From: Eli Case
Date: September, 1, 2023
Subject: MECH 450 – Project 1

---

1. To install `OMPL` and `OMPL.app`, I setup the `docker` environment. Once I was able to configure my machine correctly for `docker`, the installation process was straight-foward and took about 20 minutes.

2. Based on these timings, we can compare the demos for car-like systems. The `DynamicCarPlanning` demo appears to have been the most difficult demo overall, since it required about 5 times longer to find a solution than the nearest algorithm, `RigidBodyPlanningWithControls`. Additionally, `DynamicCarPlanning` required about 10 more states than the nearest algorithm, `RigidBodyPlanningWithControls`. Notably, both the car-like planning algorithms with controls and no obstacles took much longer time and more states to find a solution than the car-like planning algorithm without controls and no obstacles, suggesting that adding controls to a system drastically increases the motion planning complexity.

The `RigidBodyPlanning` demo took the least amount of time, along with `GeometricCarPlanning` having the second shortest time. These algorithms also required the least amount of created states. This finding is intuitive, as these planning demos occurred in spaces without obstacles.

From comparing the demos for rigid-body planning, `SE2RigidBodyPlanning` had the longest time out of the rigid body demos and required the most states. This result is expected, since this demo occurred in 3D space with obstacles, as compared to 2D space with obstacles and 3D space without obstacles for the other rigid body demos.

Overall, the most difficult problem out of the demos seems to be planning for systems with controls. Furthermore, planning in 3D space is more expensive than planning in a 2D space with a comparable environment, as expected.

Performance Evaluation of the Six Demo Programs		
Demo Program	Average Time to Find Solution [s]	Number of Created States
`RigidBodyPlanning`	0.001105	102
`SE2RigidBodyPlanning`	0.17146	879
`SE3RigidBodyPlanning`	1.391743	29095
`GeometricCarPlanning --easyplan`	0.019515	4373
`RigidBodyPlanningWithControls`	2.23453	173456
`DynamicCarPlanning`	40.084117	1303070

Table 1. Demo Program Performance

3. Below in Table 2, the solution time and created states are summarized for each problem configuration. Note that each problem used the `KPIECE1` planner to easily compare results. Based on the number of states formed and the time to find a solution, `3D/TwistyCool` appears to be the most difficult problem. It should be noted that `3D/TwistyCool` failed using the `KPIECE1` planner.

Next, the problem configuration `3D/Home` had the largest number of created states. This result is intuitive, as the object being moved has a complex shape in an environment with low clearance between obstacles. The time to compute the solution for `3D/Home` was shorter than for `3D/Abstract`, despite the fact that `3D/Abstract` required less states to solve. This result is attributed to the fact that `Abstract` has a complex environment than `Home` with more complex obstacles, but a less complex object to move. Lastly, the 2D problem configurations required less states and time to solve, which is intuitive since there is less free space to plan in when there are less spatial dimensions.

