RBE550: Motion Planning

Project 4

Student 1: Sumukh, Porwal
Student 2: Piyush, Thapar

Theoretical Questions

1. **Asymptotic Optimality:**
An algorithm is asymptotically optimal if, as the number of samples approaches infinity, the solution it provides converges to the globally optimal solution. In other words, the planner is guaranteed to find the best possible path (in terms of the cost function, such as the shortest path or least energy) in the limit of infinite samples. Examples include algorithms like RRT* (Rapidly-exploring Random Trees Star) and PRM* (Probabilistic Roadmaps Star).

   Asymptotic Near-Optimality:
An algorithm is asymptotically near-optimal if, as the number of samples increases, the solution converges to one that is close to the optimal solution, but not necessarily the exact optimal one. This means the planner finds a solution that is within a certain bound of the optimal solution, but it does not guarantee convergence to the global optimum even with infinite samples. These algorithms typically sacrifice some level of optimality for other advantages, such as faster computation or easier implementation.

2. `RRT*` by itself does not inherently account for non-holonomic constraints. The standard `RRT*` algorithm assumes a system with no such constraints and generates straight-line connections between sampled points, which may not be valid for a non-holonomic system.

   However, `RRT*` can still be used for non-holonomic systems with some adjustments:

   Steering Function: The key modification is in the steering function. Instead of directly connecting two points with a straight line (which violates non-holonomic constraints), you need a steering function that generates feasible, constraint-respecting trajectories between points. This ensures the generated paths respect the kinematics of the system.

   Cost-to-Go Estimation: The algorithm needs to estimate the cost of moving between two points in a way that respects the non-holonomic dynamics, which can be more complex than in holonomic systems.

3. (a) The **configuration space** represents all possible positions and orientations the robot can take in the environment. For a differential-drive robot like the one shown in the figure, the configuration is typically described by:
   (x, y, θ), where x and y represent the position of the robot in the 2D plane. θ represents the orientation (heading) of the robot with respect to a reference axis.
   Therefore, the configuration space is 3-dimensional, expressed as $C = \mathbb{R}^2 \times S^1$, where $\mathbb{R}^2$ is the plane for position, and S^1 (the circle) represents the orientation of the robot.

   The **control space** refers to the set of possible control inputs that can be applied to the robot to influence its movement. For a differential-drive robot, the control inputs are typically the velocities of the left and right wheels.
   $u = (u_l, u_r)$, where u_l and u_r are the linear velocity of the left and right wheel respectively.
   The control space is thus 2-dimensional, $U = \mathbb{R}^2$, representing the continuous range of possible velocities for both wheels.

   The **state-space** combines the robot's configuration and its velocity information, representing the complete description of the system at any given time. For the robot in the figure, the state can be expressed as:
   (x, y, θ, u_l, u_r) where (x, y, θ) defines the robot's position and orientation, and (u_l, u_r) defines the wheel velocities.
   Thus, the state-space is 5-dimensional, $S = \mathbb{R}^2 \times S^1 \times \mathbb{R}^2$.

(b) The robot in the figure is non-holonomic because it cannot move in all directions within its configuration space freely due to the non-integrable constraints imposed by its kinematic structure. This limits its movement and distinguishes it from holonomic systems that have no such restrictions.

(c)

$$\dot{x} = \frac{x - x_o}{dt} = \frac{r}{2}(u_l + u_r)cos(\theta_o)$$
$$\implies \frac{x - 0}{1} = \frac{r}{2}(1 + 0.5)cos(0)$$
$$\implies x = \frac{3r}{4}$$

$$\dot{y} = \frac{y - y_o}{dt} = \frac{r}{2}(u_l + u_r)sin(\theta_o)$$
$$\implies \frac{y - 0}{1} = \frac{r}{2}(1 + 0.5)sin(0)$$
$$\implies y = 0$$

$$\dot{\theta} = \frac{\theta - \theta_o}{dt} = \frac{r}{L}(u_r - u_l)$$
$$\implies \frac{\theta - 0}{1} = \frac{r}{L}(0.5 - 1)$$
$$\implies \theta = -\frac{r}{2L}$$

Programming Questions

1. • Below are the figures of Planning of pendulum using RRT.
Start State: $(-\frac{\pi}{2}, 0)$ Goal State: $(\frac{\pi}{2}, 0)$

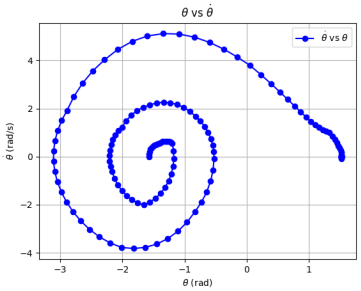

Figure 1: Torque = 3

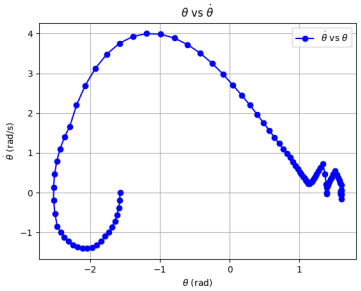

Figure 2: Torque = 5

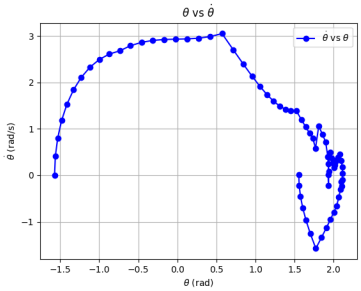

Figure 3: Torque = 10

Figure 4: θ vs $\dot{\theta}$ graph for Pendulum's planning with RRT Planner

Below are the figures of Kinodynamic Planning of square shaped car with side length 0.5 using RRT.
Start State: $(1, 1, 0, 0)$ Goal State: $(2, 8, 0, 0)$

