Assignment 2 - Franka Panda Whiteboard Cleaning Task

Documentation

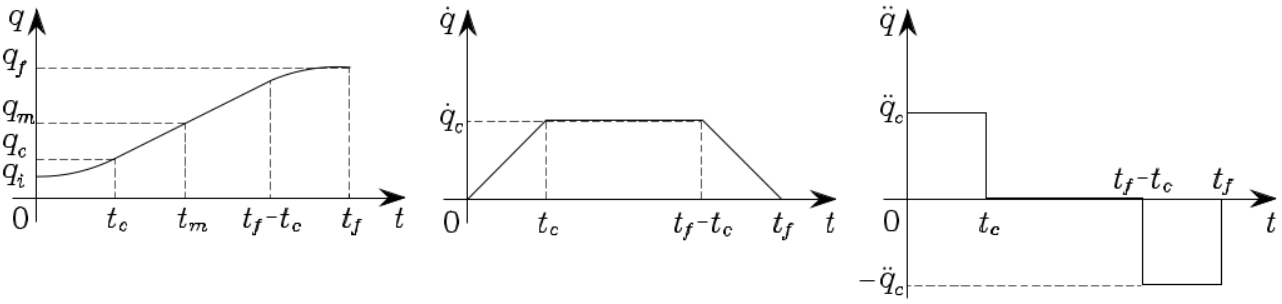
Authors	Dorian Fülöp, Jiabao Song, Adnan Fidan, Rui Tu
Supervisor	Alessandro Melone
Submission Date	Munich, 24.06.2024

Contents

1. Background

The following knowledge was obtained from the book Robotics Modelling, Planning and Control by Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani and Giuseppe Oriolo. We first implement a Trapezoidal Velocity Profile and then use the generated points to define circular and rectilinear trajectories.

1.1. Velocity Profile



Above, we see a Trapezoidal Velocity Profile in terms of position, velocity and acceleration. This defines what speed and acceleration the robotic arm should move with at a specific time during the trajectory motion. To implement this within our code, we started by defining our initial and final positions denoted by q_i and q_f , respectively. For a circular trajectory, we set $q_i = 0$ and $q_f = 2 \cdot \pi \cdot roh$, where roh is the user defined radius of the desired circle. Next, we define the cruise velocity q'_c while making sure that it satisfies the below inequality, where t_f is the user defined final trajectory time.

$$\frac{|q_f - q_i|}{t_f} < |\dot{q}_c| \leq \frac{2|q_f - q_i|}{t_f}$$

We chose the middle value of the inequality for our q'_c. Now, we can define t_c which denotes the start time of our cruise period.

$$t_c = \frac{q_i - q_f + \dot{q}_c t_f}{\dot{q}_c}$$

And finally, we can define our cruise acceleration q''_c from the eq. below.

$$\dot{q}_c = \ddot{q}_c t_c$$

Now that we have defined all the variables that we need, we can extract the position component of our velocity profile using the piecewise function shown below.

$$q(t) = \begin{cases} q_i + \frac{1}{2}\ddot{q}_c t^2 & 0 \leq t \leq t_c \\ q_i + \ddot{q}_c t_c (t - t_c/2) & t_c < t \leq t_f - t_c \\ q_f - \frac{1}{2}\ddot{q}_c (t_f - t)^2 & t_f - t_c < t \leq t_f. \end{cases}$$

Once that is done, extracting the velocity and acceleration components are trivial for a trapezoidal velocity profile as can be seen in the first figure.

For further information refer to the annotated velocity_profile.cpp script.

1.2. Circular Trajectory Generation

The goal is to define a circular trajectory in the base frame of the robot. To do this, we first define the circular trajectory in the whiteboard frame. Here, we take the assumption that the z-axis is unchanged from whiteboard frame to robot base frame. i.e. the whiteboard z-axis is parallel to that of the base frame of the robot and that the surface of the whiteboard is at $z = 0$ in the base frame. The x- and y-axis coordinates of points on the circular trajectory can be computed as shown below.

$$\boldsymbol{p}'(s) = \begin{bmatrix} \rho \cos(s/\rho) \\ \rho \sin(s/\rho) \\ 0 \end{bmatrix}$$

Where s here is the position at a certain time t that we extract from our velocity profile.

We also need a vector c which is the user defined center point of the circle we want to draw in the base frame.

