Project 5

Final Project due Nov 29, 2020 18:30 EST *Completed*

ACADEMIC HONESTY

As usual, the standard honor code and academic honesty policy applies. We will be using automated **plagiarism detection** software to ensure that only original work is given credit. Submissions isomorphic to (1) those that exist anywhere online, (2) those submitted by your classmates, or (3) those submitted by students in prior semesters, will be detected and considered plagiarism.

Project 5 Description

In the previous project, you implemented a Cartesian controller for a 7-jointed robot arm. With this you could interactively control the position of the end-effector in Cartesian space. However, this method of control is not sufficient in the presence of obstacles. In order for the robot end-effector to reach a desired pose without collisions with any obstacles present in its environment we need to implement motion planning. In this project you will code up a Rapidly-exploring Random Tree (RRT) motion planner for the same 7-jointed robot arm. This will enable you to **interactively maneuver the end-effector to the desired pose collision-free.**

The motion_plan(...) function

You are given starter code including the `motion_planning` package, which in turn contains the `motion_planning.py` file you must edit. Specifically, you must complete the `motion_plan` function. The arguments to this function and the methods provided by the MoveArm class are all you need to implement an RRT motion planning algorithm.

The arguments to the `motion_plan` function are:

- `q_start`: list of joint values of the robot at the starting position. This is the position in configuration space from which to start
- `q_goal`: list of joint values of the robot at the goal position. This is the position in configuration space for which to plan
- `q_min`: list of lower joint limits
- `q_max`: list of upper joint limits

You can use the provided `is_state_valid(...)` method to check if a given point in configuration space is valid or causes a collision. The `motion_plan` function must return a path for the robot to follow. A path consists of a list of points in C-space that the robot must go through, where each point in C-space is in turn a list specifying the values for all the robot joints. It is your job to make sure that this path is collision free.

Algorithm Overview

The problem we are facing is to find a path that takes the robot from a give start to another end position without colliding with any objects on the way. This is the problem of motion planning. The RRT algorithm tackles this problem by placing nodes in configuration space at random and connecting them in a tree. Before a new node is added to the tree, the algorithm also checks if the path between them is collision free. Once the tree reaches the goal position, we can find a path between start and goal by following the tree back to its root.

The algorithm follows the steps presented in detail in the lecture on Rapidly-exploring Random Trees. Let us break this task into smaller steps:

- Create an RRT node object. This object must hold both a position in configuration space and a reference to its parent node. You can then store each new node in a list
- The main part of the algorithm is a loop, in which you expand the tree until it reaches the goal. You might also want to include some additional exit conditions (maximum number of nodes, a time-out) such that your algorithm does not run forever on a problem that might be impossible to solve. In this loop, you should do the following:
  - Sample a random point in configuration space within the joint limits. You can use the random.random() function provided by Python. Remember that a "point" in configuration space must specify a value for each robot joint, and is thus 7-dimensional (in the case of this robot)!
  - Find the node already in your tree that is closest to this random point.
  - Find the point that lies a predefined distance (e.g. 0.5) from this existing node in the direction of the random point.
  - Check if the path from the closest node to this point is collision free. To do so you must discretize the path and check the resulting points along the path. You can use the `is_state_valid` method to do so. The MoveArm class has a member `q_sample` - a list that defines the minimum discretization for each joint. You must make sure that you sample finely enough that this minimum is respected for each joint.
  - If the path is collision free, add a new node with at the position of the point and with the closest node as a parent.
  - Check if the path from this new node to the goal is collision free. If so, add the goal as a node with the new node as a parent. The tree is complete and the loop can be exited.
- Trace the tree back from the goal to the root and for each node insert the position in configuration space to a list of joints values.
- As we have been following the branches of the tree the path computed this way can be very coarse and more complicated than necessary. Therefore, you must check this list of joint values for shortcuts. Similarly to what you were doing when constructing the tree, you can check if the path between any two points in this list is collision free. You can delete any points between two points connected by a collision free path.
- Return the resulting trimmed path.

Setup

As in the previous projects, please make sure that the first thing you do in your workspace is to `source` the '`setup_project5.sh`' script. This starts up the simulated robot and the interactive controls you can use to move it. You can now press the 'Connect' button and you should see both the robot and the controls.

Grading/ Debug

After you have implemented the RRT algorithm, you can run your code with `rosrun motion_planning motion_planning.py` and use the interactive controls. You can move the controls around in space, however the robot will not immediately follow as it did in the last project. Instead, right clicking on the controls will open a menu, in which you can command the arm to move to the desired position. You can also add obstacles of varying complexity or run a grader we provide for you to test your code. This will test your implementation on all three objects and tell you if there have been any collisions. It gives your algorithm 10, 30 and 120 seconds for each object in order of increasing complexity.

Keep in mind that the RRT algorithm is stochastic in nature. That means that it will have different results every time you run it. Therefore, it is possible that the algorithm finds a path within the time given one time and times out another time. Particularly for the most complex obstacle running time can vary considerably. The same of course is true for the grade you get when you press the 'Submit' button. We encourage to test with the grader we provide for you until you get consistent results. The delay between you submitting and receiving a grade can be a little longer than in the previous projects, but should be no longer than 5 minutes.

NOTE: Since the grading process of Project 5 takes more time than the rest of the four projects, we intentionally removed the grading report function but this won't influence your grade. You can still review your grade on the "Grades" tab on the top right corner on Vocareum.

If you submit your project before November 22th 2020, 23:30 UTC, you will earn an early submission bonus. The maximum number of points you can earn in the class is still 100. **The project's final due date is November 29th 2020, 23:30 UTC**.Good luck!

Launch Project 5 (External resource) (15.0 / 15.0 points)

Your email address will be used to identify your submission entry.

Launch Project 5

‹ Previous | Next ›