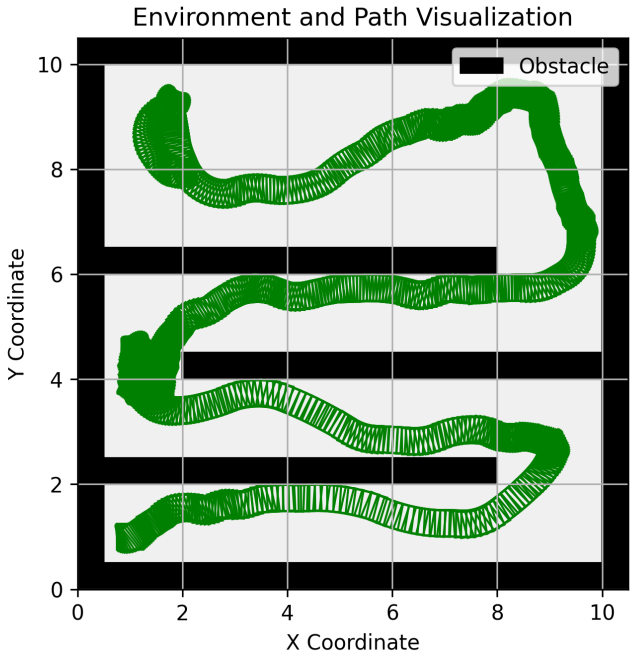

Figure 5: Kinodynamic Planning of Car using RRT

- The trajectory of the pendulum under a torque of 3 illustrates a significant challenge in reaching the upright position in an efficient manner. In this case, the applied torque is inadequate to generate a sufficiently high angular velocity to achieve the target angle in a single motion. As a result, the pendulum undergoes several oscillations, requiring an extended period to build enough speed and momentum to successfully reach the desired upright position.
Conversely, when the torque is increased to 5, the dynamics change. Although the pendulum still needs to swing back and forth to reach the target angle, the increased torque allows for a more rapid buildup of angular velocity. This results in a shorter time to achieve the upright position compared to the scenario with torque 3. The pendulum's ability to generate angular momentum increases, leading to a more efficient trajectory.
Finally, at a torque of 10, the situation markedly improves. This level of torque provides sufficient force to enable the pendulum to attain the necessary angular velocity to reach the upright position in a single,

decisive motion or we lesser number of spirals than compared with torque = 3 or torque = 5. The pendulum swings upward with the required speed, eliminating the need for oscillation and reducing the time taken to reach the goal.
Overall, this analysis highlights the critical relationship between torque and angular velocity in controlling the pendulum's motion. It emphasizes the importance of selecting an appropriate torque level to optimize performance and achieve desired positions efficiently.

2. Below are the figures of Planning of pendulum using KPIECE1.
   Start State: $(-\frac{\pi}{2}, 0)$
   Goal State: $(\frac{\pi}{2}, 0)$

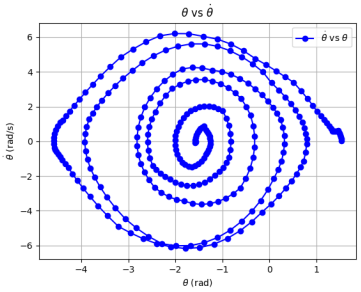

Figure 6: Torque = 3

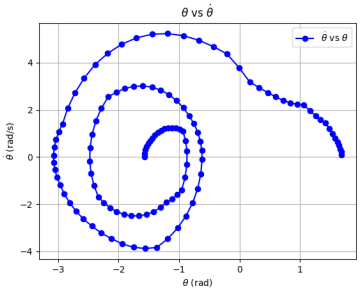

Figure 7: Torque = 5

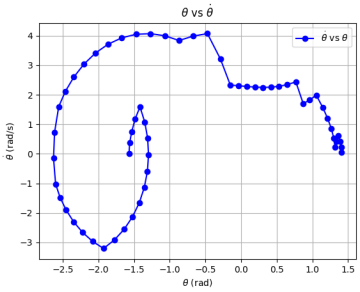

Figure 8: Torque = 10

Figure 9: θ vs $\dot{\theta}$ graph for Pendulum's planning with RRT Planner

Below are the figures of Kinodynamic Planning of square shaped car with side length 0.5 using KPIECE1.
Start State: $(1, 1, 0, 0)$ Goal State: $(2, 8, 0, 0)$

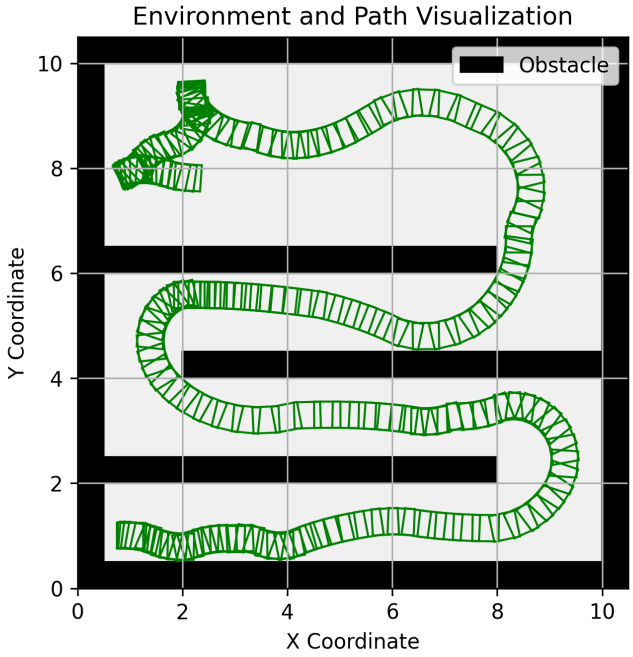

Figure 10: Kinodynamic Planning of Car using KPIECE1

3. Below are the figures of Planning of pendulum using KPIECE1.
   Start State: $(-\frac{\pi}{2}, 0)$
   Goal State: $(\frac{\pi}{2}, 0)$

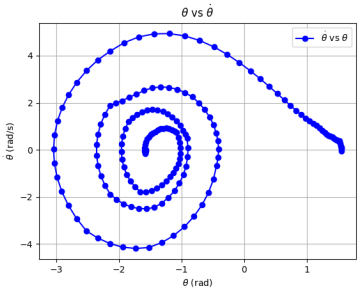

Figure 11: Torque = 3

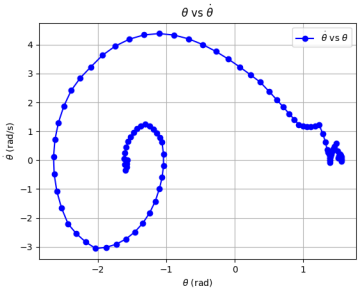

Figure 12: Torque = 5

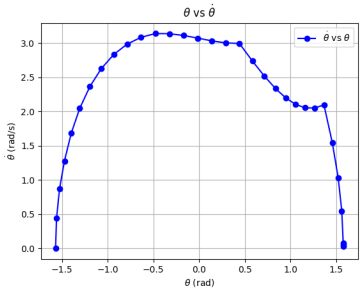

Figure 13: Torque = 10

Figure 14: θ vs $\dot{\theta}$ graph for Pendulum's planning with RRT Planner

Below are the figures of Kinodynamic Planning of square shaped car with side length 0.5 using RG-RRT.
Start State: $(1, 1, 0, 0)$
Goal State: $(2, 8, 0, 0)$

