Lab 3: Motion Planning with a 6-DOF Manipulator

Due: Apr. 12th 23:59 PST

In this lab assignment, you will implement the RRT algorithm for a 6-DOF robotic manipulator. To perform the assignment, you will need to have installed the AIKIDO infrastructure. We provide for you the file **adarrt.py**, which contains several methods you will fill in. During development, you will run your code in simulation with -

```
$ python adarrt.py --sim
```

The python file contains the following classes and functions:

- `AdaRRT`: This is the main class. It initializes the start and goal node, the number of iterations, the step_size δ when extending a node in the tree, the desired goal precision ϵ and the joint limits of the robot. It also includes information about the environment, which includes a table, a soda can that we want to grasp, the robot and a set of constraints that check for collisions. This implementation will be very similar to the RRT you built in HW4, with a few modifications discussed below.
- `AdaRRT.Node`: A Node object should contain a copy of the state, a pointer to the parent node in the tree, and a list of pointers to all its child nodes in the tree. A state in the provided code is a 6D `np.array` that contains the robot's configuration.
- `main`: this function specifies the start and goal configurations, sets up the RRT planner and computes a path. It then calls the AIKIDO function `compute_joint_space_path`, which generates a trajectory for the robot to follow.

Note: If you are using ROS-Noetic/python3 for this lab, you will need to modify **adarrt.py** by finding two instances of **raw_input** (should be on lines 200 and 253), and replace both with **input**. Then find two instances of **time.clock** (should be on lines 248 and 251), and replace both with **time.process_time**.

Steps to complete the lab:

1. **Implement an RRT algorithm** by filling in the code in the provided file. For starting configuration q_S and goal configuration q_G, and parameters ϵ and δ use:

$$\begin{aligned} q_S &= [-1.5, 3.22, 1.23, -2.19, 1.8, 1.2] \\ q_G &= [-1.72, 4.44, 2.02, -2.04, 2.66, 1.39] \\ \delta &= 0.25 \\ \epsilon &= 1.0 \end{aligned}$$

   Make sure you have `roscore` running before starting your RRT!

2. **Visualize the trajectory in rviz**. First, execute your `AdaRRT` implementation, but don't execute the trajectory. Then, open rviz from the command line using `rosrun rviz rviz`. In the bottom left module, click the "Add" button and navigate to the "By Topic" tab. You should see a `InteractiveMarkers` topic under `/dart_markers`. Add the topic before executing your trajectory generated by `AdaRRT`.

3. Use an off-shelf screen capture software (e.g., `https://itsfoss.com/kazam-screen-recorder/`) to **record a video** of the trajectory. Include the video in the root of your GitHub repo as a file named `question-3.mp4`.

4. The RRT trajectory is typically jerky. Typical planners use shortcutting algorithms to make the plath smoother. **Replace the function** `ada.compute_joint_space_path` with `ada.compute_smooth_joint_space_path`. Capture the new trajectory with two videos – one showing the default isometric view, and another showing the top view. Include the videos in the root of your GitHub repo as files named `question-4-default.mp4` and `question-4-top.mp4`.

5. The goal precision ϵ of 1.0 in the previous question is too large. In order to avoid collisions, we need to improve the precision. However, this dramatically increases the time to compute a solution. To improve computation, **add a method** `_get_random_sample_near_goal` that generates a sample around the goal within a distance of 0.05 along each axis of the search space. Then, change the `build` method so that it calls `_get_random_sample_near_goal` with probability 0.2 and `_get_random_sample` with probability 0.8. Reduce ϵ to 0.2.

   Write down your observations in the PDF file. Also capture the new trajectory with two videos – one showing the default isometric view, and another showing the top view. Include the videos in the root of your GitHub repo as files named `question-5-default.mp` and `question-5-top.mp`.

6. Explain why it is not a good idea to call `_get_random_sample_near_goal` with probability 1.0. Also present an example where this could be problematic. Write your answer in the PDF.

7. **Answered questions on lab-3-report.pdf**

In-person lab: Once you are confident in your simulation results, you are ready to run it on the real robot. This can be done on the lab workstations with -

```
$ python adarrt.py --real
```

Refer to Piazza for more instructions on scheduling time in the lab.

Additional Questions

As a very rough guideline, we anticipate that for most teams, the answers to each question below will be approximately one paragraph long (or about 4-5 bullet points). However, your answers may be shorter or longer if you believe it is necessary.

Resources Consulted

Question: Please describe which resources you used while working on the assignment. You do not need to cite anything directly part of the class (e.g., a lecture, the CSCI 545 course staff, or the readings from a particular lecture). Some examples of things that could be applicable to cite here are: (1) did you get help from a classmate *not* part of your lab team; (2) did you use resources like Wikipedia, StackExchange, or Google Bard in any capacity; (3) did you use someone's code (again, for someone *not* part of your lab team)? When you write your answers, explain not only the resources you used but HOW you used them. If you believe your team did not use anything worth citing, *you must still state that in your answer* to get full credit.

- Used Wikipedia (https://en.wikipedia.org/wiki/Rapidly_exploring_random_tree) to understand RRTs at a high-level.

Team Contributions

Question: Please describe below the contributions for each team member to the overall lab. *Furthermore, state a (rough) percentage contribution for each member*. For example, in a team of 4, did each team member contribute roughly 25% to the overall effort for the project?

- Harshita: implemented the build function for the RRT and _get_random_sample_near_goal
- Matt: implemented the rest of the code
- Rohan: tested the code on the physical robot arm at lab
- Scott: helped Rohan test the code and answered questions 4 and 5 on the lab