Stony Brook University
CSE378 – Introduction to Robotics – Spring 18
Homework 3, Due: 27-Apr-2018 11:59pm

In this homework, you will work on a motion planning problem. We will consider a two-link robotic arm in 2D space. The whole environment is on 2D surface, bounded by $[-40, 40]$ for both x, y coordinates. There is several (maybe 0) of triangular obstacles scattering on the plane. The base of the robot arm is fixed at $[0, 0]$. The length between the base and the middle joint is $L_1 = 20$ and the length between the middle joint and the tip is $L_2 = 10$. The total degree of freedom in this system is 2: rotation of the base joint and the middle joint. These two joints can rotate between $[0, 2\pi]$ (please use radians thorough this homework). You may use $[-\frac{\pi}{2}, \frac{\pi}{2}]$ if you feel this is more convenient. The whole simulation environment is shown in Fig. 1.

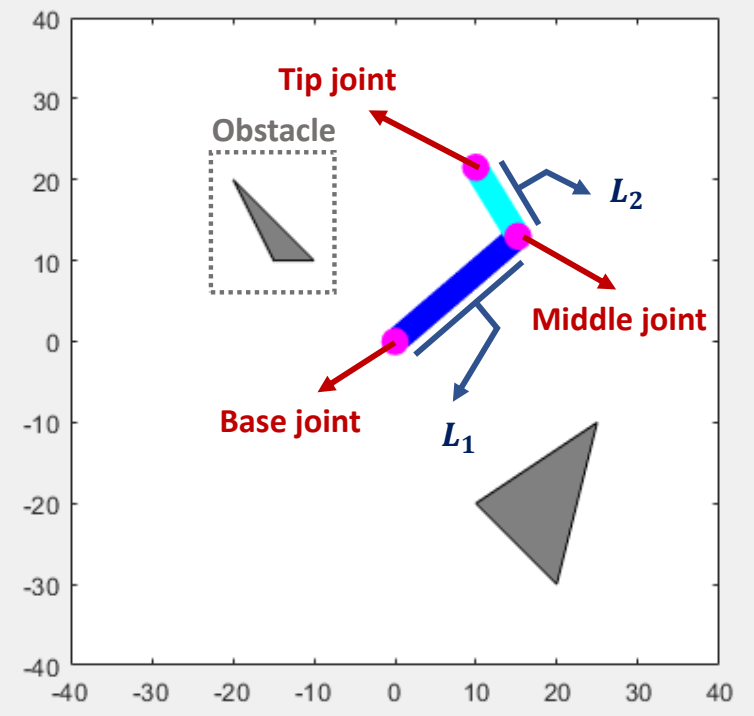

Figure 1: **The environment configuration.**

The objective of this homework is:

1. Implement a function to perform Inverse Kinematic. For a desired location of the tip joint, output the angles of the middle joint and the base joint.

2. Implement the Rapidly Exploring Random Tree (RRT) planner for an environment. The locations of the obstacles are provided as the input to the planner.

3. Implement a function to for motion planning. Given a starting point and a target point of the tip joint, you need to find a continuous motion for both the middle joint and the base joint to drive the tip joint from the starting point to the target point, without hitting the obstacle.

We have provided a two link arm simulation environment `M_TwoLinkArm.m`. You can look at this class to understand what it does but you should not modify it. File `m_main.m` is an example of how we will evaluate your policy. We have provided several obstacle configurations for you to test your motion planning implementation. We will use a different set of environment to grade your solution.

Question 1. Inverse Kinematic

Implement the function in the file `inverseKinematic.m`. The signature of the function is:

$$[\theta_1, \theta_2] = inverseKinematic(robotEnv, pos),$$

where $robotEnv$ is the instance of the environment `M_TwoLinkArm.m`. pos is the desired tip joint position. This function should return the rotation angle of the base joint θ_1 and the middle joint θ_2. If there is no feasible solution, θ_1 and θ_2 should be set to empty []. If there are multiple solutions, you can return any one of them. You don't need to consider the obstacles in this question.

Hint: You can follow the method in the lecture slides to solve this problem as one is shown in Fig. 2. You can validate your implementation by calling the method `forwardKinematic()` of `M_TwoLinkArm.m` to check for consistency.