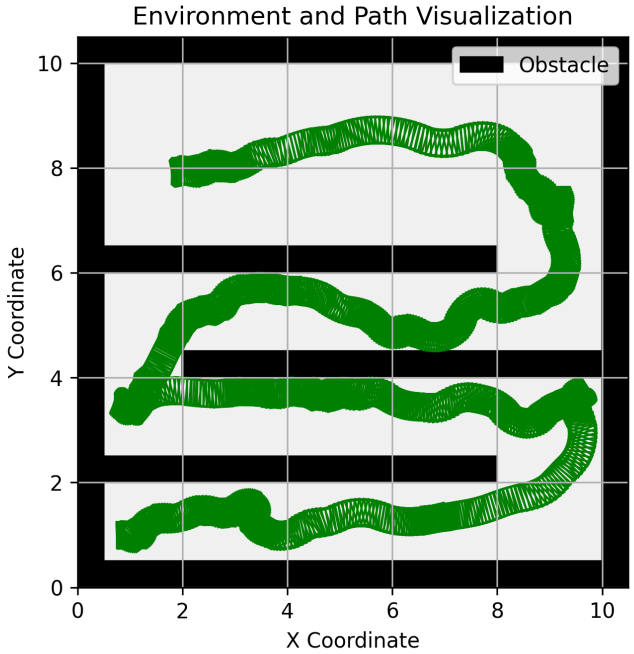

Figure 15: Kinodynamic Planning of Car using RG-RRT

4. **Benckmarking for Pendulum planning using Torque = 3:**

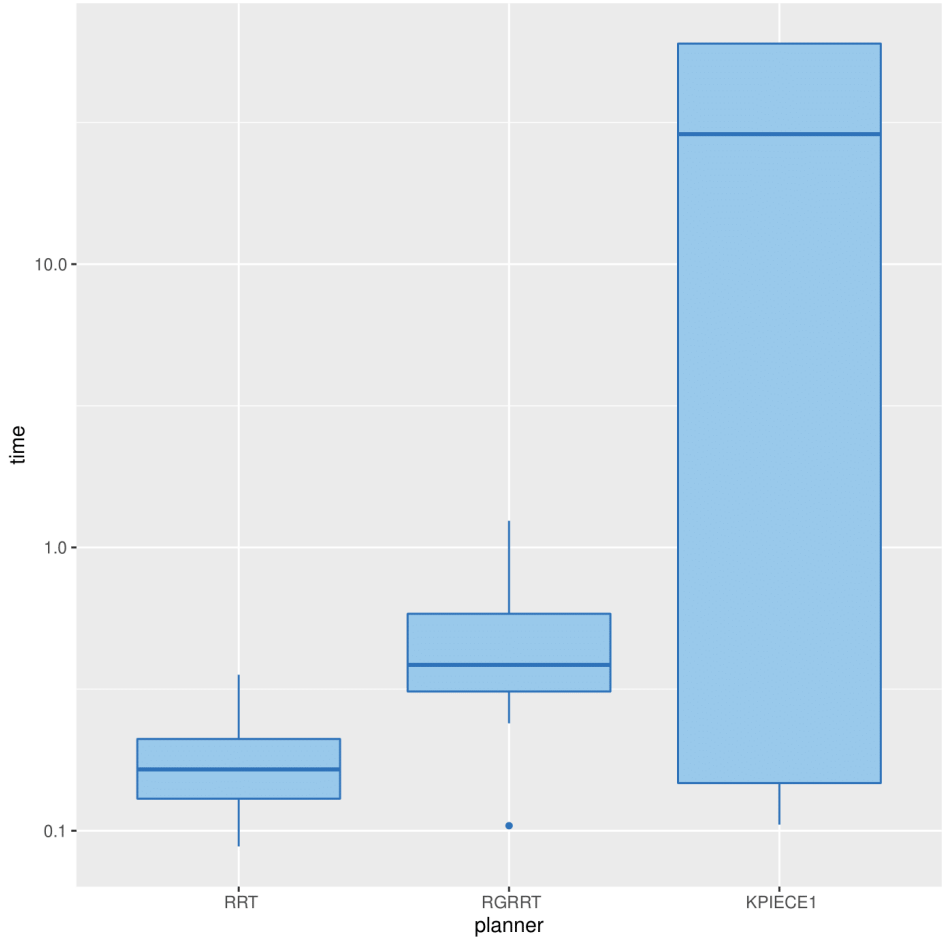

Figure 16: Pendulum Benchmark: computation time

- **RRT**:
  - The median time is the lowest among the planners, indicating that RRT generally has a quicker average planning time.
  - The interquartile range (IQR) is small, showing low variability, meaning RRT is relatively consistent in its planning times.
- **RGRRT**:
  - RGRRT has a slightly higher median time compared to RRT, indicating it takes longer on average.
  - The IQR is wider, suggesting more variability in the planning times.
  - The presence of a small outlier below the main range may represent a case where RGRRT completed planning unusually fast.
- **KPIECE1**:
  - KPIECE1 has a much higher median time compared to RRT and RGRRT, showing that it takes significantly longer to plan on average.
  - It also has a very large IQR, indicating a high variability in planning time. This suggests that while KPIECE1 may sometimes perform reasonably, it often takes a much longer time than the other planners.
  - The upper whisker is extended, highlighting instances where KPIECE1 took particularly long to complete planning.

