TECHIN 516: Robotic Sensing And Mobility

Lab 7: Pick and Place tasks and Arm Motion Planning

DATE: March 2nd. 2023

Lecturer: Queenie Qiu, John Raiti. Student: ***Shucheng Guo***

1 Learning Objectives

1. Learn about waypoint generation in robotic arm trajectories and motion planning.
2. Implement code to perform pick and place tasks in a recycle-sorting scenario.

2 Creating a robotic arm trajectory from waypoints

2.1 Trajectory planning with Bezier curve in Cartesian space for Kinova arm

Instruction: There are many trajectory planning strategies among which Bezier curve is one that is easy to apply and ideal for applications requiring a smooth trajectory. A Bezier curve is a mathematical curve defined by a series of control points, which determine the shape of the curve. The curve starts at the first control point and ends at the last control point, but it can bend and twist in any way that is defined by the other control points.

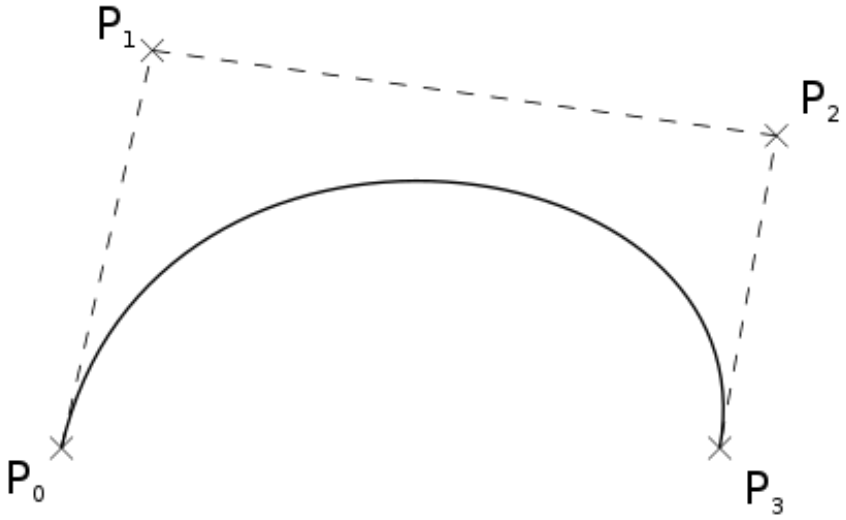
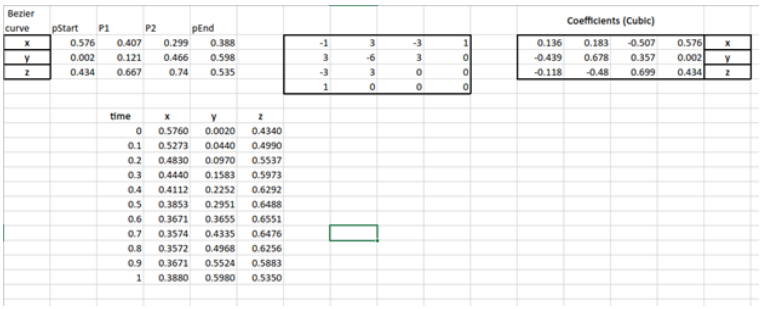


Figure 1: Cubic Bezier curve with four control points

During this session, we will guide the Kinova arm to follow a trajectory based on a Bezier curve, moving it from the Home position to the Vertical position. First, we will acquaint ourselves with the lab materials, starting with an Excel file employed to generate a trajectory using Bezier curves.

This Excel file calculates interpolate points along the Bezier curve based on 4 input control points, with two possible output versions containing 11 or 21 points.



Bezier curve	pStart	P1	P2	pEnd
x	0.576	0.407	0.299	0.388
y	0.002	0.121	0.466	0.598
z	0.434	0.667	0.74	0.535

-1	3	-3	1
3	-6	3	0
-3	3	0	0
1	0	0	0

Coefficients (Cubic)

0.136	0.183	-0.507	0.576	x
-0.439	0.678	0.357	0.002	y
-0.118	-0.48	0.699	0.434	z

time	x	y	z
0	0.5760	0.0020	0.4340
0.1	0.5273	0.0440	0.4990
0.2	0.4830	0.0970	0.5537
0.3	0.4440	0.1583	0.5973
0.4	0.4112	0.2252	0.6292
0.5	0.3853	0.2951	0.6488
0.6	0.3671	0.3655	0.6551
0.7	0.3574	0.4335	0.6476
0.8	0.3572	0.4968	0.6256
0.9	0.3671	0.5524	0.5883
1	0.3880	0.5980	0.5350

Figure 2: Bezier curve.

