1. Trajectory Generation

We implemented a smooth joint-space interpolation function based on a trapezoidal velocity profile. The profile includes acceleration, constant velocity, and deceleration phases, ensuring continuity and avoiding sudden changes in velocity.

Key details:

- Total time: 2.0 seconds
- Acceleration time: 0.5 seconds
- Deceleration time: 0.5 seconds
- Uniform timestep of 0.02 seconds
- Velocity integrates to 1 over the profile so the scalar parameter reaches 1.0 at final time

The implementation passed all autograder tests.

---

2. Loop Motion Demonstration

We collected four robot configurations using teach mode:

q_0:
[-0.0819264, -0.13599538, 0.0575315, -2.36497335, 0.0082773, 2.2991974, 0.67502163]

q_1:
[-0.05729219, -0.42088033, 0.09961394, -2.63788948, 0.00826903, 2.2058541, 0.73903088]

q_2:
[-0.20558467, -0.20446108, -0.02575163, -2.63800154, 0.00706992, 2.55150997, 0.69112535]

q_3:
[-0.11125982, -0.3682046, 0.0764298, -2.58438404, 0.00711509, 2.82580872, 0.70104034]

We generated the following trajectories:

- $q_0 \rightarrow q_1$
- $q_1 \rightarrow q_2$

- $q_2 \rightarrow q_3$
- $q_3 \rightarrow q_0$

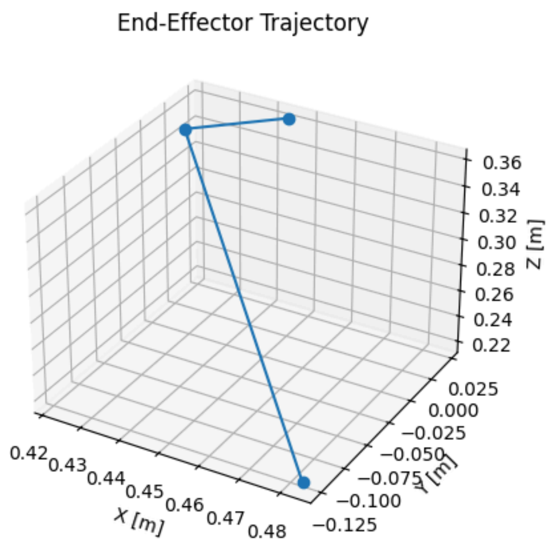
The robot completed the loop smoothly and safely within the required 10-second duration. We noticed that if the velocity max was set too high the robot would overshoot the expected endpoint. This led us to some confusion when we were debugging why our robot ended too far from the end point joint configuration we inputted. We were able to overcome this issue by decreasing the v_max, which prevented the robot from overshooting.

Video link: https://youtube.com/shorts/iV2T7LHsp40?feature=share

We produced 3D Cartesian visualizations of end-effector paths using the forward kinematics routine. For every waypoint in the computed trajectory, we extracted:

- X position
- Y position
- Z position

- A 3D plot (saved as end_effector_trajectory.png) shows the full spatial motion of the robot for debugging and validation.



---

3. Pick-and-Place Demonstration

We completed the pick-and-place task with a flashlight using the same trajectory generator and the Franka gripper.

Recorded configurations:

Home:
[-0.00608366, -0.95394664, -0.03228957, -2.59348415, -0.00837678, 1.59325033, 0.76390718]

Above pick:
[-0.0026026, 0.241709376, -0.001856, -2.22746265, -0.0078128, 2.42024084, 0.784722198]

Pick:
[0.00482188, 0.42789345, -0.00878065, -2.25411856, -0.00964446, 2.63687885, 0.78377835]

Above place:
[0.09792095, 0.315963, 0.17234519, -2.16128569, -0.01907918, 2.47031701, 1.04222218]

Place:
[0.09401129, 0.51291985, 0.15605813, -2.07454963, -0.01909078, 2.47660728, 1.04198222]

Sequence executed:

1. Move to home
2. Move above pick
3. Open gripper
4. Move into pick position
5. Close gripper
6. Lift to above-pick
7. Move above place
8. Lower to place
9. Open gripper

All movements were smooth and collision-free. We noticed that the slower the robot moved the more precise it seemed to be. We especially noticed that with how close to the pick up circle it placed the flashlight back down.

Video link: https://youtube.com/shorts/7aCoAspG-vY?feature=share

---

4. Standby / Recovery Mode

We implemented a robust user-interaction system:

- **Press 'n'** → continue to the next step
- **Press 'r'** → retry the current motion
- **Press 'q'** → safely exit

This ensured repeatability and safe operation, especially helpful for tasks requiring precise object manipulation. This also lets us debug and check for unforeseen issues by letting us retry the current motion. This was useful when we were testing the implementation of the pick and place demonstration, as it helped us determine where our bugs were located. Another example of when this mode was useful is when we were trying to pick up the flashlight, but the gripper was uninitialized. This mode allowed us to try controlling the gripper more efficiently.

Video link: https://youtube.com/shorts/-2-ulqNUB5c?feature=share