

Homework II:

Planning for a high-DOF planar arm

DUE: March 6th (Wed) at 11:59PM

Description:

In this project, you are supposed to implement a planner for the arm to move from its start joint angles to the goal joint angles. As before, the planner should reside in `planner.c` file inside the `planner()` function. Currently, this function contains an interpolation-based generation of a plan. That is, it just interpolates between start and goal angles and moves the arm along this interpolated trajectory. It doesn't avoid collisions. Your planner should return a plan that is collision-free.

Note that all the joint angles are given as an angle with X-axis, clockwise rotation being positive (and in radians). So, if the second joint angle is $\pi/2$, then it implies that this link is pointing exactly downward, independently of how the previous link is oriented. Having said this, you don't really have to worry about it too much as I'm already providing a tool that verifies the validity of the arm configuration, and this is all you need for planning!

To help with collision checking I supplied a function called `IsValidArmConfiguration`. It is being called already to check if the arm configurations along the interpolated trajectory are valid or not. So, during planning you want to utilize this function to check any arm configuration for validity.

The planner function (inside `planner.c`) is as follows:

```
static void planner(  
    double* map,  
    int x_size,  
    int y_size,  
    double* armstart_anglesV_rad,  
    double* armgoal_anglesV_rad,  
    int numofDOFs,  
    double*** plan,  
    int* planlength)  
{  
  
}
```

Inside this function, you'll see how any arm configuration is being checked for validity using a call to `IsValidArmConfiguration(angles, numofDOFs, map, x_size, y_size)`; You'll also find a code in there that sets the returned plan (currently to a series of interpolated angles). You will need to modify it to set it to the plan generated by your planner.

The directory contains a map file map1.txt. Here is an example of running the test from Matlab when planning for a 5-DOF arm:

To compile the C code:

```
>> mex planner.c
```

To run the planner

```
>>startQ = [pi/2 pi/4 pi/2 pi/4 pi/2];  
>>goalQ = [pi/8 3*pi/4 pi 0.9*pi 1.5*pi];  
>>runtest('map1.txt',startQ, goalQ);
```

When you run it, you should be able to see the arm moving according to the plan you returned. If the arm intersects any obstacles, then it is an invalid plan.

NOTE: I don't check for self-collisions inside the `IsValidArmConfiguration`. So, don't worry about self-collisions.

Also NOTE: to grade your homework and to evaluate the performance of your planner, I may use a different map and/or different start and goal arm configurations.

I would highly recommend implementing either RRT-Connect or PRM planner for this assignment but you can implement anything else, as long as it works well and is a sound approach!

To submit:

Submissions need to be made through Gradescope and they should include:

1. A folder named code that contains all MATLAB and C/C++ source files.
2. A PDF writeup named <Andrew ID>.pdf with instructions to compile code, results, and everything we need to know about your implementations and submission. Do not leave any details out because we will not assume any missing information.

Grading:

The grade will depend on two things:

1. The correctness of your implementations (optimizing data structures e.g. using kd-trees for nearest neighbor search is NOT required).
2. Results and discussion.