Then spawn a robot in an empty world:

```
$ roslaunch kortex_gazebo spawn_kortex_robot.launch arm:=
gen3 gripper:=robotiq_2f_85
```

We will use RViZ to select 4 points to generate a trajectory: Start, End, and 2 mid points. Use Home position as Start and Vertical as End. You need to find two intermediate poses (collect pose information from `$ rosrun tf tf_echo base_link tool_frame`). The trajectories generated on the file are normalized in time ($0 \leqslant t \leqslant 1$).

Select the version of your choice (11 or 21 points) and paste that information into you data.csv file. Once the data.csv file has been modified you can run the python script(plot_trajectory.py) to plot the points in 3D for visualization.

Tip: you should replace (Ctrl + H) the tab spacing with a comma (,)

So far, we have handled the position of the end-effector's pose along the trajectory with the Bezier curve. Next, we will need to generate smooth changes between the orientations that differ at positions.

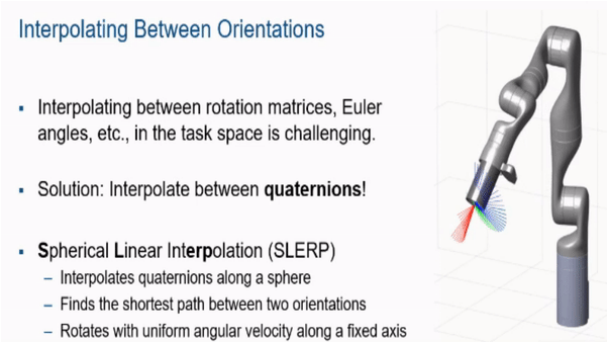


Figure 3: quaternions

For quaternion interpolation we will use one class of the python module pyquaternion. You can install it using pip:

```
$ pip install pyquaternion
```

The typical usage of pyquaternion that best suits our application is by declaring one quaternion for each start and end pose, and use one method that uniformly interpolates values in between:

```
from pyquaternion import Quaternion
qStart = Quaternion(w=<value>, x=<value>, y=<value>, z=<value>)
qEnd = Quaternion(w=<value>, x=<value>, y=<value>, z=<value>)
qList = Quaternion.intermediates(qStart, qEnd, <numIntermediates>,
            include_endpoints=True)
```

This chunk of the code is in trajectory_from_csv.py. This file combines the two composition of trajectory generation - positions along the path and quaternion interpolation. You can run the script and visualize the changes in both RViZ and Gazebo.

Tip: Execute the following command in your terminal to use pick_and_place.py class from terminal or import in different python files.

```
$ export ROS_NAMESPACE="/my_gen3/"
```

Deliverables:

1. Submit the selected control points used to generate a (Bezier curve) trajectory to go from Home to Vertical position.

Points	x	y	z
P_0	0.576	0.002	0.434
P_1	0.351	0.461	0.862
P_2	-0.113	0.245	1.220
P_3	-0.001	-0.025	1.307

Table 1: Control points in the Bezier curve trajectory

2. Compare the path you created versus the path generated by Motion Planning in RViz starting from Home to Vertical. Please provide explanation when needed.

   Answer: The self-generated path has a smaller range of motion, while the one generated by MotionPlanning in RViz has a significantly bigger one.
   In calculating the way points, only the positions of the end effector are considered, so the trajectory of my own path is more smooth as expected. When planning with RViz, it seems that the joints are all moving at the same time, finding their shortest paths without prioritizing the end end effector.
   For the above reasons, path planning with different methods may lead to distinct performances. With Bezier curve, the end effector moves the shortest and most smooth path, while the joints maybe twisted and move a longer one. With RViz, wider space is required as the movement may be bigger, but from a holistic view, the whole execution is more smooth.

3. Discuss the importance of complementing the trajectory generation process with the quaternion interpolation.

   Answer: As discussed above, generating a curved path doesn't guarantee smooth joint movement, and appropriate velocity change, which would cause trouble in path planning, like collisions and lack of accuracy. Applying quaternion interpolation, or SLERP in robotics is indispensable because it would find the shortest path between two orientations, and dictate the joints to rotate with uniform angular velocities along a fixed axis.

