COMP3702/COMP7702 Artificial Intelligence (Semester 2, 2020) Assignment 2: Continuous motion planning in CANADARM

Name: Joel Thomas

Student ID: 44793203

Student email: s4479320@student.uq.edu.au

Note: Please edit the name, student ID number and student email to reflect your identity and **do not modify the design or the layout in the assignment template**, including changing the paging.

Question 1 (Complete your full answer to Question 1 on the remainder page 1)

The C-space or configuration space in CANADARM2 is defined as the set of all possible mechanical arm configurations, where a configuration is the parameters that uniquely define the position of one of the grappled end effectors of the arm, and the subsequent angles at and lengths between every joint of the mechanical arm leading to other end effector. Mathematically, a configuration q is represented as:

$$q = \Big\{x_i, y_i, \theta_1, \theta_2, \dots, \theta_n, r_1, r_2, \dots, r_n : (x_i, y_i) \in [0,1] \times [0,1], i = 1,2, \\ \theta_j \in [-\pi, \pi], r_j \in \Big\{\min_j(armLengths), \max_j(armLengths)\Big\}, j = 1,2, \dots, n\Big\}$$

Above, (x_i, y_i) represent the x-axis and y-axis cartesian position of the i^{th} grappled end effector (always either 1 or 2). θ_j represents the angle of the j^{th} arm joint ordered relative to the i^{th} grappled end effector and is represented in radians. r_j represents the arm length of the j^{th} link (line segment) of the mechanical arm ordered relative to end effector 1. n is the total number of joints/link segments. A configuration q is collision-free only if q satisfies the environment boundaries, arm angle, length and grapple point constraints and does not collide with itself or any obstacles within the environment.

The Probabilistic Roadmap (PRM) has been used here for searching in continuous space. In this scenario, it consists of four key components – the sampling strategy, path interpolation, collision checking and the connection strategy. The basic algorithm structure of PRM is that until the goal configuration has been reached from the starting configuration, we sample several new configurations, create interpolated paths between these new configurations and existing nodes' configurations and add them to the search tree provided no path collisions occur. After conducting search, we repeat these steps until a solution is found.

Question 2 (Complete your full answer to Question 2 on pages 2 and 3, and keep page 3 blank if you do not need it)

Given the description of a configuration as in question 1 above, here are the following steps used in implementing PRM for searching in CANADARM2 continuous space environment:

1) Randomly generate 10 samples (configurations) uniformly and immediately throw away samples which do not satisfy the (individual) configuration collision checks mentioned above. The value 10 was chosen based on trial and error (neither too small nor too large). The individual collision check verifies satisfies the environment boundaries, arm angle, length and grapple point constraints and does not collide with itself or any obstacles within the environment. These are copied from $tester.py$.

2) For each valid sample generated (retained) and for every existing robot configuration (as a node) in the search graph, we first perform a distance check that calculates the minimum number of primitive steps required to move between two configurations:

$$distance = \frac{\sum_{j=1}^{n} |\theta_{1,j} - \theta_{2,j}|}{0.001}$$

$$passDistanceCheck = \begin{cases} True\ if\ distance \leq 3000 \\ False\ otherwise \end{cases}$$

This distance check is based on the idea that an interpolated path between the two configurations that is longer than 3000 primitive steps is likely to face a collision at some point. Note that this distance formula is based solely on the difference between each of the n angles in the robot configurations rather than their lengths. In addition, if both configurations are grappled to end effector 1 (EE1) then the angles used are relative to the position of EE1 ($ee1Angles$) and vice-versa ($ee2Angles$). The value 3000 was chosen based on trial and error (neither too small nor too large).

3) A proper path collision check is necessary to verify whether a path between two robot configurations collides or not so that in the event the collision check returns false, it is possible to add the configuration as a node to the search graph. Generating such a path is called interpolation and essentially entails taking two robot configurations and outputting several intermediate robot configurations that satisfy the "1 primitive step apart between any two configurations" constraint.
To perform interpolation, while the last interpolated configuration in list of configurations is not equal to the required configuration, we modify each of the angles and lengths 0.001 units closer to the required configuration. If any angle/length is within 0.001 units of the desired configuration, we instead alter using provided specification tolerance ($\frac{spec.tolerance}{3} = \frac{1e-5}{3} = \frac{1}{3}e-6$) till equality is achieved and finally output this list of configurations.

