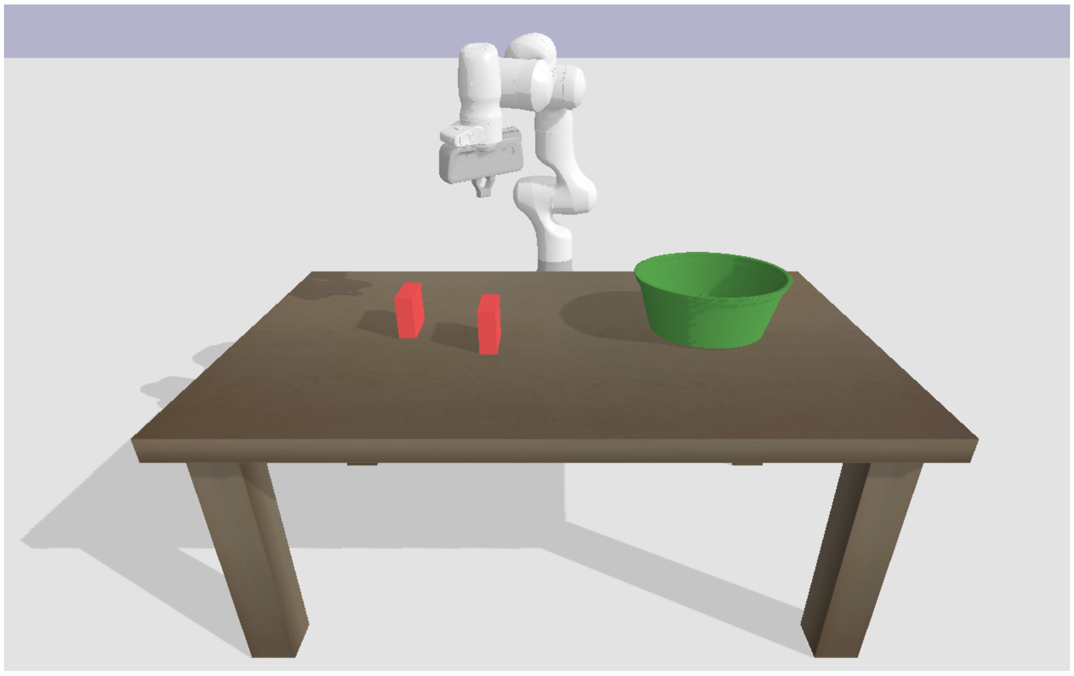

Robotics Manipulation and Locomotion

ROB-2004 Final Project

Kyle Wang

ww2301@nyu.edu

5/9/2023

Project Task

The objective of this project is to implement a controller that allows the robot to pick up the two red blocks on the table and drop them in the green bowl. The initial positions of the objects on the table are shown in the figure below.

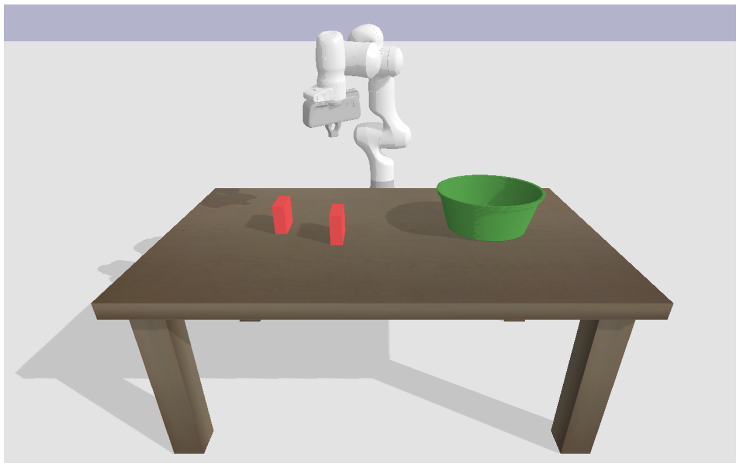

Figure 1. Initial placement of the blocks and bowl on the table

The type of controller used to control the robot is not restricted; however, at least one controller has to be implemented in the end-effector space. The robot used in this project is the Frank-Emika Panda robot. It has 7 revolute joints, and its kinematics are shown in the figure below.

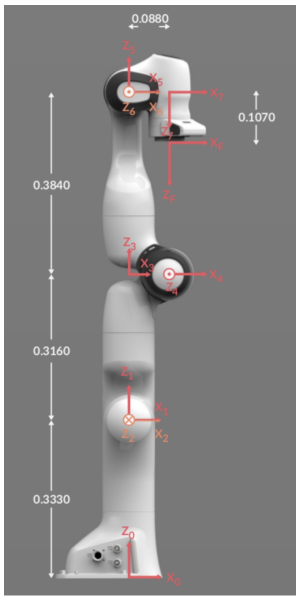

Figure 2. Frank-Emika Panda robot kinematics

Methodology

The controller employed in this experiment is the resolve rate motion controller. Unlike the experiment done in lab 4 where most of the joint computations and trajectory generations are calculated in the joint space, resolve rate control focuses more in the end effector space where the velocity of the end effector is computed based on its position in the workspace. Thus, all joint velocities of the robot are calculated based on the desired positions and velocities of the end effector in this project.