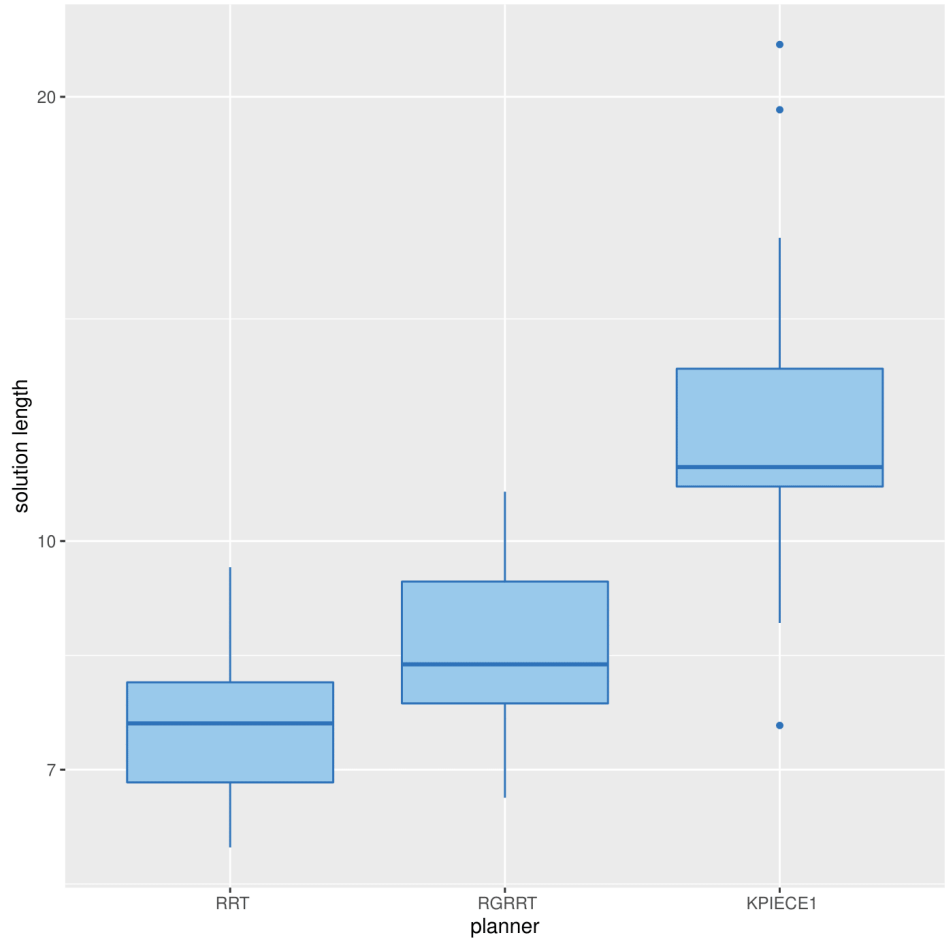

Figure 17: Pendulum Benchmark: path length

- **RRT**:
  - The median solution length for RRT is the shortest among the three planners, indicating it generally produces more direct paths in the environments with no obstacles like the one in this Pendulum's case.
  - The interquartile range (IQR) is relatively small, suggesting consistent performance in terms of solution length.
- **RGRRT**:
  - RGRRT has a slightly longer median solution length compared to RRT, indicating it may prioritize other factors over finding the shortest path.
  - The IQR is moderately large, reflecting some variability in the solution lengths.
- **KPIECE1**:
  - KPIECE1 has the longest median solution length, indicating it tends to generate more extended paths.
  - The IQR is large, showing significant variability in solution lengths, with some outliers suggesting particularly long paths.

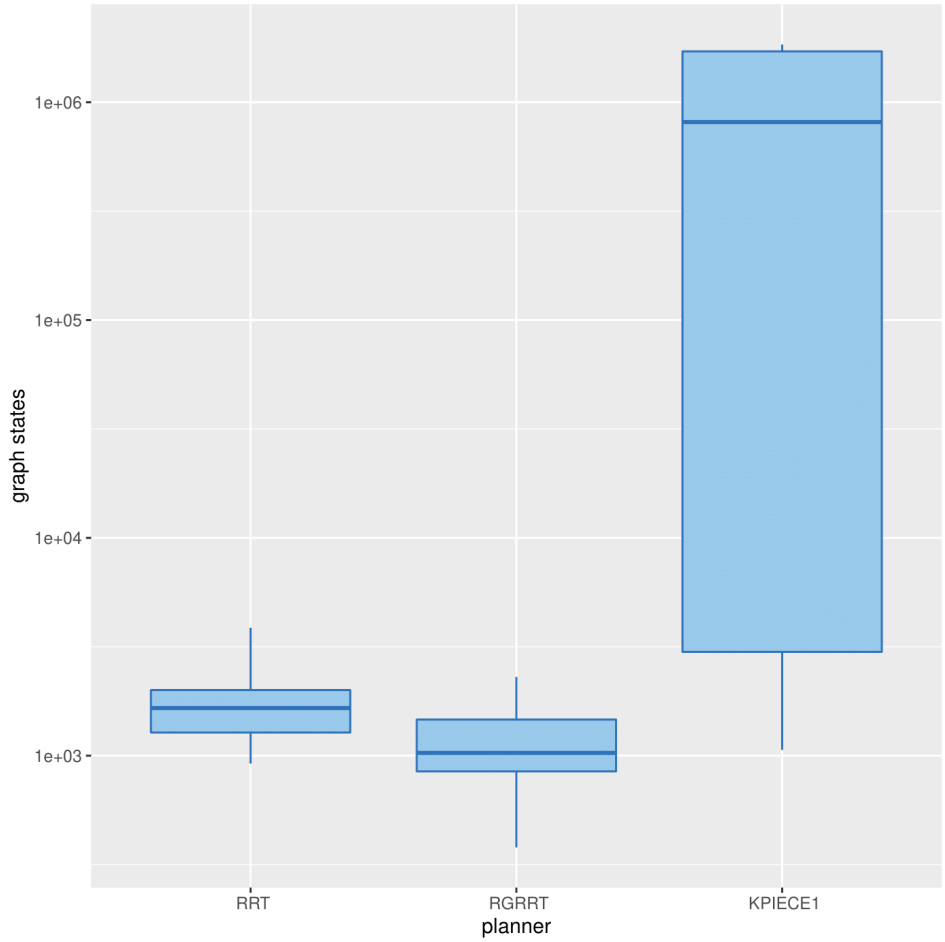

Figure 18: Pendulum Benchmark: number of tree nodes

- **RRT**:
  - The median number of graph states generated by RRT is relatively low, indicating that it typically explores fewer states but it's more than RGRRT.
  - The interquartile range (IQR) is small, showing low variability, which suggests RRT is consistent in the number of states it generates.
- **RGRRT**:
  - RGRRT has lowest median but with slightly more variability in the number of generated states.
  - The IQR is moderately larger than RRT, indicating RGRRT may sometimes explore more states depending on the planning scenario but still most of the times RGRRT generates less states than RRT.
- **KPIECE1**:
  - KPIECE1 has a significantly higher median number of states generated compared to RRT and RGRRT, indicating it explores far more states on average.
  - The IQR is very large, showing high variability, which suggests that KPIECE1's state exploration can vary widely.
  - The extended upper whisker indicates cases where KPIECE1 explored an exceptionally high number of states.

