Lab 3: Motion Planning with a 6-DOF Manipulator

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. The 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.

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 and include it with the submisison.

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 a video, and include it in the submission.

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. What do you observe? Capture the new trajectory.

6. Why not calling _get_random_sample_near_goal with probability 1.0? Present an example where this could be problematic.