The inverse Jacobian matrix is used to help convert the desired velocities and acceleration of the end effector into joint velocities of the robot. The inverse Jacobian matrix is a mathematical construct that describes the joints' relations and robot's kinematics with respect to a certain frame. This frame can be either the end effector frame or spatial frame. The inverse Jacobian matrix used in this project will be working in the end-effector frame but oriented like the spatial frame. In doing so, we ensure the gripper is always oriented in the same direction even while the body of the robot may be moving or oriented in a different direction.

The reason why the resolve rate motion controller is employed is because it produces a higher degree of accuracy and precision compared to joint space generation. In addition, the inverse kinematic function used in joint space generation can become really complicated and hard to derive in this project because we are using a robot with 7 revolute joints. The inverse kinematic function has already become relatively complex in lab 3 when the robot with 2 revolute joints is used. With a robot that has 7 revolute joints, the equations for inverse kinematics will be almost impossible to derive as we need to consider so many different possibilities. However, the resolve rate motion controller comes with some limitations. For instance, the resolve rate controller may not be ideal for tasks that require the robot to apply a specific force or interact with its environment in a more complex way. For tasks like these, an impedance controller might be a better option as it allows robots to behave like a spring.

As for the path taken by the robot to complete the task, the robot navigates to a position above block 1 first from its initial position and slowly descends to where block 1 is on table level. The robot grip block 1 using its gripper and ascends back to a position above the block's initial position. The robot moves to a position above the bowl and then drops the block into the bowl. The same concept applies to block 2. The robot moves from the top of the bowl to a position above block 2 and then descends to where block 2 is. The robot grips block 2 and ascends back to a position above the block's initial position. The robot then moves to a position above the bowl and drops the block. Here block 1 refers to the block on the right and block 2 refers to the block on the left when viewed from the base frame. The trajectory taken by the robot in the simulation is illustrated in Figure 3 in the Graphs and Plots section.

Algorithm

Using the code skeleton provided and a few additional helper functions, the control algorithm is completed. The for loop in the code skeleton is where all the end effector and joint velocities and positions are calculated. There are eight if-else statements in the for loop, and each of them determines the movement of the robot at a specific time frame. For instance, the first if statement is executed between 0 to 3 seconds, and it moves the robot from its initial position to a position above block 1. While the second if-else statement is executed between from 3 to 6 seconds, and it moves the robot down to where block 1 is, preparing the robot for gripping the block. Inside each if-else statement consists a function call to the function compute_trajectory. The input arguments of this function are starting position (start_pos), desired ending position (end_pos), initial time (t_init), final time (t_final), and current time (time[i]). After receiving the inputs, the function compute_trajectory uses a 5th degree polynomial function to compute the desired position and velocity. By implementing a 5th order polynomial, the function establishes 6 different constraints in controlling the robot. The 6 constraints are described as follows:

$$s(t_{init}) = 0, \ s(t_{goal}) = 1$$

$$s_{dot}(t_{init}) = 0, \ s_{dot}(t_{goal}) = 0$$

$$s_{dot-dot}(t_{init}) = 0, \ s_{dot-dot}(t_{goal}) = 0$$

As for the 5 degree polynomials used for computing desired positions, velocities, and acceleration, they are defined as follows:

$$\theta_{des}(t) = \theta_{init} + \left(\frac{10}{T^3}(t - t_{init})^3 + \frac{-15}{T^4}(t - t_{init})^4 + \frac{6}{T^5}(t - t_{init})^5\right) * (\theta_{goal} - \theta_{init})$$

$$\theta_{dot_{des}}(t) = \left(\frac{30}{T^2}(t - t_{init})^2 + \frac{-60}{T^4}(t - t_{init})^3 + \frac{30}{T^5}(t - t_{init})^4\right) * (\theta_{goal} - \theta_{init})$$

$$\theta_{dot-dot_{des}}(t) = \left(\frac{60}{T^3}(t - t_{init}) + \frac{-180}{T^4}(t - t_{init})^2 + \frac{120}{T^5}(t - t_{init})^3\right) * (\theta_{goal} - \theta_{init})$$

Having a 5th order polynomial with 6 constraints to compute the end effector's desired positions and velocities greatly improves the stability and accuracy of the controller. This is because we ensure both velocity and acceleration are zero in the starting and ending positions. If linear time interpolation in which only 2 constraints are established is used, constant desired velocity and acceleration will be created, and this can pose potential problems since the robot starts from rest, and we want velocity and acceleration to be zero initially.

