

## Lab 3: Motion Planning with a 6-DOF Manipulator

CSCI 545

Team Roomba

Brandon Rathburn, Devayani S Krishnan, Lakshya Ahuja, Bilin Pattasseril, Rut Patel

**(72 points total)**

### 1 Main Implementation

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  $\delta$  when extending a node in the tree, the desired goal precision  $\epsilon$  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.

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  $\epsilon$  and  $\delta$  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 `roslaunch 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.

- (a) Accept the GitHub Classroom invitation  
(GitHub Classroom)
- (b) Download the docker image (we will not use the same docker images as in Lab1 and Lab2).

If use **ARM-64** architecture:

- If use Python2:  
Download and directly load this docker image:  
(Google Drive)  
`docker load -i cs545-lab3-melodic-arm64.tar.`
- If use Python3:  
Download and directly load this docker image:  
(Google Drive)  
`docker load -i cs545-lab3-noetic-arm64.tar.`

If use **X86-64** architecture:

- If use Python2:  
Download and directly load this docker image:  
(Google Drive)  
`docker load -i cs545-lab3-melodic-x86-64.tar.`

If build your own Docker image:

- Follow this instruction:  
(Google Drive)

- (c) Build the provided workspace (already in the docker image, you can skip this step): (Google Drive)

Unzip it to your home directory. Before running the assignment codes, make sure you have run `source /ros_ws/devel/setup.bash` in the same terminal as your lab script. Remove build, devel, and log from your ros\_ws and rebuild the workspace by running

```
catkin_make_isolated -j4 # Use 4 parallel jobs
```

- (d) Create Docker Container

- (e) Run lab3 code

If use [ARM-64](#) architecture:

- Terminal 1: Run roscore  
Run roscore  
`source /opt/ros/melodic/setup.bash`  
`roscore`
- Terminal 2: Run your script  
Put your script in ' /ros\_ws/src/lab3/adarrt.py'.  
Run your script:  
`source ~/ros_ws/devel_isolated/libada/setup.bash`  
`export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:/root/ros_ws/src/libada`  
`python ~/ros_ws/src/lab3/adarrt.py --sim`
- Terminal 3: Run Rviz visualization  
`source /opt/ros/melodic/setup.bash`  
`source ~/ros_ws/devel_isolated/libada/setup.bash`  
`export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:/root/ros_ws/src/libada`  
`roslaunch rviz rviz`

If use [X86-64](#) architecture:

- Terminal 1: Run roscore  
Run roscore  
`source ~/.bashrc`  
`roscore`
- Terminal 2: Run your script  
Put your script in ~/lab3/adarrt.py.  
Run your script:  
`source ~/.bashrc`  
`python ~/lab3/adarrt.py --sim`
- Terminal 3: Run Rviz visualization  
`source ~/.bashrc`  
`roslaunch rviz rviz`

3. **(40 points)** 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. **(10 points)** The RRT trajectory is typically jerky. Typical planners use shortcutting algorithms to make the path 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. **(10 points)** The goal precision  $\epsilon$  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  $\epsilon$  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.mp4` and `question-5-top.mp4`.

Question 5 Observations: While the distances traveled between the robotic arms are quite similar, the notable difference is the Yaw value, comparing Question 4's Yaw value of 6.24 to Question 5's Yaw value of 0.08. While the end effector in both paths reach the target, the path using the algorithm in question 4 (without `_get_random_sample_near_goal`) appears to be more "awkward". The path in question 5 seems more "natural" to the human eye, wrapping around the outside of the cylinder. The RRT trajectory in question 4 explores the configuration space uniformly, which results in an unnecessary bulky path.

6. **(10 points)** 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.

Question 6: If we call `_get_random_sample_near_goal` with probability 1.0, it is impossible (there is 0.0 probability) for a sample to be collected outside of this 0.05 distance along each axis of the search space. This eliminates the "Random" part

of the Rapidly-exploring RANDOM Tree Algorithm, essentially turning this into a greedy planner. Obstacles are more of a hindrance in this scenario. If an obstacle exists between the start and the goal, the planner will never find a path to the goal. This is because the robot will go straight towards the goal, hit the obstacle, and always fail to navigate around it.

**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.

## 2 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.

### 2.1 Resources Consulted

**Question: (1 points)** 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.

Resources Consulted: To better understand and visualize the concept of RRT, this YouTube video (<https://www.youtube.com/watch?v=OXikozpLFG0>) served as a guide to the implementation of the RRT algorithm in step 1. Outside of this, no other resources were consulted.

### 2.2 Team Contributions

**Question: (1 points)** 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?

Team Contributions: Everyone contributed 20% to the overall effort for the project. Since not everyone was able to make it to the in-person lab due to time constraints and other commitments, those members picked up extra slack where needed, whether it was with completing extra parts of the code or the report.