**Lab Assignment 2:**

1. **Discuss how the performance (cost and time) of the planner is affected by the hyper-parameters .**

We received the following figures of the performance for the different goal bias values when running our algorithm for 2000 iterations:

For

A graph with colored dots and numbers

Description automatically generated

And for :

A graph with numbers and dots

Description automatically generated

As we can see in the figures above, the average computation time for is generally lower than the computation time for . Moreover, the cost of the solution that was found is also generally lower (not always) for runs with , this makes sense because for very low bias values we expect a slower convergence to the solution. In addition, there weren’t many obstacles in the running environment in the path to the goal configuration hence whenever we take a step in the goal’s direction it’s more likely to be part of the shortest path to the goal, shortening both the path cost and computation time.

As for the step size, we can’t really distinguish a correlation between the step size and the performance. In both figures we noticed that was the value with either the best path or very close to it, meaning this value of step size is compatible with our environment. Other than that, we can’t really make a clear distinction between the quality of the paths based on the step size.

1. **from the paths you have found, Choose the path with the lowest cost and execute it on the UR5e manipulator. include a video that visualizes it.**

To open the video, double click on the icon below:



But since it caused us a few problems on our device, we’ve included the video in the submission zip file to be safe. (with the name Q4-Video.mp4)

1. **Compare the path with the shortest time you found to the path found by OMPL.**

The path we found compared with the one generated by OMPL took significantly longer to compute. This discrepancy is likely attributed to several factors. Firstly, our implementation was developed in Python, which is inherently slower than languages like C/C++ commonly used in libraries such as OMPL. Additionally, OMPL benefits from optimizations and algorithmic enhancements specifically tailored for efficient motion planning and collision detection, which might not have been fully incorporated into our custom implementation. These optimizations can significantly expedite the search for feasible paths in complex environments. Therefore, the observed disparity in computation time underscores the importance of leveraging specialized libraries like OMPL for expedited and optimized motion planning tasks. It's worth noting that despite the difference in computation time, we didn't observe any major discrepancies in terms of the path's length. This consistency suggests that, at their cores, both the OMPL algorithm and the one we implemented achieve similar path lengths.