Next, we need to define the rotation matrix R to transform the whiteboard frame trajectory into the base frame. As previously stated, the z-axis in the whiteboard frame remains unchanged so: $z' = z$. The y-axis in the whiteboard frame can be found by finding the normal vector to the z-axis in the whiteboard frame and the -c direction. In other words, we cross-product z' with the norm of -c. So, $y' = z' \times (\|\text{-}c\|)$. The x-axis can then be found by taking the cross-product of our defined y- and z-axis. So, $x' = y' \times z'$. The Rotation matrix can now be found as shown below.

$$\boldsymbol{R} = [\boldsymbol{x}' \quad \boldsymbol{y}' \quad \boldsymbol{z}'];$$

Now we can transform any point of our trajectory from the whiteboard frame to the base frame at any time t during the trajectory using the equation below.

$$\boldsymbol{p}(s) = \boldsymbol{c} + \boldsymbol{R}\boldsymbol{p}'(s)$$

1.3. Rectilinear Trajectory Generation

To define rectilinear trajectories, similarly, we use the position component of our velocity profile s and specify initial and final positional vectors, p_i and p_f in base frame.

Then we apply them to the formula below.

$$\boldsymbol{p}(s) = \boldsymbol{p}_i + \frac{s}{\|\boldsymbol{p}_f - \boldsymbol{p}_i\|}(\boldsymbol{p}_f - \boldsymbol{p}_i)$$

For further information refer to the trajectory_gen.cpp script.

2. Results

For information on how to setup the docker container for reproducing these results, please refer to the README.md file on GitLab

2.1. Process

We developed a state machine composed of circular and rectilinear motions to control the robot arm. At first, we defined the center point of the circle to be a little bit below the whiteboard. With our user defined value for the radius of the circle, roh, we have our touch point on the whiteboard surface. We defined a rectilinear trajectory to this point and, with the force feedback received when the end-effector is in contact with the whiteboard, we stop the rectilinear movement. The external force of the Franka robot was used with a predefined force threshold of 5N. After the robot touches the whiteboard, it will then start a circular trajectory motion. When the cleaning motion is complete, the robot arm will return to its home position with another rectilinear trajectory.

The video of the simulation and the real robot trajectory is available on our GitLab page.

2.2. Effect of the Stiffness

Due to friction, the obtained circle is imperfect. The desired and actual circular trajectory deviate. The error accumulates as can be seen by the spiral trajectory it creates when in contact with the whiteboard. In free air, the trajectory is perfect as shown in our simulation as well as in our testing with real robot. To try and address this issue, different stiffness values were tested to try and force the robot to stick to the desired trajectory. We expected this to be successful as the friction of the whiteboard should not be very high. To compare the results, we recorded the published sensor data from the frankastate topic and following are the generated plots of the 3D coordinates of the end effector.

