Assignment 3: Path Planning

Robot Autonomy

Prof. Oliver Kroemer

0 Prerequisites

In this homework, you will implement the Rapidly-exploring Random Trees (RRT) path planner for the Franka robot. You will also implement planning with constraints using Projection Sampling.

Like with HW2, you need to first launch rviz:

`roslaunch hw3.launch`

If you're using your own computer (not the VM), make sure to specify the `path` argument in the launch file to point to the `hw3` folder. This path should be an absolute path, not a relative one.

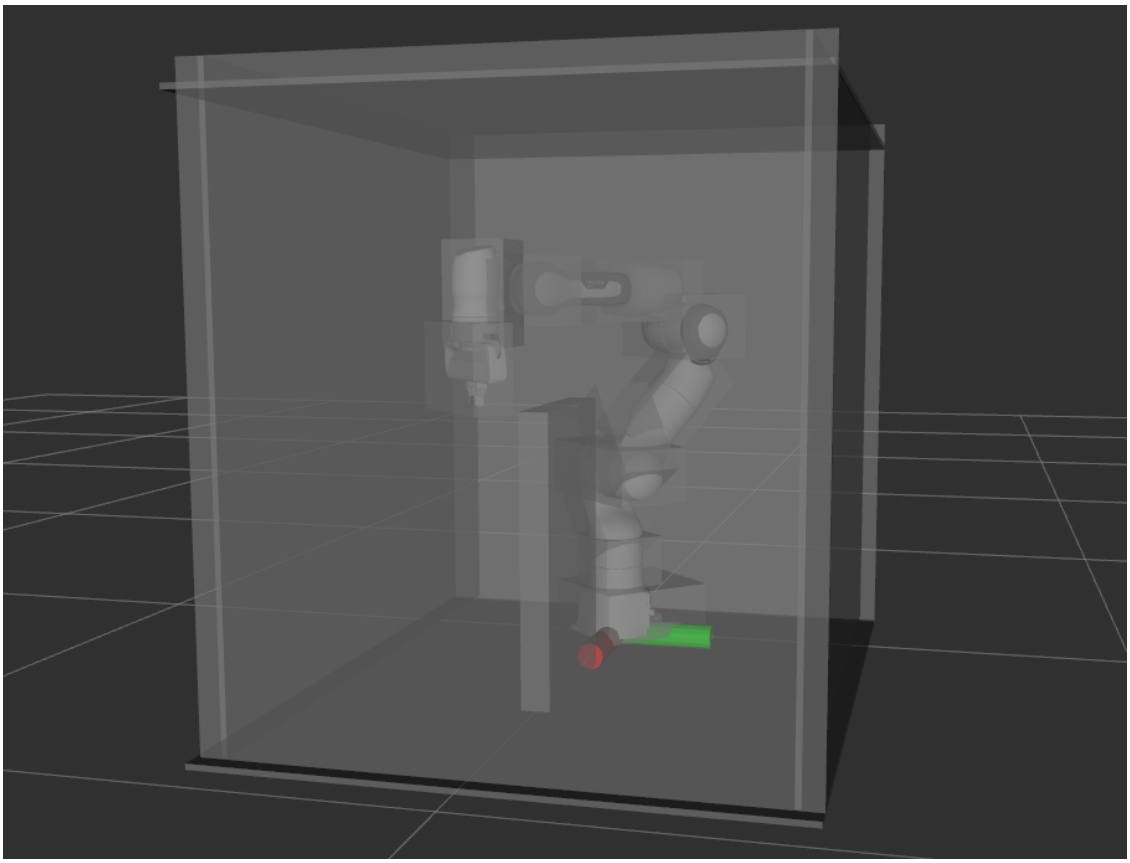

Figure 1: rViz Visualization of Franka robot, surrounding walls, and an obstacle. You will see this visualization after RRT has found a plan.

1 Path Planning via RRT

Open `test_rrt.py` - this is the runner file to perform path planning:

- We provided a collision checking function `is_in_collision` that calls `FrankaRobot`'s box collision checking function on a list of boxes. Currently, there is just one collision box in `boxes`. The 6 numbers that specify a box are its center points in the order of xyz and its lengths in those dimensions: $[x, y, z, r, p, y, l_x, l_y, l_z]$.
- The function `ee_upright_constraint` will be used in the next part for constrained planning. Ignore it for now.
- The default start and target joint configurations are `joints_start` and `joints_target`. Note they're slightly modified versions of the Franka's home joints `fr.home_joints`.

