University of Queensland

COMP7702 Assigment 2 - Report

Canadarm - Continuous Motion Planning

Author:
Duy Long Tran

Student Number:
44992305

September 27, 2019

1 Define the Configuration Space and Method of Continuous Searching

The Canadarm motion planning problem can be defined in the following C-space:

$$C = \Bigg\{ (\mathbf{g},\ \mathbf{l},\ \theta) \mid \mathbf{g} \in \mathbb{G},\ \ l_{min} \le l_i \le l_{max}\ \ \forall i \in \{1, 2, ..., n\}, \\ -180 \le \theta_1 \le 180,\ \ 165 \le \theta_i \le 165\ \ \forall i \in \{2, ..., n\} \Bigg\}$$

where $\mathbf{g} = (x_{ee}, y_{ee})$ is the coordinates of the grapple point
$\mathbb{G}$ is the set of all grapple points
$\mathbf{l}$ is the vector of the length of n sections from the robot's arm
θ is the vector of the angles of the n sections

The additional conditions of the C-space is that all the angle coordinates of the robot's arm should not lie outside the boundary [0, 1] or inside the obstacle boxes. We can constraint on these two conditions by define the forbidden region in the C-space as:

$$C_f = \Bigg\{ (x,\ y) \mid (x,\ y) \in \Bigg((\mathbb{R} \setminus [0,\ 1]) \ \cup \ \bigcup_{i \in O} D_i \Bigg) \Bigg\}$$

where $(x,\ y)$ is the coordinates of the angle points
O is the set of all obstacles
D_i is the region of the obstacle $i \in O$

To find the path from the initial configuration (I) to the goal configuration (G), I used the Probabilistic Road Map (PRM) to build a graph of valid configurations and then use Bread First Search (BFS) to find a possible path from I to G. The random configuration for PRM is generated by 3 following parameters:

1. The specified grapple point and the grappled end of the arm (`ee1` or `ee2`).
2. The minimum lengths of all segments. The reason I use minimum length is that the smaller the robot is, the more possibility it can transverse among the obstacles.
3. The randomly generated angles of all segments within the boundary stated in the C-space above.

2 Random Sampling Strategy

2.1 Overall Sampling Strategy

The overall strategy includes 3 main steps:

1. *Generate a possible order that the grapple points can change from one to the other.*
   This order should be consistent with the grappled ends of the arm. For example, if there are 4 grapple points in the space, and the `ee1` end of the arm is initially grappled to grapple point 1. Suppose we need to find a path such that in the final position, `ee2` end is grappled to grapple point 4. We know that the first move will make the arm from `ee1` grappled to `ee2` grappled, and the last move will also make the arm from `ee1` grappled to `ee2` grappled. Thus, one possible order can be one move from I directly to G, and the other orders are feasible if they add an even number of grapple points between I and G. For 4 points problem, we can list out the orders as follows {1, 4}, {1, 2, 3, 4}, {1, 3, 2, 4}.

2. *Generate configurations that connect 2 consecutive grapple points in the order from step 1.*
   The goal of this step is to divide the big problem of finding a path between grapple points into several sub-problems of finding a path at the same grapple point. The technique to generate the connecting configurations is explained in the section 2.2.2.

3. *Find the paths for the sub-problems resulted from step 2.*
   The main idea in finding the paths at this step is to generate a (small) number of random configurations which connected to either (sub)-initial or (sub)-goal configurations. A connected configuration is defined as connecting to the initial/goal configuration or connecting to the neighbors of initial/goal configuration. The generation stops when it finds a configuration that connects to both initial and goal configuration. Some further techniques are summarized in section 2.2.3.

2.2 Paths Verification Techniques

2.2.1 Paths and collision verification

Checking the collision between the configuration and the environment's boundary is trivial, by verifying all angle points falling in $[0, 1] \times [0, 1]$ region. To check the collision with the obstacles, I use the technique of linear function valuation.