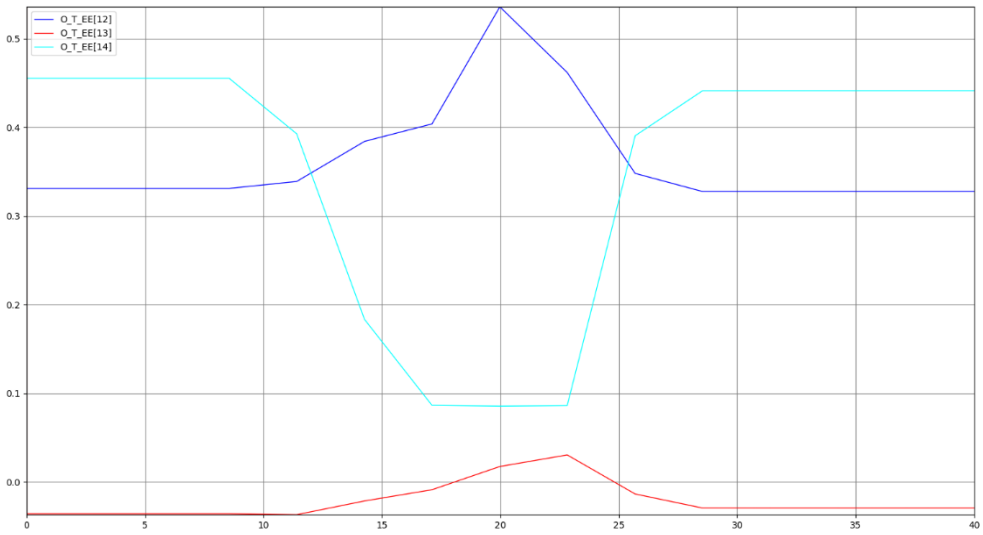


Fig. 1: Results of translation stiffness: 230 N/m

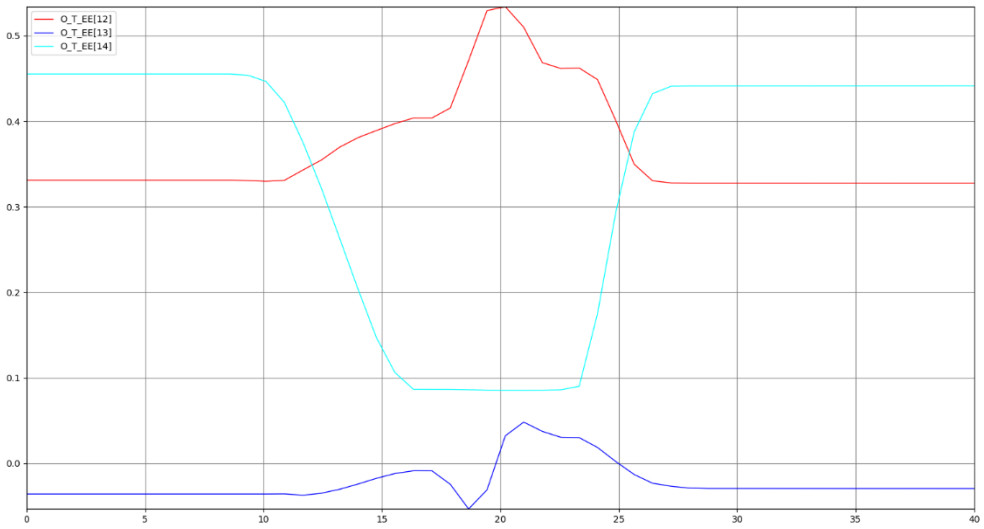


Fig. 2: Results of translation stiffness: 250 N/m

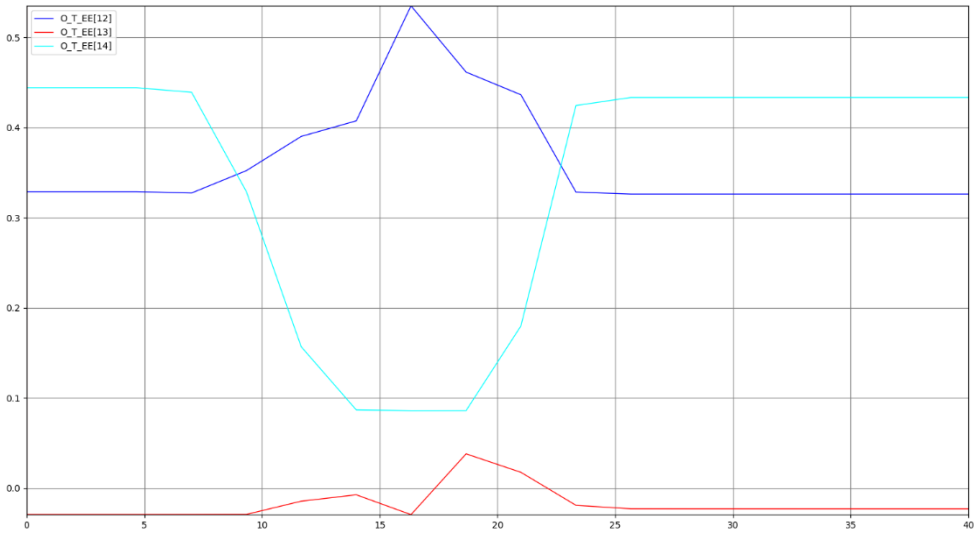


Fig. 3: Results of translation stiffness: 270 N/m

From the plots we can see that, the higher the stiffness, the quicker the robot reacts to try and correct the deviation in trajectory. However, we can see that the movements become more abrupt and possibly start to oscillate as the damping was left unchanged leading to an imbalance. Due to time limitations, we were not able to perfect the circular motion which may be just a matter of increasing the stiffness more and finding the right balance with the damping to make the robot follow the desired trajectory strictly and absorb the disturbances caused by the friction. The best procedure would be to design our own impedance controller and minimize the error between desired and actual trajectory like a PID controller.