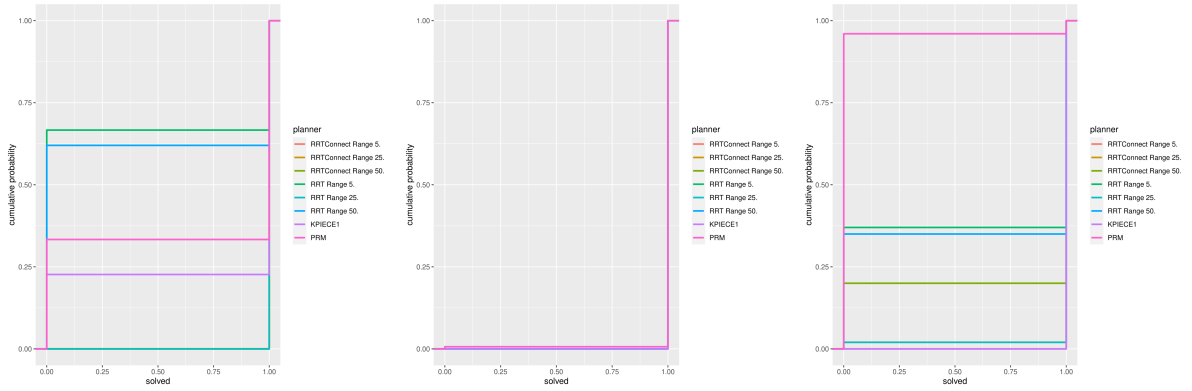
Performance Evaluation of the Five Problem Configurations			
Problem Configuration	Time to Find Solution [s]	Number of Created States	Planner
`2D/BugTrap_planar.cfg`	0.017708	453	`KPIECE1`
`2D/UniqueSolutionMaze.cfg`	0.714159	2497	`KPIECE1`
`3D/Abstract.cfg`	7.532828	26579	`KPIECE1`
`3D/TwistyCool.cfg`	10.090663	70240	`KPIECE1`
`3D/Home.cfg`	4.921847	42322	`KPIECE1`

Table 2. Demo Program Timings

4. To evaluate planner performance, the success of the planner across the 5 problem configurations is considered. The `PRM` planner was not able to exactly solve `3D/TwistyCool.cfg` or `3D/Abstract.cfg`. `RRT-Connect` also failed on these problems. `KPIECE1` and `RRT` were able to solve 4 of the 5 problem configurations, being the most verstile. Generally, the planners with ranom sampling could have better performance if the sampling happened to be better, while `KPIECE1` had more consistent but sometimes worse performance.

5. From increasing the goal bias, it appears a higher goal bias places more weight on achieving a path that is close to the straight goal distance. Too much goal bias seems to degrade the obstacle navigation performance. Increasing the range parameter seems to be analogous to increasing the sensor range. In the `3D/TwistyCool.cfg` problem for example, increasing the range allows some planners to be able to find a solution, which is intuitive, since increasing the range allows the object to more easily see the other room between the wall gap. Lastly, increasing the max number of neighbors decreases the number of sampling states, as each state can be connected to more states, reducing the need to sample a large number of states.

6. First, the success rate for the various planners on each problem need to be discussed. Figure 1 gives the cdf statistics for each problem configuration. `Home` and `Abstract` did not achieve a success rate over 90 % for some of the planners, despite increasing the time limit to 10 minutes for all problem configurations. As a result, the issue with the planners solving these problem configurations is assumed to be an issue with my computer architecture. Likely, a limitation imposed by the processor or memory making the benchmarking difficult to complete within a reasonable time constraint.



(a) Home (b) TwistyCool (c) Abstract

Figure 1. Solved Statistics for Each Problem Configuration

Since the `TwistyCool` problem configuration had the best data, with all planners having a success rate of 90 % or higher to obtain an exact solution for each planner, we can compare the planners for this problem.

First, we can examine the solution length for each planner. These results are summarized below in Figure 2. For the planners except `KPIECE1`, similar performance for the solution path length is obtained. The `RRT` planner with a range of 5 was able to obtain the shortest solution path. `KPIECE1` planner had the longest average path length, but had a much smaller distribution compared to the other planners. Since `KPIECE1` does not do statistical sampling like the other 3 planners, a smaller distribution of solution paths is expected.