Suppose we have a function for line AB: $ax + by = 0$, then the two points C, D are on two different sides of AB if and only if $(ax_C + by_C)(ax_D + by_D) < 0$. And the two closed lines AB and CD intersect if and only if:

(C, D are on different sides of AB) AND (A, B are on different sides of CD)

Hence, we must have the following conditions for all pairs of robot's segments and edges of obstacle boxes:

$$(ax_C + by_C)(ax_D + by_D) < 0$$

$$(cx_A + dy_A)(cx_B + dy_B) < 0$$

Regarding paths verification, a possible path between two configuration is verified and built up by mid-point verification. To be specific, suppose we need to check a pair of configurations (config A, config B). Then, the algorithm will generate the middle configuration (mid config), by taking the average value of the lengths and angles of config A and config B. If the mid config is valid, the process is repeated with two new pairs of configurations (config A, mid config) and (mid config, config B). This is a recursive problem but will quickly reach the depth because the distance between two configurations decreases by $O(2^{-n})$.

2.2.2 Grapple points connection

To find the configurations that connect 2 different grapple points, the algorithm keeps generating random configurations grappled at one of the 2 grapple points, until having a "close" configuration. The "close" one is defined as the distance between the second last point (counting from the grapple point) of the arm and the other grapple point is less than the maximum length of the last segment. In other words, the arm can rotate and stretch/shrink the last segment to reach the other grapple point.
Following that, we need to recalculate the last angle of the arm, which can be done by cross products of the vectors between 3 points: the grapple point, the second and third last points of the arm. Once the new configuration is generated, it comes in a pair, with exactly the same points in the space, but different grapple ends. Hence, to move from one grapple point to the other, we can directly move from one configuration to the other of the pair, without adding any other configurations between them.

2.2.3 Sub-paths finding techniques

Given an initial configuration (I) and a goal configuration (G), we generate a set of n neigbors for I, and other n neigbors for G. By doing that, we can safely ignore any potentially useless random configurations. Our goal is to find a connecting configuration that connects to both I's neighbors and G's neighbors. But limiting to only $2n$ configurations is not a good strategy. We need to expand the graph when the connecting configuration can not be found within some given of iterations. From my testing, this number of limitation should be hardcoded to 20 random generations. There is no reasons to decide the number, because it substantially depends on the problem specification. The number in my code is decided mainly because of my hardware limitation. In more powerful systems, we can set this number smaller, to get the neighbors of I and G increasing quickly, and get more the chance to achieve the connecting configuration.

Another technique in finding the sub-paths is to regenerate the grapple configurations. From the neighbors of I and G, if we keep adding more neighbors to the list, the sizes will blow up the problem. One reason we can not find the connecting configuration is that the grapple paths connection did not generate good grapple configurations. To handle this problem, we can regenerate the grapple paths when the size of neighbors set exceed some values. And once again, I set this limit to 10 neighbors, mainly due to the hardware. One important note here is that we need to regenerate only the (sub)-goal configuration in the sub-problem, because the (sub)-initial configuration is related to the previous sub-paths which are already found.

3 Testing Results and Discussions

The performance assessments are summarized in (Table 1) for some test cases from simple to difficult levels. We can see that the number of configurations generated increases for harder levels, but the number of useful configurations is almost the same between the levels. This is because the algorithm discards a huge number of generated configurations due to collision, or non-connecting to initial/goal.

What I mostly concern about is the rule of generating random configuration with only the minimum lengths of all segments. However, the results show that the rule really helps to speed up the calculations without losing any capacity of the algorithm.

Test Case	Number of configurations				Elapsed Time
	Generated	Colided	Deleted	In use	
5g3_m3	746	612	118	30	109s
5g3_m1	804	635	155	27	69s
4g3_m2	244	188	42	21	23s
4g4_m1	143	95	35	36	61s
3g1_m2	195	144	48	11	25s

Table 1: Number of configurations generated and the elapsed time for different test cases. The Colided and Deleted column are the number of configurations which did not pass the collision verification and non-connecting to the initial/goal's neighbors.

This algorithm is able to solve the problems with simple grapple points changing, since it only allows one grapple point to be attached by one out of the two ends of the arm. For a problem with more complex grapple movements, for example, 2 ends must go around between 3 grapple points, and 1 grapple point might be visited by both ends, this algorithm can not found a feasible grapple path. Another problem is that I do not set the recalculating options for the whole algorithm. Since this is based on PRM method, then in some cases, we do need to recalculate the possible order of grapple movements from step 1 in section 2.1.

However, the advantage of the algorithm is the power in searching on a narrow space by means of the minimum length's rule, and by dividing the big problems into sub-problems. This might be not good for simple problems, because some imperative steps are unnecessary, but would save a huge time for the class of 4-segment-arm or more.