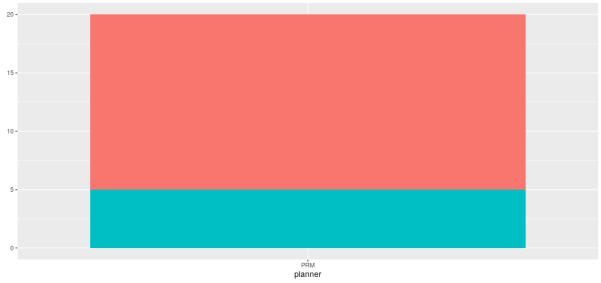


(c) Best Cost

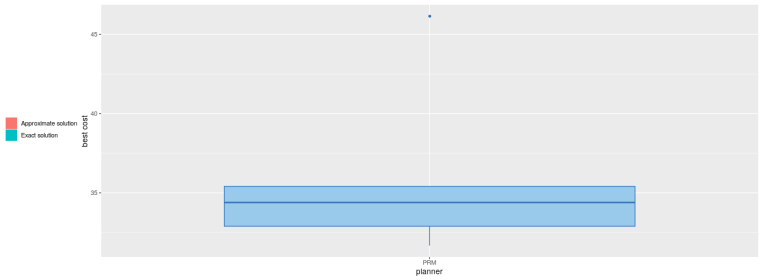
Figure 7: Bridge Sampler



(a) Time



(b) Correct Solution



(c) Best Cost

Figure 8: Obstacle Sampler

From the above figures, we can see that, the Gaussian sampler has:

i. **Lowest mean time** for reaching to a solution path.
ii. **Highest percentage** of finding an exact solution within the given time.
iii. **Lowest mean path cost**.

Thus for our case, the Gaussian Sampler is the best sampler.

3. **Complete the following functions in plan.cpp:**

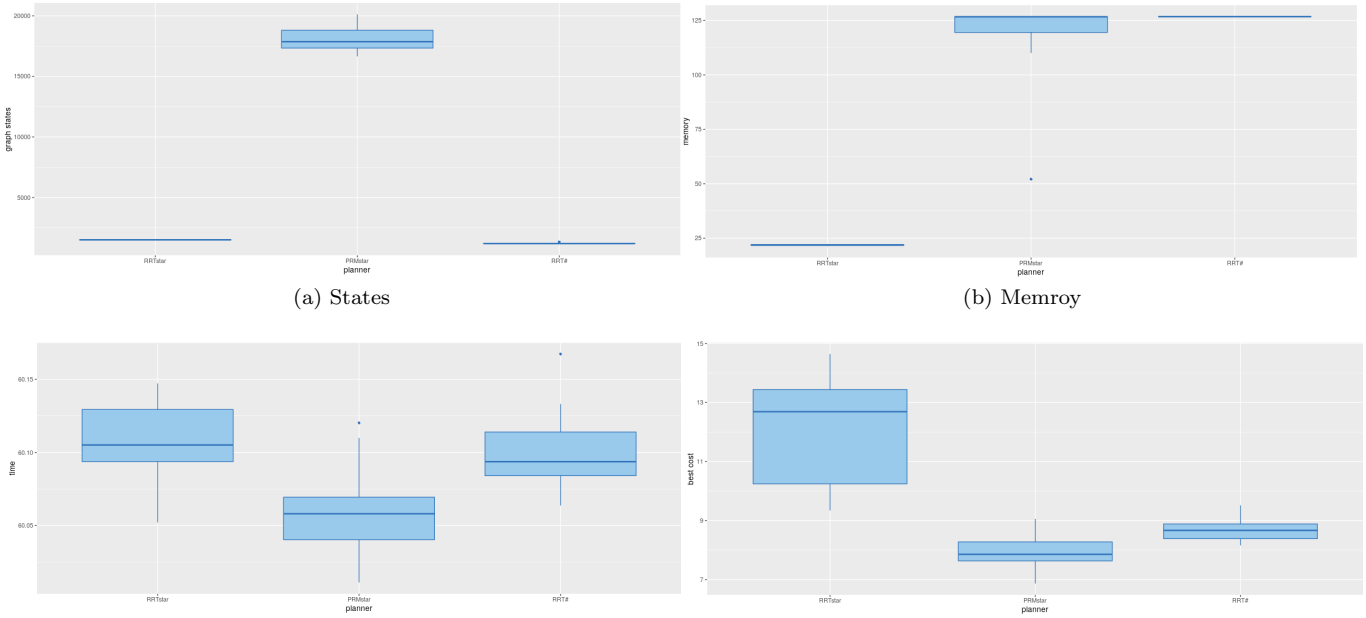
   (a) **planScenario2: Solve Scenario2 by finding a path with maximum workspace clearance.**

   - **The environment obstacles, start, and goal are already provided to you in makeScenario2.**
   - **Implement a clearance optimization function. This tutorial could be helpful optimal planning tutorial.**

- **Calculating the true c-space clearance is impractical in most situations. Simply approximating a workspace clearance by calculating the distance from the box center to the obstacle corners. will suffice for this homework.**
- **Choose an asymptotically optimal planner to solve the problem and submit your solutions as clear.txt and clear.gif from the visualizer.**

(b) **benchScenario2: Benchmark different AO planners**

- **Benchmark rrtstar, prmstar, and rrt# using your the custom clearance objective. In your report,identify the most effective planner and explain the results with figures from PlannerArena. Include the benchmarking figures from plannerarena in your submission.** The Figures 9, 6, 7 and 8 show the different solution metrics for the planners:



(a) States (b) Memroy

(c) Time (d) Best Cost

Figure 9: Uniform Sampler

From the above figures, we can see that even though the PRMstar has:

i. **Highest** number of states
ii. **Very High** memory usage.
However, it has
iii. **Least** mean solution time
iv. **Least** mean path cost

Thus, **if memory is not a constraint**, then **PRMstar** is the best planner for our case.

4. **Implement the true workspace clearance function, which calculates the minimum distance between the entire chainbox robot and the obstacles. Explain how you implemented this in your report. Submit the solution as optimal clear.gif and optimal_clear_path.txt**

5. **Clearance Challenge! The top three optimal clear_path.txt submissions with the highest total maximum clearance will receive 10, 7.5, and 5 points, respectively.**
For calculating the minimum distance between chainbox robot and the obstacles:

- Calculate the position of all vertices of the chainbox robot.
- For each vertex, calculate the distance between the vertex and each line segment of the robot.
- Check if this distance is the minimum distance and if it is, then store it the minimum distance variable as the value of this distance.