Lab 4: Planning in Task-Space Regions

Due: Dec. 1st 11:59 PM PST

In the last lab, you implemented the RRT algorithm to generate a motion plan from a given start to goal configuration. However, the robot typically does not know the goal configuration in advance. Instead, it knows only the position of the object in world coordinates, as extracted for instance from sensing. In this lab, you will implement a planning pipeline that takes as input the 3D position and identity of the can, reaches the object and pick it up. During development, you will run your code in simulation with -

```
$ python soda_grasp_ik.py --sim
```

1. We will describe the grasping task with a Task Space Region (TSR) constraint. Read [**?**] through Section 4 to build an understanding of TSRs.

2. Implement the TSR constraint for the can. Specify the bounds matrix $\mathbf{B}^w$, so that TSRs are sampled uniformly around the can at the same height i.e. only allow rotation about ϕ. You can do this by filling in the function `createBw` with the appropriate *min* and *max* values for x, y, z, ψ, θ, and ϕ.

   You can visualize the TSRs in **rviz** by calling - `viewer.add_tsr_marker(sodaTSR)`

   You can use the visualization of the end-effector frame as reference and set the height (z) so that the robot can grasp the can without colliding with the table. **Save a picture** of your simulation in rviz as `tsr_vis_1.png`

3. Modify the $\mathbf{B}^w$ matrix, so that only TSRs in the semicircle facing the robot are sampled (the robot will not have to go behind the can to grasp it). Visualize the result. **Save a picture** of your simulation in rviz as `tsr_vis_2.png`

4. The pipeline then calls `ik_generator` to compute IK solutions for the robot, using the TSR constraints that you specified. For each computed IK configuration, **use the RRT planner that you have implemented in Lab 3** to find a motion plan. Once the RRT planner finds a plan for a configuration, execute and visualize the trajectory.

5. Now the robot should grasp the can. To do that, call the function `close_hand(hand, preshape)`, which takes as input the hand. The preshape is a vector (f_1, f_2), where

$0 \leq f_1, f_2 \leq 1.6$. Specify a value for the fingers that encompass the can without grasping it too tightly. Then, call the function `hand.grab(soda)` which will attach the can to the robot's end-effector. Execute and visualize this procedure.

6. Lift the can vertically by 0.5 m using the Jacobian pseudoinverse method (Algorithm 1 in [**?**]) such that it stays in its original vertical orientation.

   Perform one iteration (lines $6-8$ in Algorithm 1) of the Jacobian pseudoinverse method with $\Delta x = [0, 0, 0, -0.5, 0, 0]$.

   Get the Jacobian at the current robot configuration by calling the function - `arm_skeleton.get_jacobian(hand.get_endeffector_body_node())`

   The first 3 columns of the Jacobian matrix are the rotational portion, and the next 3 the linear (translational) portion. Then, compute the pseudoinverse using `numpy.linalg.pinv` function. Use the function `arm_skeleton.get_positions()` to get the current configuration of the arm, and `ada.set_positions(q)` to set the configuration of the arm to q.

   Does the final pose of the can look accurate? **Report** your observations and justifications in `answers.pdf`. **Save a picture** of the final object pose as `jac_vis_1.png`

7. Rectify the implementation of the Jacobian pseudoinverse method such that the final pose of the can is exactly 0.5 m above its starting pose (without drifting in any other direction).

   Hint: Iterate using a small vertical displacement of 0.01 until you have lifted the object by 0.5 m.

8. Execute the whole pipeline and **record the result** by saving the video of the entire task as `full_trajectory.mp4`.

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 soda_grasp_ik.py --real
```

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