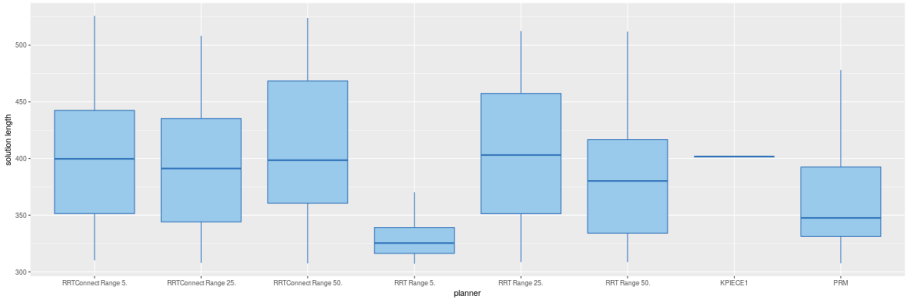


Figure 2. TwistyCool Solution Length

Next, the time to compute a solution for each planner can be compared. The benchmark results are summarized below in Figure 3. From examining Figure 3, the `RRT` and `RRT-Connect` planners at ranges 5 and 15 used the lowest time to compute a solution on average. The `KPIECE1` planner had the highest upper quartile and average time to compute a solution. Additionally, the `PRM` planner had performance slightly better than `KPIECE1`.

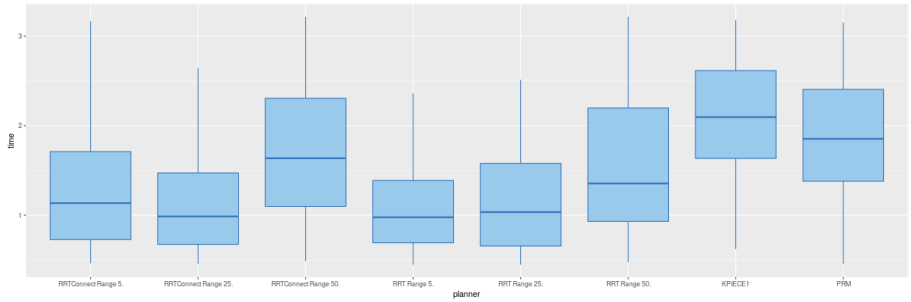


Figure 3. TwistyCool Time to Compute Solution

Lastly, the number of graph states is compared for the TwistyCool problem. The benchmark results are given in Figure 4. The `RRT` and `RRT-Connect` planners at a range of 10 and `PRM` had the best performance with the fewest average graph states, while `KPIECE1` had the worst performance with most average graph states.

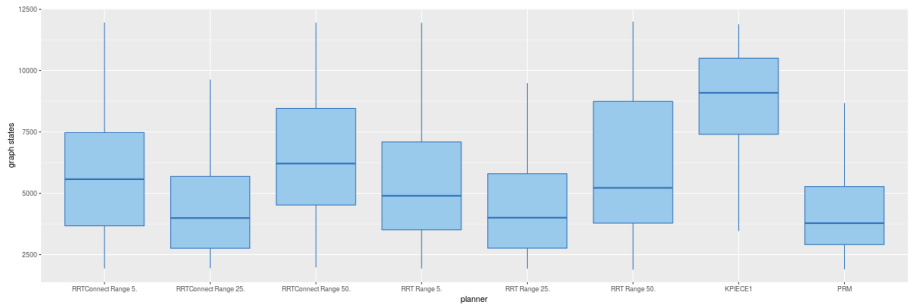


Figure 4. TwistyCool Number of Graph States

Overall, for the TwistyCool problem configuration, the `KPIECE1` planner had the worst performance across the three metrics discussed here: solution path length, time to compute solution, and number of graph states. This result is likely because `KPIECE1` offers a tree-based planner with reliable performance, but it cannot take advantage of variations in how the free workspace is sampled to achieve high performance in some scenarios. The `RRT` and `RRT-Connect` planners at a range of 10 had consistently good performance across the three metrics. Both the average and lower quartile performance outpaced the other planner for time to compute solution and number of graph states. This performance result with a range value of 10 suggest there is an optimal range value that needs to be tuned when using these planners for a problem.

7. Problems 1,2,3,4, and 5 had a difficulty of 2. Running the OMPL demos and problems with the GUI was straightforward. Problem 6 had a difficulty of around 8. This difficulty is attributed to the fact that I could not get good benchmark results for complicated problems due to limitation of my hardware, not matter how long I made the runtime.