ME 570: Robot Motion Planning

Homework 4 Report

By Cameron Cipriano

12/02/2021

Problem 1: Graph Search

Question 1.1 code: Graph.heuristic

Implemented using numpy's linalg.norm() and subtracting the two points' physical locations in the graph

Question 1.2 code: Graph.get_expand_list

Implemented using the graph's neighbor list

Question 1.3 code: Graph.expand_element

Implemented by following the logic of lines 9-16 of the A* pseudocode

Question 1.4 code: Graph.path

Implemented by basically treating the nodes as a linked list and iterating over the backpointers until I got to the beginning

Question 1.5 code: Graph.search

Implemented by following the logic of the A* pseudocode

Problem 2: Application of A* to the Sphere World

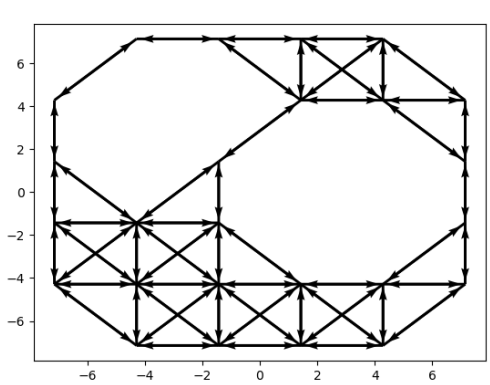
Question 2.1 code: SphereWorldGraph.__init__

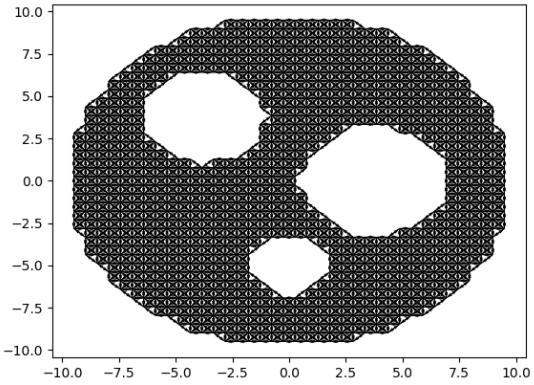
Implemented by using my lambda functions from the previous homework and modifying it to translate values into 0s and 1s before the grid2graph function would vectorize that into booleans

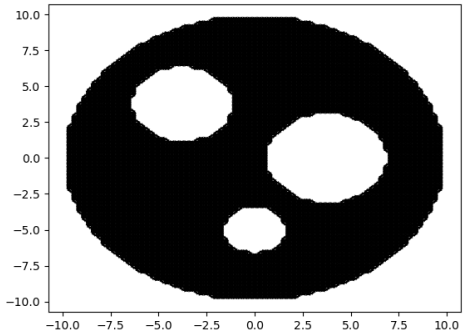
Question 2.2 code: Graph.search_start_goal

Implemented by following the description's indication to use the nearest_neighbors function with number of neighors as 1 and calling search.