4) Given an interpolated list of configurations between an existing configuration and a valid sample configuration, we perform a path collision check by testing for individual configuration collision on each of the interpolated configurations in the list.

5) Provided the path collision check is satisfied i.e. returns false, the valid sample configuration can be added as a new node to the search graph and the sample configuration and the existing configuration can be made neighbours. The sample configuration's cost is the sum

of the Manhattan distances between each of the joint positions of the sample configuration and the end configuration.

6) After adding all valid retained sample configurations to the search graph, we perform search using the A* search algorithm. A* search works here since we are only adding valid sampled configurations that have at least one valid interpolated path to another existing configuration – we're able to then add them as each other's neighbour. Eventually, there will be enough nodes such that traversal from one neighbour to another will yield a valid path from the start configuration to the goal configuration. In addition, using Manhattan distance as a heuristic should help guide the search to find the solution faster.

7) We repeat steps 1)-6) indefinitely until a valid solution is found. This also applies to finding bridge paths between grapple points which can be thought of as intermediate solutions – the final solution is all the intermediate solutions concatenated together.
Wherever a solution is possible (between start and goal configurations) in a given setup, a solution is guaranteed to be found because since we are sampling each configuration uniformly, theoretically, any point in the C-space can be generated with probability $\epsilon > 0$. Hence, we have:

$$\lim_{\#samples \to \infty} [\mathbb{P}(sample\ a\ feasible\ path)] = 1$$

Thus, after generating several nodes in the graph, the likelihood of a valid traversal from the start to goal nodes via neighbours increases in magnitude. However, this can take a very long time depending on the scenario.

Note that in step 7), the statement that a solution is guaranteed to be found where possible assumes that all components are working together perfectly. However, this was not the case for my program. See the response made about multiple grapple points in question 3 below for further explanation on the pitfalls of the program.

Question 3 (Complete your full answer to Question 3 on page 4)

Given the provided testcases, it is easy to identify which scenarios the program implementing PRM as above performs well in and struggles in.

Since there is only one sampling strategy used for finding solution paths (not $generate\_bridge\_sample$ which is used for calculating an end node configuration for bridging between two grapple points), the program tends to perform well when there is a lot of freedom i.e. free space available in the environment and the goal configuration is not located too far from the start configuration. In particular, the program struggles in scenarios where the presence of obstacles requires a very tight and constrained path for the arm to move through. Since uniform random sampling is used in PRM, as explained in question 2 above, every configuration in the C-space has equal probability of being generated which in this case directly translates to a lot of thrown away samples due to the heavy motion constraints. While a solution is guaranteed to be found, it can take a very long time since the program isn't able to achieve greater persistence in these difficult areas.

In addition, the program can handle tests with two grapple points fairly well, however it is never able to find a solution for a problem with more than two grapple points. This is explained by the brute-force approach used to calculate the final angle between the last joint and the next grapple point in $generate\_bridge\_sample$. Essentially, it was too difficult to find a common algorithm for finding this last angle – there was always some dependency on the net angle that could not be removed leading to trying different angle combinations which could potentially lead to the true final angle. For example, in some test cases, where the final angle could be calculated using $\pi - \theta - net\_angle$ where $\theta = \arctan\left(\frac{\Delta y}{\Delta x}\right)$, it would not work for other test cases which would instead require $net\_angle - \theta$, or some other variation of this. Thus, this led to a brute-force approach of trying several different variations because it was too difficult to find the single formula which would work for all cases (see $angle\_fixes$ list in $generate\_bridge\_sample$). This also meant that since not all cases were accounted for, there were test cases where the program would run indefinitely because it would get caught in an infinite loop – under the while loop on line 524 in the program, $generate\_bridge\_sample$ would always return None because the last point of the sample configuration could never equal the next grapple point.

Finally, the program appears to not be affected by the number of links/segments used in the problem description, so it can handle these types of problems well.

In conclusion, the program should be able to easily solve test cases that:

- Do contain a lot of free space
- Do not contain several obstacles or tight constrained pathways leading to the goal
- Do not contain multiple grapple points
- Do contain several links/segments