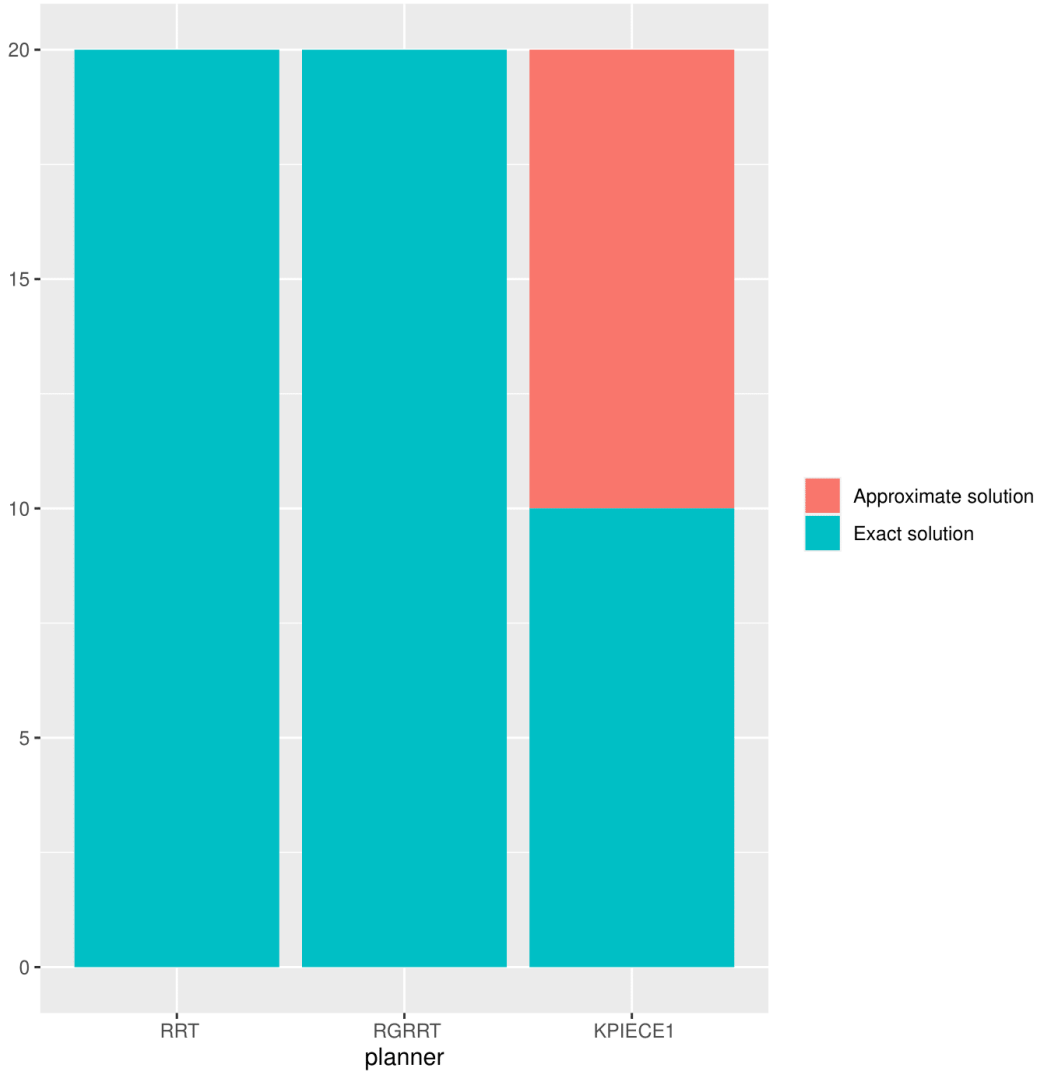

Figure 19: Pendulum Benchmark: success rate

- **RRT**:
  - In the pendulum case with environment with no obstacles RRT finds exact solution every time.
- **RGRRT**:
  - Similar to RRT, in the pendulum case with environment with no obstacles RG-RRT too finds exact solution every time.
- **KPIECE1**:
  - KPIECE1 still is not able to converge to goal and finds approximate solution 10 out of the 20 times benchmarking was ran.

Benckmarking for Car planning:

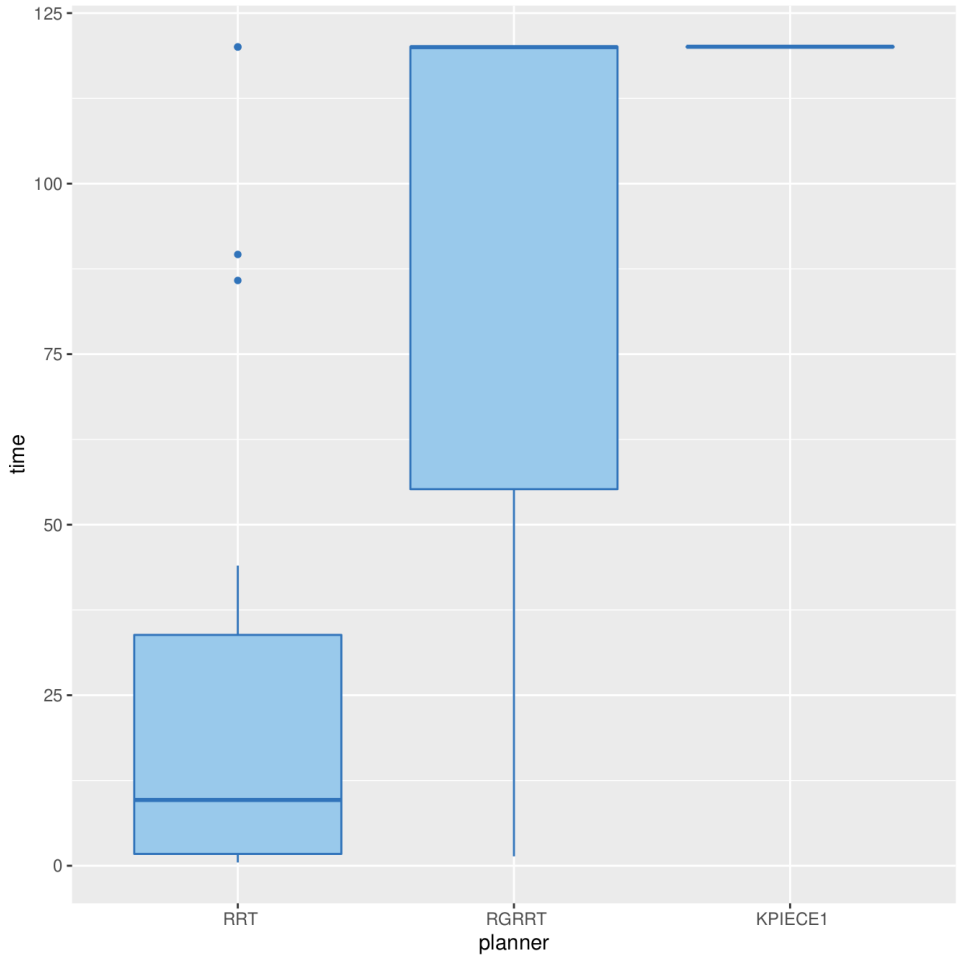

Figure 20: Car-Like Robot Benchmark: Computation Time

- **RRT**:
  - The median computation time for RRT is relatively low, indicating that it is generally efficient in finding solutions for the car-like robot.
  - The interquartile range (IQR) is small, suggesting consistent performance in terms of computation time across trials.
  - There are a few outliers with longer computation times, potentially due to the planner struggling in more complex portions of the environment.
- **RGRRT**:
  - RGRRT has the highest median computation time among the planners, indicating that it takes significantly longer to compute paths, possibly due to prioritizing path smoothness or robustness for the car-like constraints.
  - The IQR is very large, reflecting high variability in computation times, which might be attributed to different environmental configurations or specific car-like maneuvering needs.
  - The presence of multiple outliers suggests that RGRRT occasionally requires much longer computation times, likely in more complex pathfinding situations.
- **KPIECE1**:
  - KPIECE1 has a fixed computation time across trials, shown by the lack of variance. This may indicate that it operates within a deterministic or predefined time budget, making it more predictable in terms of performance.
  - The absence of variability or outliers suggests it is well-suited for environments where consistent computation time is critical.

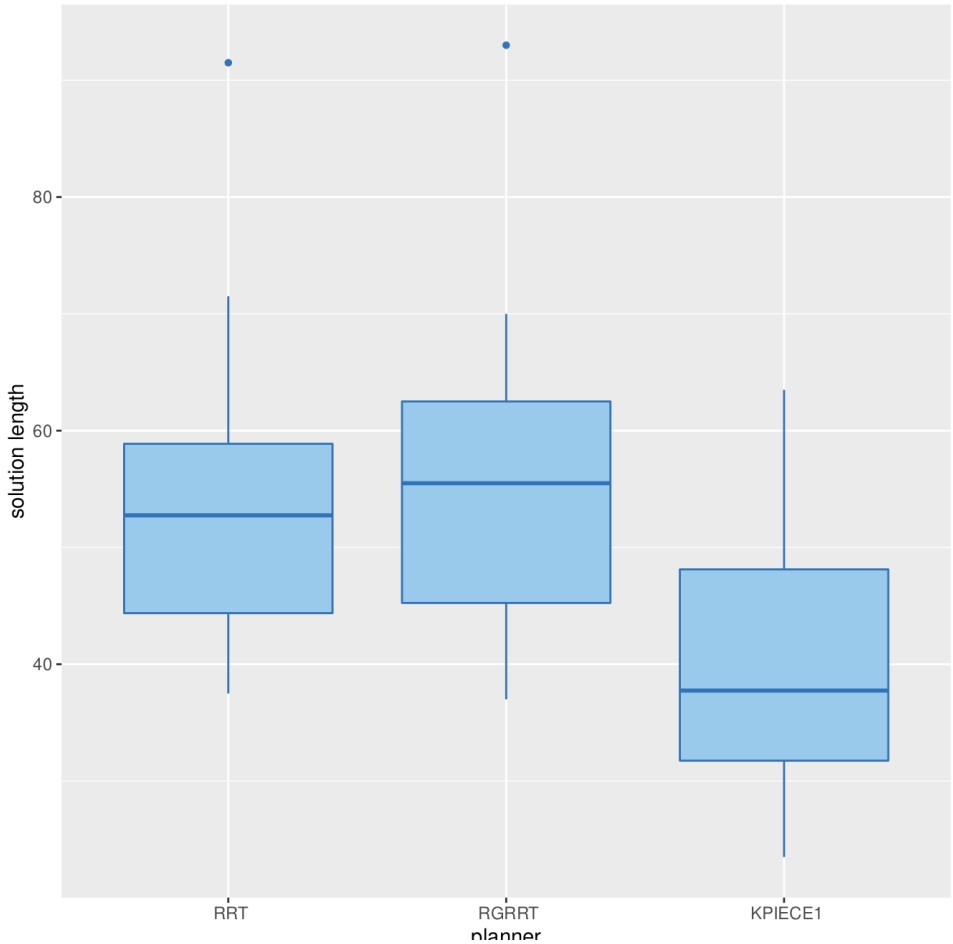

Figure 21: Car-Like Robot Benchmark: path length

- **RRT**:
  - The median solution length for RRT is relatively short, indicating it tends to produce fairly direct paths for the car-like robot.
  - The interquartile range (IQR) is moderate, suggesting reasonably consistent performance with some variation in solution length.
  - A few outliers indicate occasional longer paths, possibly due to difficulty in pathfinding around tight turns or obstacles.
- **RGRRT**:
  - RGRRT has a longer median solution length compared to RRT, suggesting that it might prioritize smoother or safer paths for car-like dynamics over shortest paths.
  - The IQR is large, reflecting variability in solution lengths, possibly due to complex maneuvers required in the environment.
  - An outlier shows the planner occasionally generates significantly longer paths, which may occur when navigating around obstacles.
- **KPIECE1**:
  - KPIECE1 has the shortest median solution length, indicating it generally finds the most direct paths while accommodating the constraints of car-like motion.
  - The IQR is relatively small, suggesting consistent path lengths across different runs.
  - No significant outliers, highlighting its stability in performance for car-like robots in this benchmark.