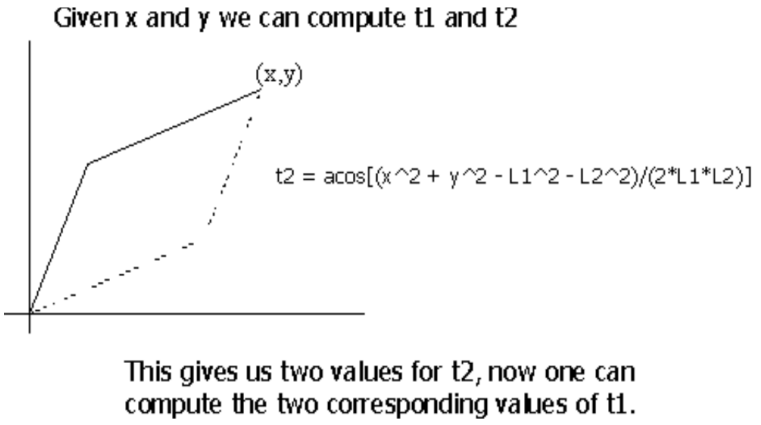

Figure 2: **Inverse Kinematic.**

Question 2. RRT planner

You need to implement the function in the file `createPlanner.m`. The signature of the function is:

$$[Node, Edge] = createPlanner(robotEnv, K),$$

where $robotEnv$ is the instance of the environment `M_TwoLinkArm.m`. The input to the function is a robot environment $robotEnv$ and K the number of iterations for running RRT. The output of the function is a graph, represented by a set of nodes and an adjacency matrix. $Node$ is a $2 \times K$ matrix for the K nodes of the graphs, each node correspond to a vector of θ_1 and θ_2. $Edge$ is the $K \times K$ adjacency matrix; $Edge(i, j) = 1$ if there is a edge connecting node i and node j, and $Edge(i, j) = 0$ otherwise.

The returned graph should satisfy the following to conditions: 1) no graph node should correspond to a configuration where the robot arm collide with the obstacles. 2) two graph nodes i and j can only be connected by an edge if the difference between the corresponding joint angles is smaller than 10 degrees. Formally, suppose θ_1^i, θ_2^i are the joint angles for node i, and θ_1^j, θ_2^j are the joint angles for node j, then node i and j can only be connected if $|\theta_1^i - \theta_1^j| < \frac{\pi}{18}$ and $|\theta_2^i - \theta_2^j| < \frac{\pi}{18}$.

Question 3. Motion Plan

You need to implement the function in the file `plan.m`. The signature of the function is:

$$[\theta_{1s}, \theta_{2s}] = plan(Node, Edge, startXY, endXY),$$

where $Node$ and $Edge$ are the matrixes produced by the function `createPlanner()`. $startXY$ and $endXY$ are starting and desired positions of the tip joint. You should search the graph to find a path from the node that is nearest to the starting point to the node that is nearest to the end point (you should use `inverseKinematic` to find the point in the configuration space). You may want to take a look at the Dijkstra's algorithm: `https://bit.ly/1MURZs9`. θ_{1s}, θ_{2s} are two $N \times 1$ vectors where N is the number of steps the arm needs to move the tip joint from $startXY$ to $endXY$. You can visualize the motion of the arm by calling the member function `execute()` of a `M_TwoLinkArm.m` It will also check for collision. For a smooth motion, the difference between two consecutive joint angles in the two vectors θ_{1s}, θ_{2s} should be smaller than $\frac{\pi}{18}$.

Note that under certain obstacle configuration and starting/end point combination, there might be no feasible solution. In this case, your function should return two empty vectors θ_{1s} and θ_{2s}.

Look at the file `m_main.m` for an example of how we will call and evaluate your code:

```
% obstacles is m*6 matrix for m triangular obstacles
% each row is the list of coordinate of the obstacle: X1 Y1 X2 Y2 X3 Y3
obstacles = [10 -20, 20, -30, 25, -10;
             -10, 10, -20, 20, -15, 10];

robotEnv = M_TwoLinkArm(obstacles); % a robot arm instance
[Node, Edge] = createPlanner(robotEnv); % create RRT planner
startXY = [20, 10]'; % the starting and end points of the tip joint
endXY = [10, 20]';
[theta1s, theta2s] = plan(Node, Edge, startXY, endXY);

isFail = robotEnv.execute(theta1s, theta2s, 1);
```

We will provide you with some test cases in the file `maps.mat`. You can also create for your own. `maps.mat` contains 3 variables: `obstacle_list`, `start_list` and `end_list`.