Question 2.1 report : nb_cells discretization

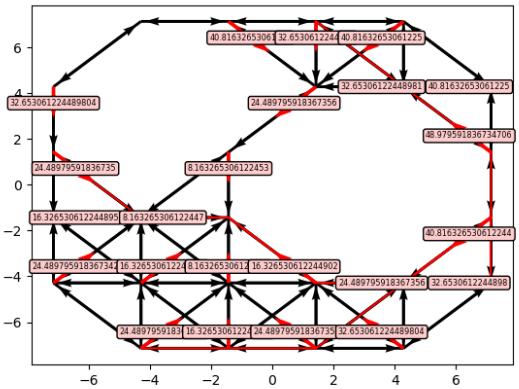


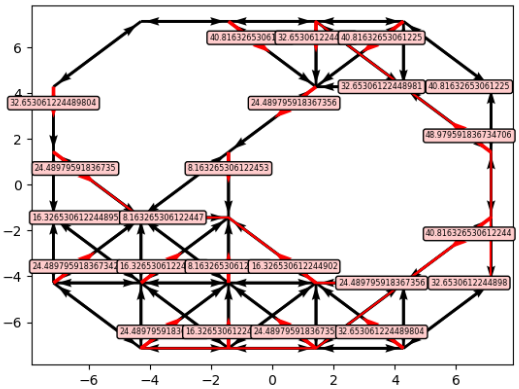




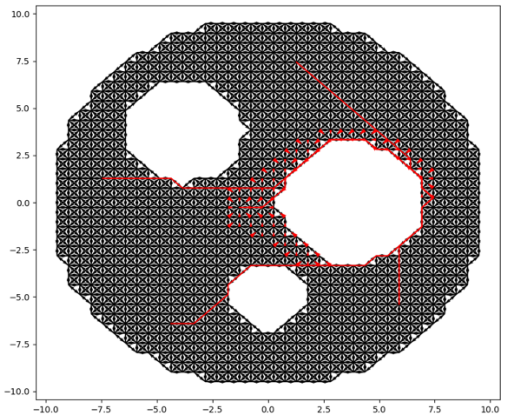
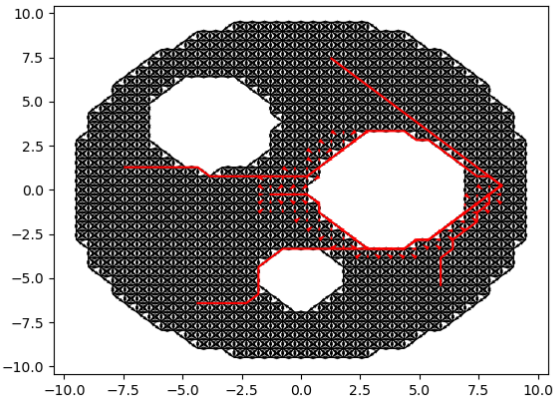
Top Left: Discretized with 8 Nodes
Top Right: Discretized with 40 Nodes
Bottom Left: Discretized with 80 Nodes

Question 2.2 report : SphereWorldGraph run_plot method

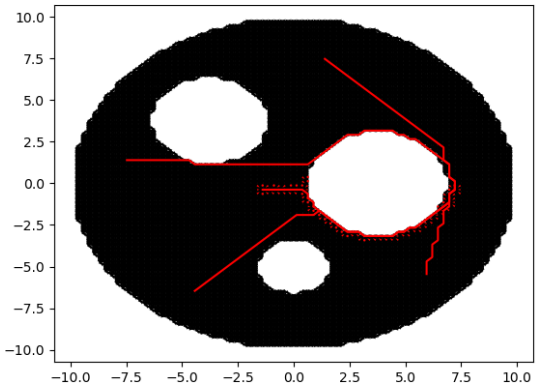
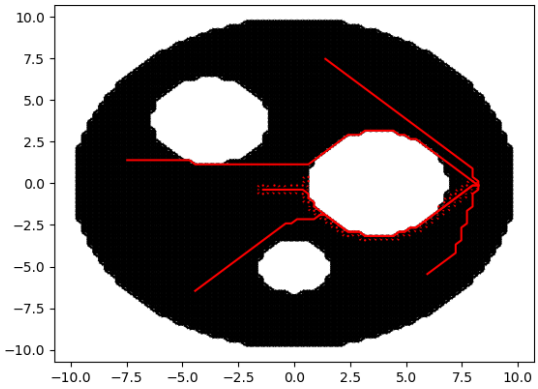




Left: Discretized with 8 nodes goal location 1
Right: Discretized with 8 nodes goal location 2

Left: Discretized with 40 nodes goal location 1
Right: Discretized with 40 nodes goal location 2

Left: Discretized with 80 nodes goal location 1
Right: Discretized with 80 nodes goal location 2

Question 2.3 report: A* Behavior given choice of nb_cells

Given the choice of nb_cells, A* has a few properties. The first would be the speed of execution, as the number of cells remains small, A* will complete very quickly. If the number of cells is very large, then it will take much longer.

Another property based on the choice of nb_cells is how close the path is to the true path that a robot would take to get to a goal. The higher nb_cells, the higher the resolution the environment is discretized into, thus allowing for finer control of where the robot will go in the environment.

Something else that is noticeable is the relative size of the explored nodes for certain sizes of nb_cells. When nb_cells is still below a reasonable size, the A* planner will tend to explore more nodes as the nodes have relatively similar distances to the goal. When nb_cells is larger, this

won't be as apparent since each node will be slightly different from the others, allowing for a more direct path to the goal.