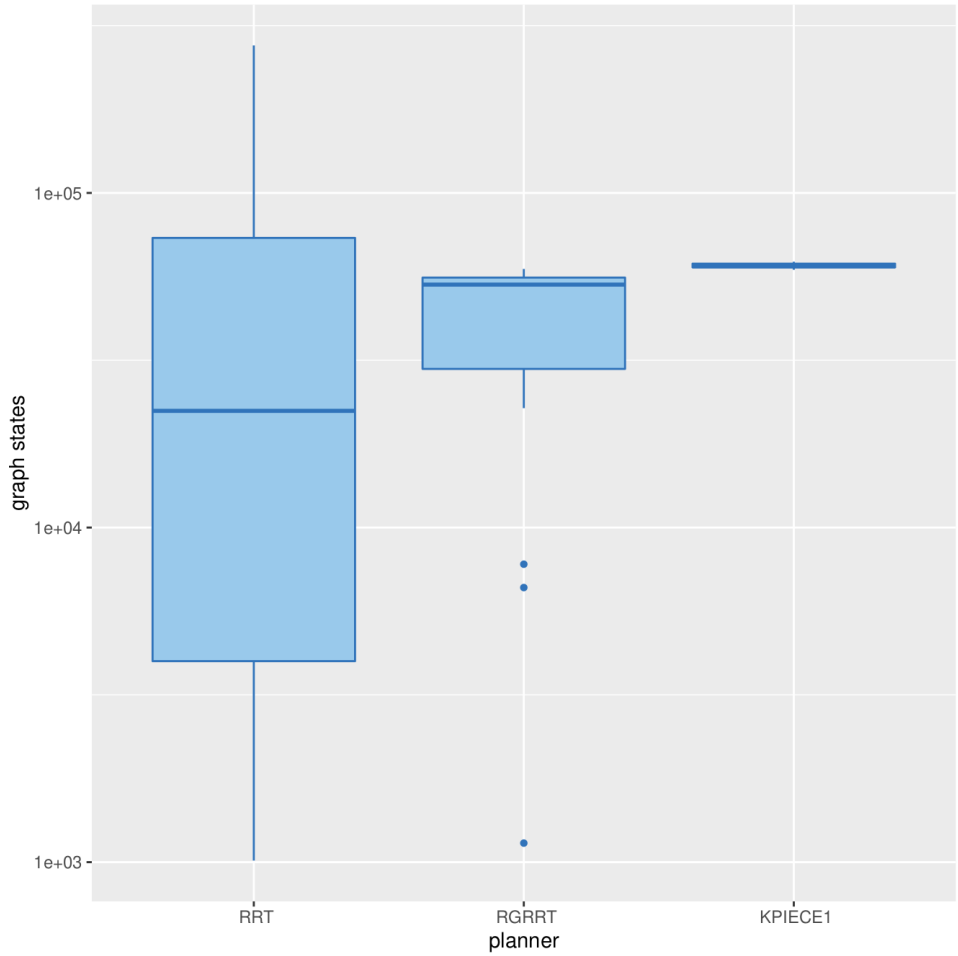

Figure 22: Car Benchmark: number of tree nodes

- **RRT**:
  - RRT has the smallest median number of graph states, indicating it achieves solutions with the least effort among the planners.
  - The interquartile range (IQR) is wide, reflecting significant variability in the number of states across different runs.
  - There are a few extreme outliers, suggesting that in some cases, RRT encounters situations where it needs a significantly higher number of states, possibly due to complex obstacle configurations.
- **RGRRT**:
  - RGRRT has a relatively moderate median graph state count, indicating it requires few more states than RRT to find solutions.
  - The IQR is narrower than RRT, showing more consistency in the required graph states across runs.
  - A couple of lower outliers suggest that in some situations, RGRRT can complete the task with very few graph states, possibly when simpler paths are available.
- **KPIECE1**:
  - KPIECE1 has a large median number of graph states, indicating it requires a substantial number of states to find solutions.
  - The IQR is very small, reflecting highly consistent performance in terms of graph state requirements.
  - No outliers are present, showing stable performance across different test cases.

