Linfeng Liu 44563376

Yunwen Wu 44816371

Fan Yang 44594213

COMP3702

Assignment2

Group 666

September 27, 2019

1. Please define the Configuration Space of the problem and describe which method you use for searching in continuous space.

The dimension of the C-space is dependent on how many line segments the robot has. For example, if the robot has three segments, then we can define the C-space as below:

C = ,

Where (x,y) is the position of the grappled end effector in the workspace, a1 is angle between the first segment and the grappled end point, a2/a3 are joint angles ordered relative to the end effector, L1/L2/L3 are the lengths of each link which are also ordered relative to the end effector.

So, for 3 segments robot, C-space is 8 dimensional. If the number of segments increases so will the dimension as the number of angles and lengths will increase equally.

Method used for this assignment is Probabilistic Roadmaps (PRM) which is based on discretization and sampling. The first step is to use uniform sampling to generate sample points in C-space. Since end effector is fixed on grapple point, the sampling function will only generate a list of lengths and angles. Before being added into the C-space, they will be made into a robot configuration to determine if they are valid. If the generated sample point is collision free, it will be added into the node list with the distance between sample point and other nodes in list being checked. If the distance is below a certain limit, these two nodes will add each other into the “neighbour”, indicating that they are connected in search tree. Once sample list is created with enough sample, Breadth First Search algorithm is applied to finding the path from initial state to goal state.

To check if the path between two sample points is valid, the path will be discretized. It will be divided into several smaller segments and for each segment the validation check will be applied to make sure that there won’t be invalid configuration lies in the path. The validation check includes self-collision check, obstacle-collision check and boundary check.

Full demonstration on the method applied in the assignment is showed in question 2.

2. Describe the strategy you applied to develop each of its components.

The whole algorithm consists of three main parts: sampling, connecting and searching. First step is to sample a configuration q uniformly at random in space. A test is applied in the process to make sure all configurations added into list are valid. The second step is to check whether q is a neighbour of existing configurations. This step consists of two parts:

1) Check the distance so that the two configurations are not too far away in C-space.

2) Check whether the path between two configurations is valid.

The pseudocode is shown as below:

For I in range (sample amount) {

Q = Uniformly sampling ();

For config in All Configurations {

If distance between q and c is below certain amount and path is valid {

add a neighbour relationship between nodes

}

}

}

After connecting all the edges, use breadth-first search (BFS) algorithm to find the solution.

More detailed description for each part and how the function is implemented in code is shown below.

2.1 Sampling strategy

Angle and length are sampled uniformly random using random.uniform(). We set different amount of sample points to be generated based on the complexity of map. For problems with less than or equal to three arm segments, 1000 sample points are generated and for problems with over three arm segments, 3000 sample points are generated. It can reduce the run time on simple maps and make sure a valid path can be found on more complex maps. Increasing in the size of sample points leads to much more checks, so the amount of sample points should be controlled under a limit, otherwise it will take long time for finding a path.

With respect to uniformly random generate angles and length, the generated angles are between -165° to 165° and generated lengths are between the minimum and maximum length of each arm. With all the angles and length generated, apply test functions to make sure that the configurations are not in collision with obstacles, bound and itself.

2.2 Connection strategy

2.2.1 Add neighbour

The randomly generated sampling point q is required to be connected to other configurations in the nodes list containing all previously generated sample points as well as initial state and goal state. Applying the following two steps to see if q can be added to an existing node’s neighbour.

2.2.1.1 Check distance

The check distance function is applied to check whether the distance between q and other configurations is under requirement. Compute the difference between two same sequence angles and lengths. If any of the differences is larger than the 0.5 rad, these two nodes will not be connected as “neighbour”. If the limit is above 0.5, the path will be more likely to have path problem, and more checks are required to make sure the path is collision free. If the limit is below 0.5, more sample points are required or it may not be able to find the path.

2.2.1.2 Check path

To check validation, the path between two configurations are divided into 20 segments and each segment will have a collision free check. Similar to what has been done in sampling, this includes self-collision check, obstacle check and bound check.

The pseudocode of path check is shown as below:

List x = distance between angles

List y = distance between lengths of segments

For I in range (20) {

For n in range (number of segments) {

New angle[n] = q angle[n] + list x[n]\*0.05I

New length[n] = q length[n] + list y[n]\*0.05I

}

Using new angles and lengths to construct new configuration W

If W is invalid: return False

}

return True

2.3 Multiple grapple points strategy

If the map has more than one grapple points, the algorithm needs to find the bridge between two grapple points so that the arm can walk along the way. To extend the model of single grapple point problem, we separate the whole search problem into several steps, each with its own initial state and goal state. A bridge state is obtained and used as sub-goal because it is both ee1-grappled and ee2-grappled.