4. Take a screenshot or save an image from your visualized trajectory from the plot_trajectory script.

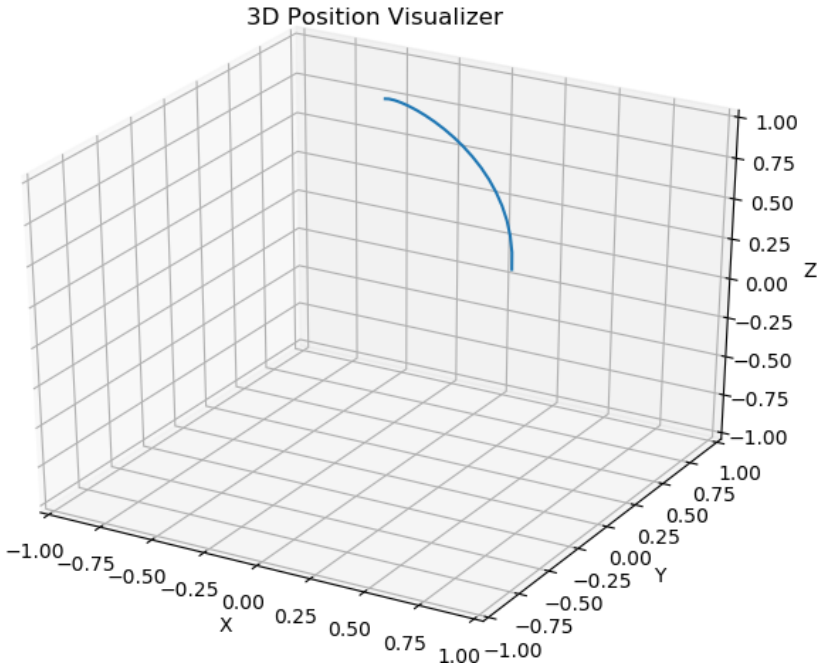


Figure 4: Visualizing the original trajectory plot

5. If you have any modifications to the script that would result in a better visualization experience, propose them here and take a second screenshot.