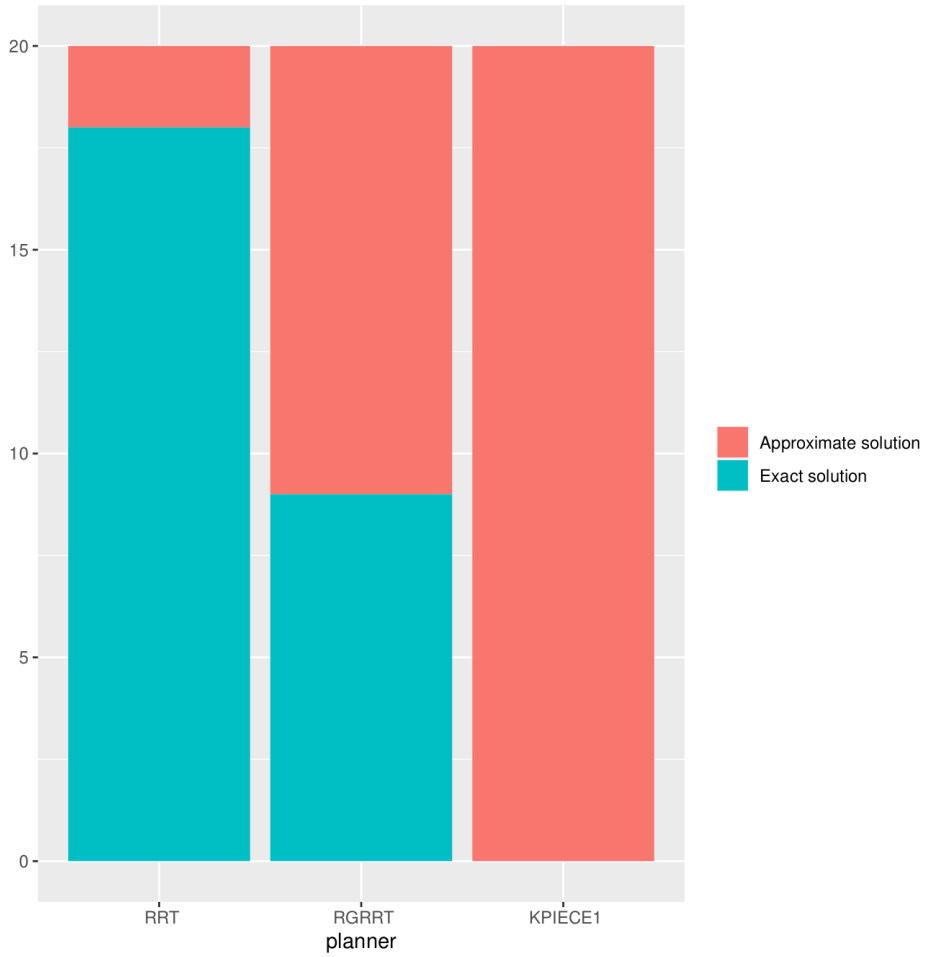

Figure 23: Car Benchmark: success rate

- **RRT**:
  - RRT has a best distribution between exact and approximate solutions, with 90% exact solutions and 10% approximate.
  - This indicates that RRT has a high success rate in finding exact solutions, suggesting reliability in achieving precise paths.
- **RGRRT**:
  - RGRRT shows a mix of exact and approximate solutions, but with a higher proportion of approximate solutions (55%) compared to RRT.
  - This suggests that RGRRT may prioritize faster solutions that are approximate, possibly sacrificing exactness in favor of efficiency.
- **KPIECE1**:
  - KPIECE1 primarily finds approximate solutions, with no exact solutions.
  - This implies that KPIECE1 favors approximate solutions over exact ones, likely focusing on rapid pathfinding rather than precision.

RGRRT Trade-Off comparison:

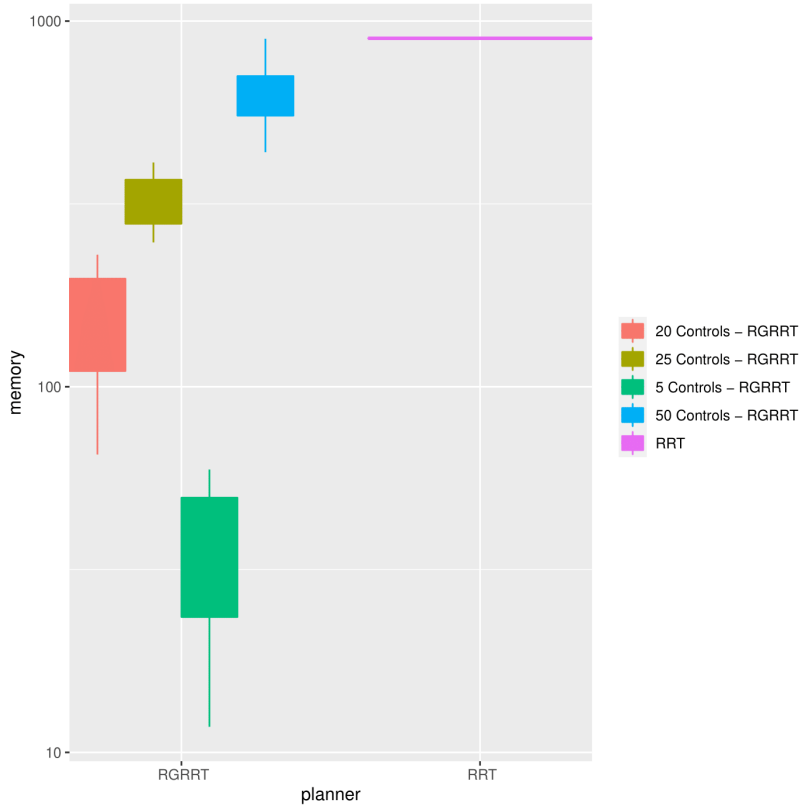

Figure 24: RGRRT Trade-Off Benchmark: memory

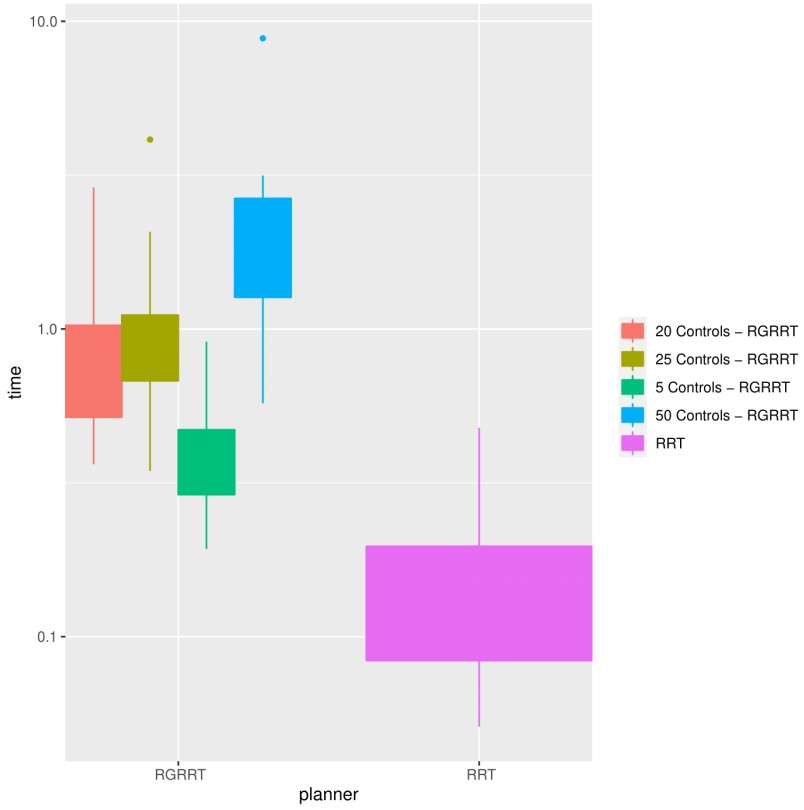

Figure 25: RGRRT Trade-Off Benchmark: time

As the number of control inputs grows, the size of the reachable set $R(q)$ in the RG-RRT planner also increases. For example, when various number of controls are considered such as 5, 15, 25, and 50—the planner's computational demands rise significantly as the control count reaches 50. This increase in controls leads to a corresponding increase in computational time and memory usage. More control options expand the planner's exploration of possible states, but this greater reachability set $R(q)$ requires additional memory to store the expanded states. Consequently, this larger state set not only slows down the computation but also places a heavier load on the system's memory resources.
But, even if number of controls are 50 still the memory occupied by RGRRT is less than that of RRT. On the other hand, RRT is faster and RGRRT even with number of controls as 5.