After computing the desired end effector velocity, it is mapped to joint velocities using an inverse Jacobian matrix. The Jacobian matrix is first computed using the given get_Jacobian function where current positions of the joints are passed as an argument. The returned Jacobian matrix is then processed by extracting only the linear part of the Jacobian and stored in a variable named J_linear. J_linear is then passed into a function called inverse_jacobian to find the inverse Jacobian matrix. In inverse_jacobian, the Jacobian matrix is first decomposed to compute the singular values of the robot. These values are measures of the sensitivity of the robot's end effector to changes in joint velocities. When one or more singular values approach zero, the robot is approaching singularity. Singularity happens when the robot loses one or more degrees of freedom. It is crucial that singularity is avoided because it can cause instability and potentially damage the robot. When potential singularity is detected, a damping constant is added to the Jacobian matrix to prevent singularity. The damped Jacobian matrix is then passed into the numpy pseudo inverse function to compute the inverse Jacobian matrix.

With inverse Jacobian matrix computed, the desired joint velocity is calculated by finding the product of the inverse Jacobian matrix and desired end effector velocity. Velocity error is then found by subtracting desired joint velocity from current joint velocity. The last step of the controller is to compute joint torque, where D gains are multiplied with velocity error and added to the returned values from function g. Function g is employed to compensate for the effects caused by gravity. The computed joint torque is then sent to the robot to perform movement in the simulation.

Graphs and Plots

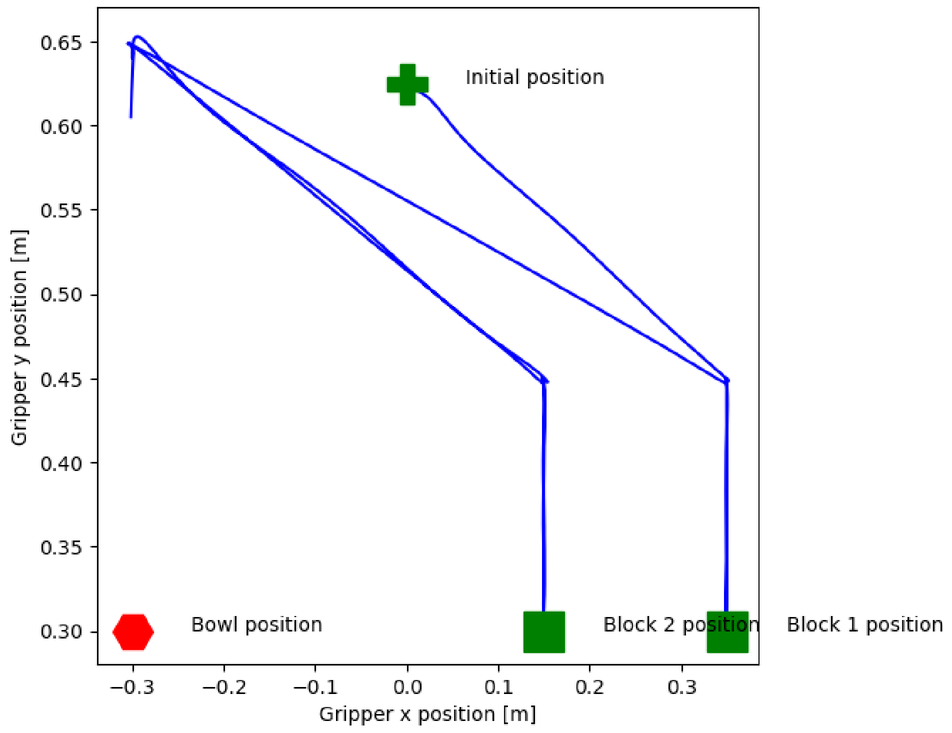

Figure 3. Trajectory of the end effector (view from the base frame)