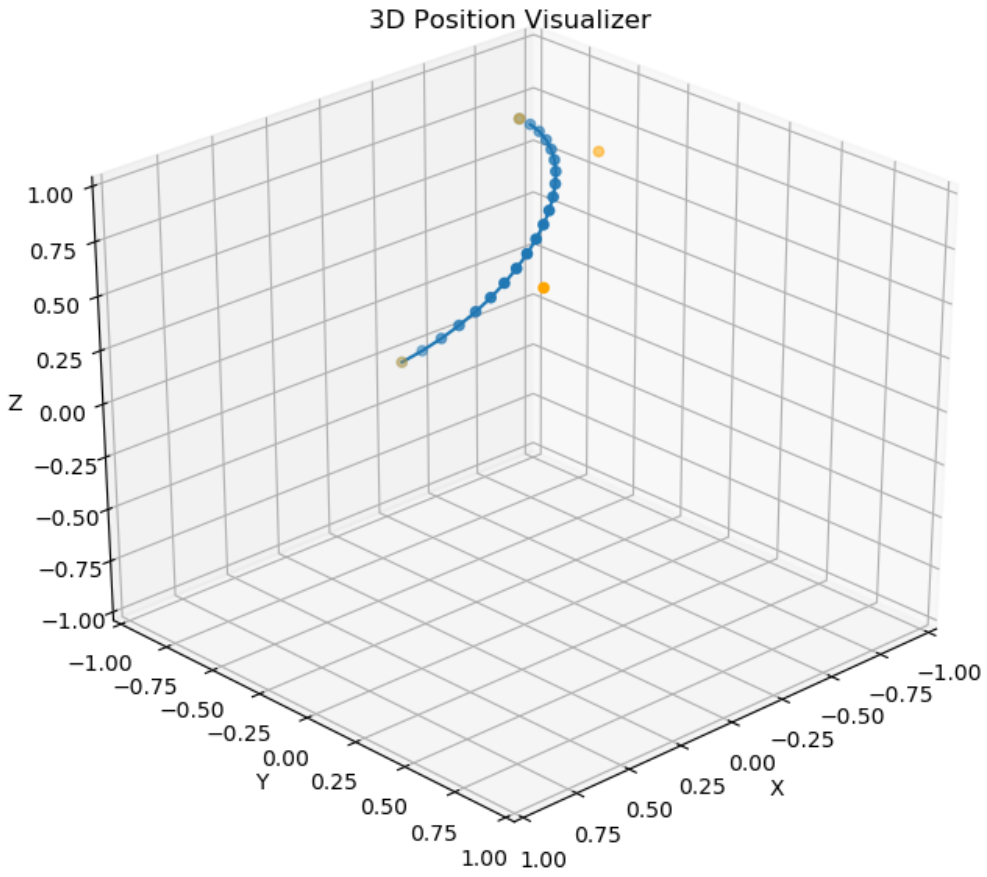


Figure 5: Visualizing improved trajectory plot with points

Answer: Compared to the original plot, where only the final trajectory is present, the enhanced plot comprises of four control points, 21 waypoints, and the line connected by the dots. The plot looks very much like the curve in 3, whose shape is determined by a series of control points.

3 Pick and Place task with obstacle avoidance

Instructions: We have now become acquainted with all the aspects related to the arm's kinematics, trajectory planning, and the use of joint angles (configuration space) or end-effector poses (task space) to direct the robot's movements. In this section, we will apply all of these elements to a sorting task, which involves picking up cubes of different colors and placing them on top of the corresponding bins. It should be noted that the simulated environment includes walls that require careful planning to avoid collisions.

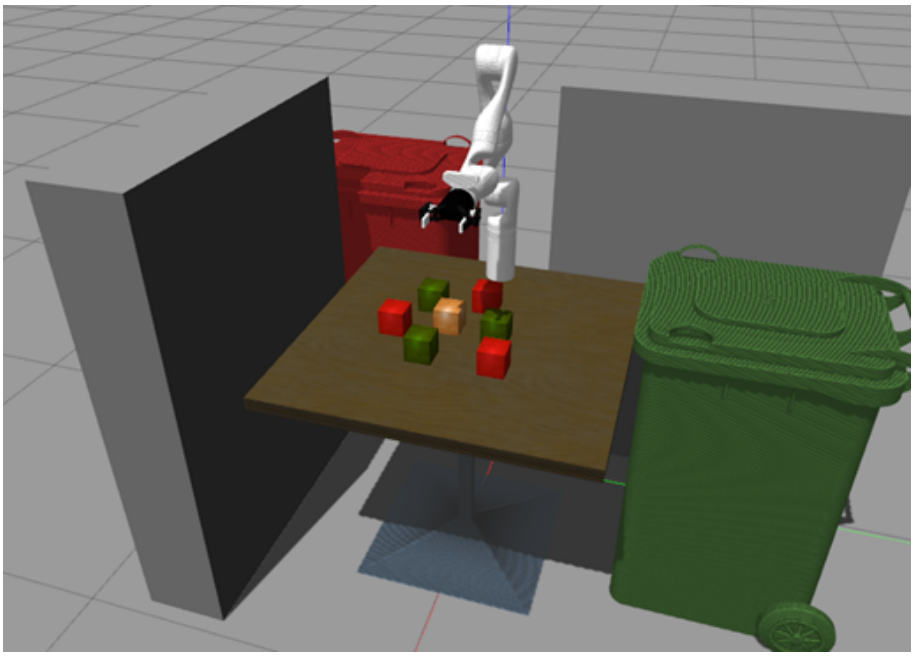


Figure 6: The planning scene

In the start_gazebo.launch file (roscd kortex_gazebo/launch) add the following argument to open the new generated world:

```
<arg name="world_name" value="<path to your Lab7_code folder>/sorting_world.sdf"/>
```

Additionally, in the spawn_kortex_robot.launch file, modify the line that starts the RViZ node to include the configuration file provided in the lab materials:

```
<node name="rviz" pkg="rviz" type="rviz" args="-d
<path to your Lab7_code folder>/pickandplace.rviz" if="$(arg start_rviz)"/>
```

Once you have completed those changes, run the following command:

```
$ export GAZEBO_MODEL_PATH=$GAZEBO_MODEL_PATH:<path to your Lab7_code folder>
$ roslaunch kortex_gazebo spawn_kortex_robot.launch arm:=gen3 gripper:
=robotiq_2f_85 z0:=0.775
```

Drawing on your prior knowledge of trajectory planning from the previous section, as well as your experience from the previous lab assignment, think about how the sorting task can be broken down into smaller sub-tasks, such as sorting the green cube and sorting the red cube. Additionally, the process for sorting the green cube can involve the same steps as the pick-and-place task from lab 6. To refresh your memory, the pick-and-place task involves the following steps:

1. Reach position above the center of the object to be picked with gripper oriented
2. Adjust gripper to open or semi-open state.

