CS 5335 Project

Team: Deepak Bansal, Sugandha Kher

Project #3

Content

Installation

Requirements to run the program:

1. MATLAB, preferably latest version. Try latest version if you face any issue with older versions.
2. Peter Corke's Robotics Toolbox 9.10
3. Bioinformatics Toolbox

To execute the program, run `startup_rvc` in MATLAB.
Then run `main()` to run the main project file.

Problem

The problem that we're trying to solve is to use MATLAB Robotics Toolbox to implement Probabilistic Roadmaps Algorithms (PRM) instead of Rapidly-Exploring Random Trees (RRT). We've a highly complex obstacle environment and Puma 560 robot in our workspace.

Algorithms

We've geometry of robot and obstacles as our input. To create a path, we use Probabilistic Roadmaps algorithm which involves three things:

1. Uniform Sampling which creates a graph.
2. Adding source and destination to the graph created by Uniform Sampling.
3. Finding a path from source to destination in given graph.

Uniform sampling

First of all, the program uniformly samples the configuration space. Uniform sampling gives a set of vertices and edges as output which form our graph.

The algorithm is as follows:

```
1. V ← ∅ and E ← ∅
2. repeat
3.     q ← a configuration sampled uniformly at random from c
4.     if CLEAR(q) then
5.         Add q to V
6.         $N_q$ ← a set of nodes in V that are close to q
7.         for each q′ ∈ $N_q$, in order of increasing d(q, q′)
8.             if LINK(q′, q) then
9.                 Add an edge between q and q′ to E.
```

This algorithm is implemented in `main` function in our project's `main.m` file.

In our implementation we first create few sets,

1. `vertices` for V in above algorithm i.e. vertices.
2. `edgesX` for an edge starting i.e. source node in an edge.
3. `edgesY` for an edge target i.e. destination node in an edge.
4. `weights` for the weights of edges, or in our case, distance between two constituting nodes of an edge.

Afterwards, we are creating following the algorithm and running the loop from line 2 to line 9 of the algorithm. In this, we're taking a number of samples and checking if they're clear of any obstacles.

If a node is clear of all the obstacles then we move on and find all its closest vertices within a particular distance and keep them in an array called `nq`.

Then, for each vertex in `nq`, we add those nodes in `edgesX` set with which this vertex's edge is clear while adding this vertex in `edgesY` at corresponding index of `edgesX`.

We've created two functions `clearNodes` and `linkNodes`. Former checks if the new vertex is clear of any obstacle and latter checks for an edge.

Moreover, we add distance between these vertices in `weights` set at corresponding index of `edgesX` and `edgesY`. After running for a certain number of samples, we get our graph consisting of vertices and edges between those vertices along with weights on those edges i.e. distance between vertices in an edge.

The program also includes various conditions for edge cases. For example, if the program is not able to find any vertex clear of obstacles, it'll run the loop again and try to find some vertices. It'll retry 10 times and if still not able to find, the program will exit after displaying a message conveying this fact.

The graph G consisting of edges and vertices we get in the process is called Probabilistic Roadmap. The nodes in this graph G are called milestones.

Query Processing

After uniform sampling, we connect the starting configuration q_{init} of robot and desired goal configuration q_{goal} to the graph.

To add q_{init} to the graph, the program will add it to vertices set. Afterwards, it'll try to find nearest vertices to this source vertex within a particular distance. If unable to find, we're increasing the distance to find within. We try this 10 times and if still unable to find a nearest vertex, program exits.

If we're able to find nearest vertices, we add them to `nq` of our source vertex. Then we try to find if there exist obstacle free edges from source vertex to vertices in nq. If such edges exist, we add those vertices in `edgeX` and source vertex in `edgesY` and distance between them in `weights` for each new vertex in `edgeX`.

If program is unable to find any clear edges, it'll exit since we can't find a path from source to destination in that case.

Similarly it'll run for destination vertex `q`$_{goal}$ and add it to the graph. If unable to find any nearest node or clear edge to nearest nodes, the program will exit since we can't find a path to destination in that case.

Finding Shortest Path

Thereafter, we try to find shortest path. Our program uses `shortestpath` function of MATLAB. First we create a graph `G` using graph function of MATLAB by sending it `edgeX`, `edgeY` and `weights` are arguments. Then we call function `shortestpath` which takes the graph `G`, source vertex and destination vertex as arguments. We send index of source vertex and index of destination vertex as in vertices set.

This function returns an array which describes path between source and destination. The array has couple of values representing vertices as nodes in the path. The first vertex is source, the last one is destination and there are one or more vertices in between them denoting the path from source to destination.

Tests

The program has been tested on various sources and destination and has been verified to output correct indices in answer for source and destination as well as the indices of path in out don't collide with obstacles. We're using a couple of spherical obstacles in our workspace. Additionally, we've added handling for many edge cases which could break program and have exited program with message describing the reason. The number of vertices in output path has been observed increasing with number of sampled vertices.

Conclusion

The PRM motion planning algorithm avoids collision with obstacles and finds a path between given source and target vertices. First we construct a graph and then query it and find our path if one exists.

One observation is that the implementation of algorithm can be done in a number of ways with varying time complexity. It involves multiple nested loops and conditional statements. The time complexity directly proportional to the number of random samples we want to take. Moreover, search for nearest vertices is directly proportional to the size of vertices set. Same goes for finding out edges. This may create program very slow.

An advantage is the PRM claim that if a path is returned, it is always correct. However, difficulty arises as it also suggests that if no path is found, the answer may or may not be correct.

Additionally, the sampling may often result in many small connected components, the solution to which is resampling. Resampling for expansion requires us to create a tree in addition to existing graph, making the implementation highly complex. Moreover, it includes randomness of selection. Gaussian sampler can be used for narrow pathways.

Thank you.