Lab 2: Motion Planning

Robot Autonomy

Prof. Oliver Kroemer

0 Prerequisites

In this lab, you will implement RRT Connect and run it on the real robot to follow a path that avoids a static obstacle.

Hint: You can split up your lab group to work on Sections 1 and 2 in parallel. These two sections do not depend on each other.

0.1 Update Lab Repo

1. Run `git pull` on the lab repo `robot-autonomy-labs` on your machine.
2. Install the requirements in `requirements.txt` by going into your virtualenv (or conda env) first, then running `pip install -r requirements.txt`

0.2 Starting the Robot

Refer to the last lab for detailed instructions.

1. Turn on the robot.
2. Unlock the joints via launching a remote Firefox session on the Control PC.

0.3 Note on Safety

When running a program that commands the robot, please:

1. Stay outside of the robot's workspace (defined by the edges of the table).
2. **Always keep a hand on the e-stop. Under no circumstances should the e-stop be out of reach.**

1 Specifying the Planning Problem

1.1 Launch Control PC

1. Remote launch `franka-interface` on the Control PC by:

   `cd` to where `frankapy-public` is on your machine.

   Run `bash bash_scripts/start_control_pc.sh -i iam-<name>`

2. Source the compiled frankapy catkin workspace in the `frankapy-public` folder:

   `source catkin_ws/devel/setup.bash`

1.2 Registering the Obstacle Box

We will first find the dimension and the pose of an obstacle in the workspace. This is usually done with vision, but a quick and easy way is to simply move the robot's end-effector to the vertices of the box obstacle to "register" its size and pose.

1. Place a cardboard box in the workspace like so:

Figure 1: Obstacle Placement

   Once you've placed the box, be careful not to move it again! If the box is moved, you'd have to re-register it.

2. Run `python utils/close_gripper.py` to close the gripper.

3. Run `python utils/run_guide_mode.py` to record the positions of either pair of the top two diagonal corners of the box by placing the gripper tip near those corners.

4. Use the recorded positions to compute the dimensions, position, and rotation of the box in the workspace.

   Note:

   x-axis is coming out of the robot base toward the grippers in home pose, and z-axis is up.

   If the box is axis-aligned with the workspace, then rotation angles should all be 0.

   You can treat the bottom of the box 0 on the z-axis.

5. Replace the obstacle box in the `boxes` array in `test_rrt_connect.py`

1.3 Getting Start and Target Joint Configurations

We will replace `joints_start` and `joints_target` with new ones that are closer to the obstacle.

1. Use `run_guide_mode.py` to record the joint positions for when the end-effector is to the left and to the right of the box:

Figure 2: Start (left) and Target (right) Configurations

2. Replace `joints_start` and`joints_target` in `test_rrt_connect.py` with these recorded joints.

2 Implement RRT Connect

RRT Connect is often faster than RRT in practice because it extends an additional tree from the target node. The search alternates extending each tree until two nodes in the tree are close enough to be connected.

1. Fill in the `extend` function in `rrt_connect.py`. See Algorithm 1 for details.

Algorithm 1 ConstrainedExtendRRTConnect

Require: T_0, T_1

while True **do**
  $q_{sample} \leftarrow$ SampleValidJoints()
  $q_{near} \leftarrow T_0$.Nearest(q_{sample})
  $q_{new} \leftarrow q_{near} + \min(\Delta q_{step}, \|q_{sample} - q_{near}\|_2) \frac{q_{sample} - q_{near}}{\|q_{sample} - q_{near}\|_2}$
  $q_{new} \leftarrow$ ConstraintProjection(q_{new})
  if InCollision(q_{new}) **then**
    continue
  end if
  T_0.AddVertex(q_{new})
  T_0.AddEdge(q_{near}, q_{new})
  $q_1 \leftarrow T_1$.Nearest(q_{new})
  if $\|q_{new} - q_1\|_2 < d_{connect}$ AND IsSegmentValid(q_{new}, q_1) **then**
    return True, q_{new}, q_1
  end if
  return False, q_{new}, q_1
end while

3 Test RRT Connect

1. Replace the `ee_upright_constraint` function in `lab2/test_rrt_connect.py` with your implementations from hw2.

2. Run `roslaunch lab2/lab2.launch` - you may need to modify the `path` argument in this file to point to the `lab2` directory on your machine.

   If you run into the error `No module named defusedxml.xmlrpc` - try running the `roslaunch` command outside of your virtualenv by first running `deactivate`

3. Run `python test_rrt_connect.py` to make sure your code runs and is able to find a path. Test this with both using the constraint and not using the constraint. Check the plan in rViz and make sure it looks reasonable.

4 Run on Real Robot

1. Run `python test_rrt_connect.py --run_on_robot` to run the planner and execute the plan on the real robot.

   When the `run_on_robot` flag is given, the script will visualize the computed plan 3 times and ask if it should proceed with executing the plan on the real robot.

2. If you proceed, the script will then give you the opportunity to manually move the robot to a position that's close to the start configuration.

   This is needed because our motion planner only planned the trajectory from the start to the target configuration, and not how to get to the start configuration. So if the script were to command the robot to go to the start configuration directly, the robot may encounter collisions.

   Lastly, the script will move the robot to cycle between the start and end configurations indefinitely until you stop the program.

   Please keep a hand on the e-stop while the robot is moving!!!

3. Test your planner with and without the constraint. If your plan runs successfully on the real robot in both cases, congrats - you're done! If not, debug your code until it does.

5 Turning Off the Robot

Turn off the robot as instructed in the previous lab:

1. Reset joints
2. Press down on the e-stop to return robot to manual mode (white)
3. Click shutdown on the web interface.
4. Wait until the robot shuts down, the flip the switch on the FCI.