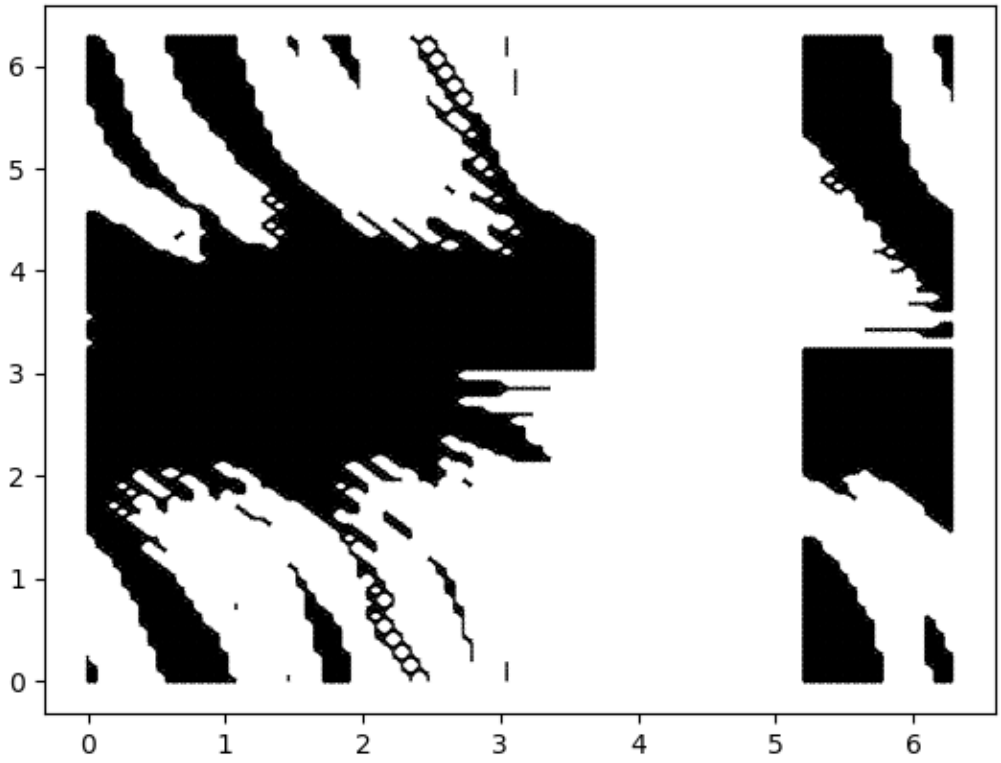
Question 2.4 report: A* Behavior with respect to the potential planner

The potential planner from homework 3 has no effect on the behavior of A*. This is because the graph simply takes the presence of some potential to be true and the absence of potential to be false, and thus an obstacle. As long as the potential function chosen spans the entire environment (which is the case for attractive potentials defined as a distance), then the only metric that matters is the resolution of the graph.

Problem 3: Application of A* to the Two-Link Manipulator

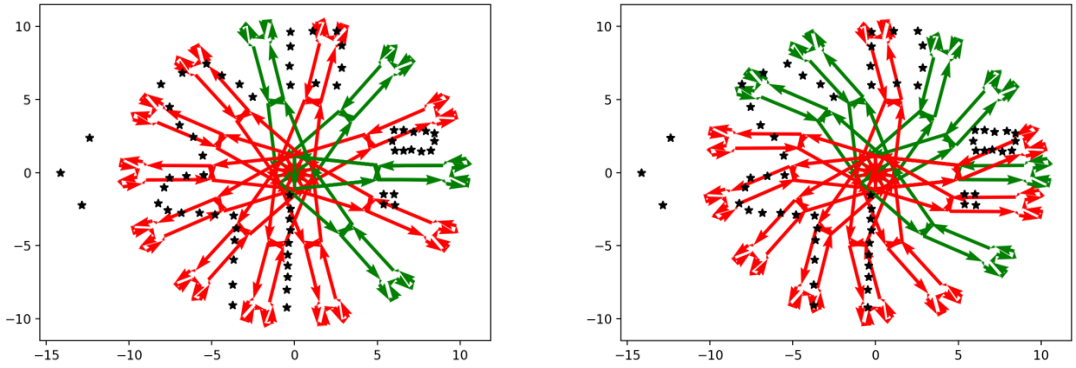
Question 3.1(a) report: TwoLinkGraph.load_free_space_graph