3. Decrease the end-effector's z-value position
4. Close gripper to grasp object.
5. Reach position above the desired location to place the object with gripper oriented.
6. Decrease the end-effector's z-value position
7. Open gripper to release object.
8. Increase the end-effector's z-value position to clear the object
9. Return to home position.

However, for sorting the cubes, you should consider finding waypoints that won't hit the walls, and generating trajectories that guarantee a successful movement between picking and placing.

We have only started covering the many functionalities of MoveIt! You can use the following link to go through their documentation. There are very valuable resources in Mathworks. Those resources are part of a commercial product, but it is worth looking into them. If you require additional information about the pyquaternion module, visit their documentation.

Deliverables:

1. Provide a walkthrough of your process to complete the sorting task:
   (a) What sub-tasks did you select to divide the sorting task into? What did each sub-task entail? Discuss the steps within each of your sub-tasks
      i. Approach the object to pick
         - Reach position above the center of the object to pick
         - Adjust the gripper to open or semi-open state
         - Lower the end effector by decreasing z to approach object
      ii. Pick the object
         - Adjust the gripper to close state
         - Raise the end effector by increasing z to move object up
      iii. Move it to the goal position
         - Calculate waypoints based on current and goal pose
         - Plan trajectories with waypoints to avoid crashes
         - Execute the movement and reach position above the desired location
      iv. Place the object
         - Lower the end effector by decreasing z to place object
         - Adjust the gripper to open state
         - Raise the end effector by increasing z to clear object

v. Return to home position

(b) Mention some of the challenges and how you addressed them.

Answer: One of the biggest challenges I faced was not being able to initialize RViz. After running the commands in the terminal to instantiate the arm in Kortex_Gazebo, an error would pop up, saying the parameters aren't found. This issue persisted over a week and was resolved by reinstalling the operating system and modifying the xacro parts in the launch files.
Another difficulty was to plan a route without the risk of crashing and execute with python. It was achieved by calculating waypoints with self-selected control points in the Belzier curve. Firstly, I determined the current and goal positions, the states after picking and before placing the object; then I moved the arm to up above where it wouldn't hit the walls and recorded two points as the midway control points. Waypoints were then calculated based on the four control points and the trajectories were executed without errors.

2. Report values of interest used to complete the task:

(a) Poses of the cubes, the bins, the selected waypoints, and desired placing location.

Answer: Below is an instance of a part of the sorting task, where the furthest red cute is picked from the table and placed on the reb bin. Other points of interest outside of this task aren't included to prevent confusion.

Table 2: Pose information of locations of interest

(a) Red cube

x	y	z
0.526	0.126	0.774

(b) Red bin

x	y	z
-0.798	-0.837	0.015

(c) Destination

x	y	z
-0.432	-0.429	0.992

time	x	y	z
0	0.5260	0.1150	0.0620
0.1	0.5393	0.1353	0.2069
0.2	0.4857	0.1237	0.3399
0.3	0.3813	0.0863	0.4559
0.4	0.2424	0.0295	0.5496
0.5	0.0853	-0.0406	0.6158
0.6	-0.0740	-0.1177	0.6492
0.7	-0.2192	-0.1955	0.6448
0.8	-0.3341	-0.2678	0.5973
0.9	-0.4024	-0.3284	0.5014
1	-0.4080	-0.3710	0.3520

Figure 7: Waypoints between approaching and placing object locations

(b) Selected height to approach object and place locations.

Table 3: Pose height

State	*Approach*	*Place*
Height	0.062	0.280

(c) Percentage of the gripper's open/close position.

Table 4: Gripper position

State	*Open*	*Close*
Percentage	0	30%

3. Record a video of the entire sorting task or select a specific portion of the task of the robot carrying out the necessary movements. You can choose to control the Kinova arms using Rviz or by writing code. If you opt for the latter, you can refer to the "pick_and_place.py" file for guidance.

   Google Drive link to the video of sorting a red cube