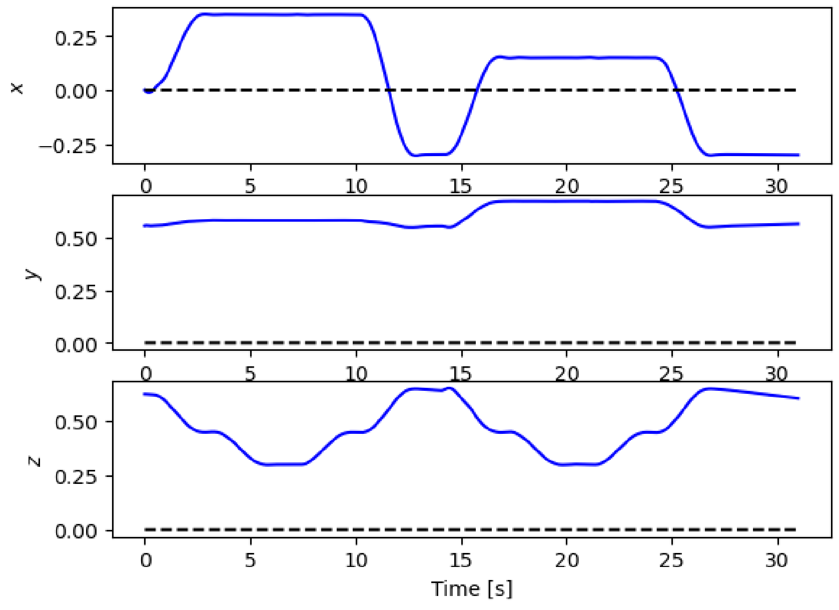

Figure 4. Positions of the end effector in x, y, and z directions

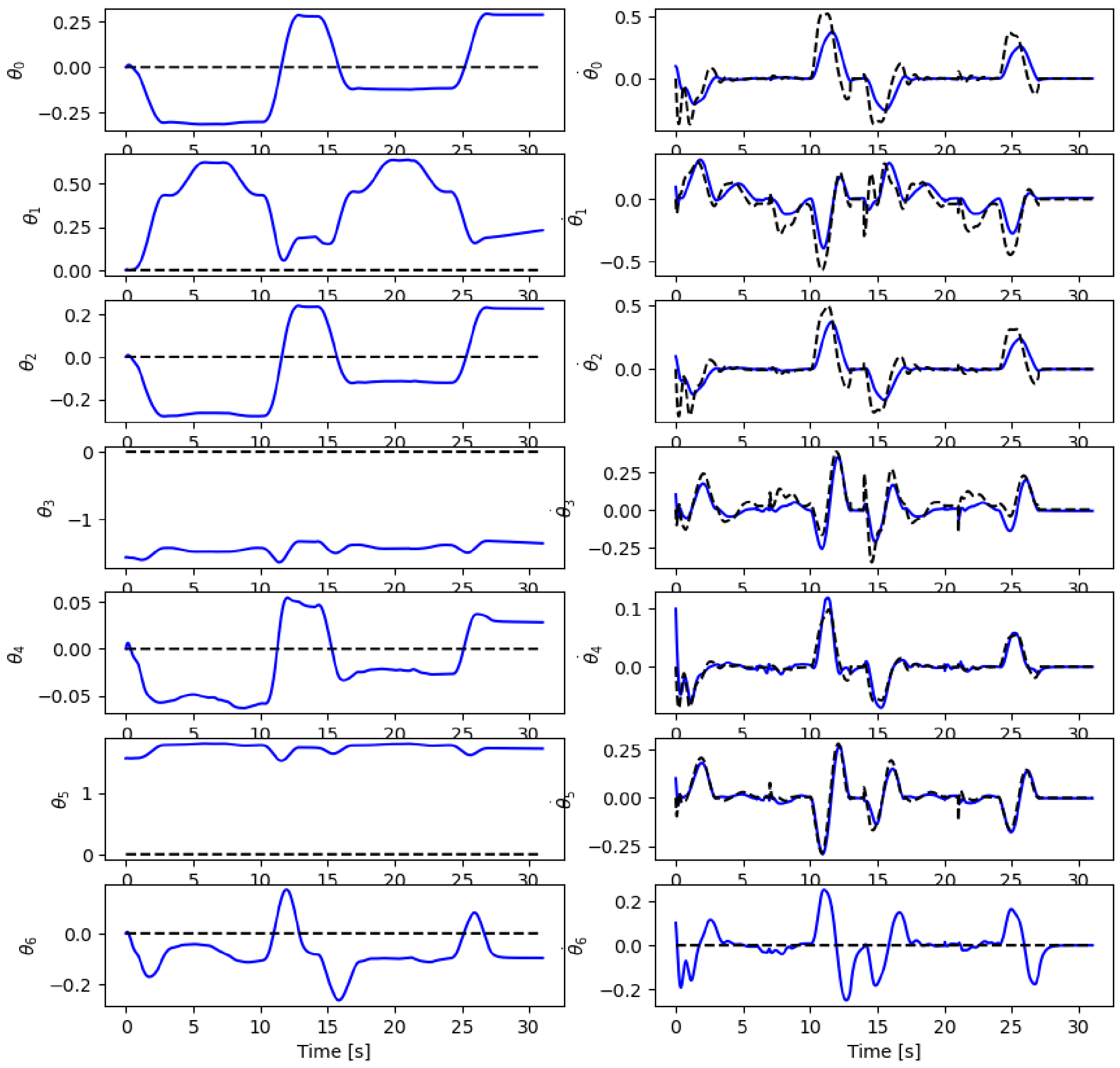

Figure 5. Joint positions and velocities