Question 3.1(b) report: TwoLinkGraph.plot

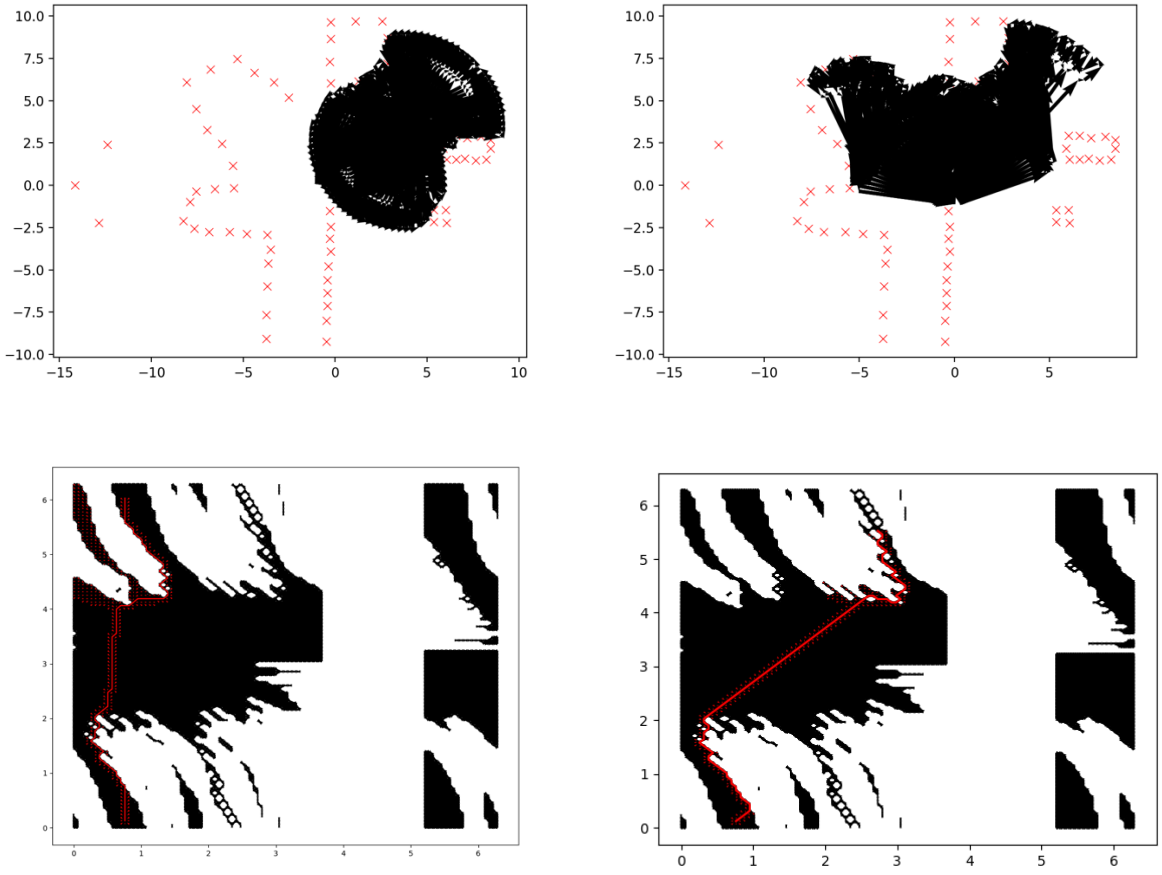


Question 3.1(c) report: TwoLinkGraph.search_start_goal

Question 3.2 report: Plot the points obstacle_points



Left: obstacle_points with straight end-effector positions
Right: obstacle_points with end effector rotated by π/7



Left Side: Easy theta_start to theta_goal configuration and final path
Right Side: medium theta_start to theta_goal configuration and final path

Question 3.3 report: Comment on the *unwinding* phenomenon in the easy case

For the easy case, if we take a look at the path in the bottom left image, it is simple to see that despite the final and ending theta for the first link being identical, there is no straight black line connecting the two final theta's. Because of this, the robot thinks that its range of motion is restricted in particular areas and must maneuver itself in a complicated way to avoid the "obstacles" that it thinks are present. Because we are not using the torus option, it doesn't know that the path can wrap around to the other side, allowing the first theta to stay fixed and find the true optimal path. This unknown knowledge causes the sort of "back and forth motion" when traveling up the entire graph, unwinding itself.

Question 3.4 report: Comment on the obstacle closeness issue and practical solutions

Given the fact that A* is optimal with an optimistic heuristic, it is always going to try and hug the obstacles because that will be the shortest path between any start and the goal. In order to potentially avoid the closeness to obstacles, we can arbitrarily expand the borders of all obstacles. This expansion of the obstacle's borders will allow the A* algorithm to still hug the "obstacle", finding the true optimal path in the graph, while now avoiding the true obstacle's dimensions very closely.