Open `rrt.py` - this contains the implementation of the RRT planner.

1) Implement RRT

(1) [5 points] Implement the `sample_valid_joints` function that uniformly samples a random joint configuration within the Franka joint limits. This function does not check for collisions.

(2) [20 points] Implement the `extend` function for RRT. We will implement a slightly simplified version of the one seen in class. See Algorithm 1. Ignore the constraint projection part for now.

(3) [10 points] After you've implemented the above two functions, you should be able to run `test_rrt.py` without errors. You are encouraged to play around with the hyperparameters in `RRT`'s `__init__` function.

Run the planner for 5 different random seeds. You can change the seed in the beginning of the `test_rrt.py` script. Your RRT implementation should be able to find a plan in the unconstrained case for the default start position, target position, and collision boxes in less than 30s in at least one of the 5 seeds. Because RRT is probabilistically complete, it is possible for some runs to take a long time to find a plan. This is why we ask for running the planner for 5 different random seeds.

In your report, please include:

1. A video of the rviz visualization of the plan running in progress.
2. Any hyperparameter that you've changed and its value.
3. Mean, min, and max planning times, number of nodes sampled, and path lengths for running with the default start joints, target joints, and collision boxes for 5 random seeds.

Algorithm 1 ConstrainedExtend

Require: T, q_{target}

while True **do**
  if $rand() < p_{target}$ **then**
    $q_{sample} \leftarrow q_{target}$
  else
    $q_{sample} \leftarrow$ SampleValidJoints()
  end if
  $q_{near} \leftarrow T$.Nearest(q_{sample})
  $q_{new} \leftarrow q_{near} + \min(\Delta q_{step}, \|q_{target} - q_{near}\|_2) \frac{q_{target} - q_{near}}{\|q_{target} - q_{near}\|_2}$
  $q_{new} \leftarrow$ ConstraintProjection(q_{new})
  if InCollision(q_{new}) **then**
    continue
  end if
  T.AddVertex(q_{new})
  T.AddEdge(q_{near}, q_{new})
  if $\|q_{new} - q_{target}\|_2 < \Delta q_{step}$ **then**
    return True, q_{new}
  end if
  return False, q_{new}
end while

2 Constrained Planning

From the plan generated in the previous part you might've noticed that the end-effector of the robot undergoes significant rotations. This may not be desirable in real-world scenarios; for example, if the robot is grasping an object, we may wish the object to remain upright.

In this section, we will implement constrained planning via constraint projections, and the constraint will be keeping the end-effector in the vertical orientation. This requires fixing two of the three angles of the end-effector pose.

1) Implement Constrained RRT

(1) [5 points] Implement the `ee_upright_constraint` function in `test_rrt.py`.
Set `constraint=ee_upright_constraint` instead of `None`.

(2) [10 points] Implement the `project_to_constraint` function in `rrt.py`. This uses gradient descent to project a given joint configuration towards the constraint manifold until a certain threshold is reached. Modify your `extend` function from the previous part to perform constraint projection if the given constraint function is not `None`.

(3) [10 points] After you've implemented these two functions, you should be able to run `test_rrt.py` with the constraint function passed to `rrt.plan` and generate a plan where the robot end-effector stays upright the entire trajectory.

Again, you're encouraged to tune the RRT hyperparameters in the `__init__` function. The values that allowed fast path planning in the previous part may not work as well in this part. There are two additional hyperparameters for constraint projection - the constraint threshold and the gradient step size.

Run the planner for 5 different random seeds. Your RRT with constraints with default the start joints, target joints, and collision box should be able to find a plan in less than a minute in at least one of the 5 seeds.

In your report, please include:

1. A video of the rviz visualization of the plan running in progress.
2. Any hyperparameter that you've changed and its value.
3. Mean, min, and max planning times, number of nodes sampled, and path lengths for running with the default start joints, target joints, and collision boxes for 5 random seeds.

3 Submission Checklist

- ☐ Create a zip of `rrt.py`, `test_rrt.py`, your 2 videos, and the report `<andrewID>.pdf`.
- ☐ Upload the zip to Canvas.