Since there are infinitely many possible bridge states between two points, we made sub-goal a list so that the program has more chance to find the path. In the assignment, 10 sub-goals are created. Sub-goal state is obtained by the following steps:

1. Make a sample with uniform sampling mentioned above, but with one less angle and length. We call it a “partial config” because there is a segment missing.
2. With the partial configuration we know where the other end of the arm is. We use the end effector location and grapple point location to “draw” the last segment through calculating the required length and angle of the last segment.
3. Repeat the process 10 times to obtain a list of sub-goals and take the elements as the goal state of the current search problem.

The pseudocode of sub-goal generation is shown as below:

Start point (x1, y1) is the current grapple point

Goal point (x2, y2) is the next grapple point

Create an empty sub-goal list to keep the goal we find.

For I in range (10) {

For j in range (Number of segments - 1) {

Sample some lengths and angles

}

If grapple point is even order {

Partial config = Generate configuration (Start point, length and angle)

If the distance between ee2 and Goal point is within the length limit {

Calculate the angle and length of the last segment

Use the parameters to generate a completed configuration

}

If the configuration is collision free {

add the configuration into sub-goal list}

}

}

Else do the former steps but with ee2 grappled at the grapple point

}

3. Which class of scenarios do you think your program will be able

to solve and under what situation(s) do you think your program will fail? Please explain your answers.

Our solver can solve all one grapple point maps with respect to 3 joints with fixed arm length (e.g. 3g1\_m0 to 3g1\_m2) and 4 joints with telescoping length arms (e.g. 4g1\_m1). It can also solve all two grapple points maps and three grapple points with 3 joints safely (e.g. 3g3\_m1). If there are 4 joints and 3 or 4 grapple points, it fails sometimes but still have a large chance to find an appropriate path within 3 attempts.

The program cannot solve any map with 5 or more joints.

Here is a table showing which map could be solve in what time period.

|  |  |  |  |
| --- | --- | --- | --- |
| Map | Proportion of successful trials | Time | Step count |
| 3g1\_m0 | 3/3 | 6.9s | 2506 |
| 3g1\_m1 | 3/3 | 8.05s | 6513 |
| 3g1\_m2 | 3/3 | 8.13s | 7015 |
| 3g2\_m1 | 3/3 | 23.01s | 7015 |
| 3g2\_m2 | 3/3 | 30.18s | 10521 |
| 3g3\_m1 | 3/3 | 24.03s | 9520 |
| 4g1\_m1 | 3/3 | 28.78s | 8016 |
| 4g1\_m2 | 3/3 | 28.45s | 8016 |
| 4g3\_m1 | 3/3 | 70.18s | 14028 |
| 4g3\_m2 | 3/3 | 97.62s | 22044 |
| 4g4\_m1 | 2/3 | 108.34s | 20541 |
| 4g4\_m2 | 1/3 | 110s |  |
| 4g4\_m3 | 0/3 | 115s | - |
| 5g3\_m1 | 0/3 | Exceed time limit | - |
| 5g3\_m2 | 0/3 | Exceed time limit | - |
| 5g3\_m3 | 0/3 | Exceed time limit | - |

It is obvious to see that with the joints number, grapple point amount and map complexity increasing, the time for solver to find a valid path and failure probability for finding valid path increases dramatically.

For maps with more than 4 joints and 3 grapple points, the failed attempt is mainly caused by three errors:

1. Running time exceeds the time limit (1 minutes for level 1 and 2, 2 minutes for level 3 and 4)
2. Path cannot be found
3. Collision in some primitive steps

Here are a few issues that may lead to the failure:

* Search algorism (Using BFS for the solver) is not efficient
* The size of sample points is not large enough to find the path
* Sampling strategy is not intelligent enough to find more useful sample points
* Some obstacle collision is not detected when checking the validation of path (Each path is separated into only 20 segments, which is not enough to guarantee a perfect collision free path)

If advanced search algorism was applied for example A\* search, it can reduce the searching time so that the solver can solve 5 grapple points problem much quicker. Since our search algorism is not good enough, we cannot generate more sample points otherwise it will also exceed time limit.

In order to increase the efficiency, during the collision free check, each path is separated into 20 segments. However, there might be obstacles smaller than the distance between checked node, so that the collision cannot be detected. Hence the path found by the solver is sometimes not valid.

However, unless a more intelligent algorithm is found, the problem cannot be solved by simply increasing the accuracy. If we just increase number of segments for each path to check collision, the run time would increase proportionally and cause the run time to exceed the time limit. The same problem also applies to the sampling size, as a larger size increases the probability to find a path but also significantly increases the running time.