EE368 Project

Dynamics

Student: Fu Xuanzhu, Xie haotian, Gan yuhao, 12212807, 12212806, 12211629
Lecturer: Meng qinghu, mengqh@sustech.edu.cn

Main Part One

Reference Material:

1. *Chapter 6 of Introduction to Robotics*

Task:

1. Add comments to function get_torque in *Dynamics.py*
2. Move the manipulator and draw the curves of the real torques and the calculated torques for each joint. Describe the curve. (hint: use rqt_plot)

Submission:

Submit code with comments and report.

Problem 1: Function *get_torgue* with comments

1. The code with comment is showed below:

```
def get_torque(self, thetas, thetas_d, theta_dd, f_ext, n_ext):
        # Calculate joint torques based on joint angles,
        velocities, accelerations,
        # and consider external forces and moments

        f_ext = np.array(f_ext).T
        # Convert external forces to column vectors
        n_ext = np.array(n_ext).T
        # Convert external moments to column vectors

        link_num = len(self.link_list)  # Number of links in the robot arm

        # Initialize matrices to store rotation matrices, position vectors,
        and center of mass positions for each link
        R_i_iplus1_list = np.zeros((3, 3, link_num))
        P_i_iplus1_list = np.zeros((3, link_num))
        P_i_c_list = np.zeros((3, link_num + 1))

        # Compute transformation matrices and related data for each link
        for i in range(link_num):
            T_i_iplus1 = self.link_list[i].transformation_matrix(thetas[i])
            R_i_iplus1_list[:, :, i] = T_i_iplus1[:3, :3]
            P_i_iplus1_list[:, i] = T_i_iplus1[:3, 3]
            P_i_c_list[:, i + 1] = self.link_list[i].center
```

```python

        # Initialize angular velocity, angular acceleration,
        linear acceleration, etc.
        omega = np.zeros((3, link_num + 1))
        omega_d = np.zeros((3, link_num + 1))
        v_dot_i = np.zeros((3, link_num + 1))
        v_dot_c = np.zeros((3, link_num + 1))
        v_dot_i[:, 0] = [0, 0, 9.8]  # Set gravity acceleration in the base
        coordinate system
        F = np.zeros((3, link_num + 1))
        N = np.zeros((3, link_num + 1))

        # Forward recursion to compute velocities, accelerations, forces,
        and moments for each link
        for i in range(link_num):
            R = R_i_iplus1_list[:, :, i].T
            m = self.link_list[i].mass
            P_i_iplus1 = P_i_iplus1_list[:, i]
            P_iplus1_c = P_i_c_list[:, i + 1]
            I_iplus1 = self.link_list[i].inertia_tensor
            theta_dot_z = thetas_d[i] * np.array([0, 0, 1]).T
            omega[:, i + 1] = R.dot(omega[:, i]) + theta_dot_z
            omega_d[:, i + 1] = R.dot(omega_d[:, i])
            + np.cross(R.dot(omega[:, i]), theta_dot_z)
            + theta_dd[i] * np.array([0, 0, 1]).T
            v_dot_i[:, i + 1] = R.dot(np.cross(omega_d[:, i], P_i_iplus1) +
            np.cross(omega_d[:, i], np.cross(omega_d[:, i], P_i_iplus1)) +

            v_dot_i[:, i])
            v_dot_c[:, i + 1] = np.cross(omega_d[:, i + 1], P_iplus1_c)
            + np.cross(omega[:, i + 1], np.cross(omega[:, i + 1], P_iplus1_c))

            + v_dot_i[:, i + 1]
            F[:, i + 1] = m * v_dot_c[:, i + 1]
            N[:, i + 1] = I_iplus1.dot(omega_d[:, i + 1]) +
            np.cross(omega[:, i + 1], I_iplus1.dot(omega[:, i + 1]))

        # Initialize storage arrays for forces, moments, and joint torques
        f = np.zeros((3, link_num + 1))
        n = np.zeros((3, link_num + 1))
        tau = np.zeros(link_num + 1)

        # Backward recursion to compute joint torques
        for i in range(link_num, 0, -1):
            R = T_i_iplus1[:3, :3]
            if i == link_num:
                f[:, i] = f_ext + F[:, i]
                n[:, i] = N[:, i] + n_ext
                + np.cross(P_i_c_list[:, i], F[:, i])
                tau[i] = n[:, i].T.dot(np.array([0, 0, 1]).T)
            else:
                R = R_i_iplus1_list[:, :, i]
```

```
                f[:, i] = R.dot(f[:, i + 1]) + F[:, i]
                n[:, i] = N[:, i] + R.dot(n[:, i + 1]) +
                np.cross(P_i_c_list[:, i], F[:, i])

                + np.cross(P_i_iplus1_list[:, i], R.dot(f[:, i + 1]))
                tau[i] = n[:, i].T.dot(np.array([0, 0, 1]).T)

        return tau[1:]  # Return the calculated joint torques
        # (excluding the torque of the base coordinate system)
```

Problem 2: Move the manipulator and draw the curves of the real torques and the calculated torques for each joint. Describe the curve (hint: use *rqt_plot*)

1. **The curves of the real torques and the calculated torques for each joint**

   Figures are showed at the *Figure Attachment* Part.

2. **Describe the curve**

   (a) The trend is consistent, but there is a time lag.

   Phenomenon: The simulation curve (in red) has a waveform shape that is roughly similar to the actual curve (in blue), but the peak appears with a delay of approximately 0.1 to 0.3 seconds.

   Note: The dynamic model is correct (the approximate acceleration and motion trend are consistent). However, there is a delay in information processing or a lag in state estimation.

   Analysis: Acceleration estimation uses first-order differencing, which is prone to lag.

   (b) The amplitude of the simulated torque is significantly smaller.

   Phenomenon: The absolute value of the red curve is significantly lower than that of the blue curve.

   Note: The calculated joint acceleration or inertia term is not strong enough, or some essential load/friction compensation components may have been omitted.

   Analysis: Not conclude friction or use smaller acceleration comparing to the real case for larger torque to compensate error.

   (c) The Anomalous phenomenon of joint3

   Phenomenon: The trend of real and calculated joints torque is opposite.

   Analysis: Inconsistent definition of coordinate system or joint axis directions.

Figure Attachment

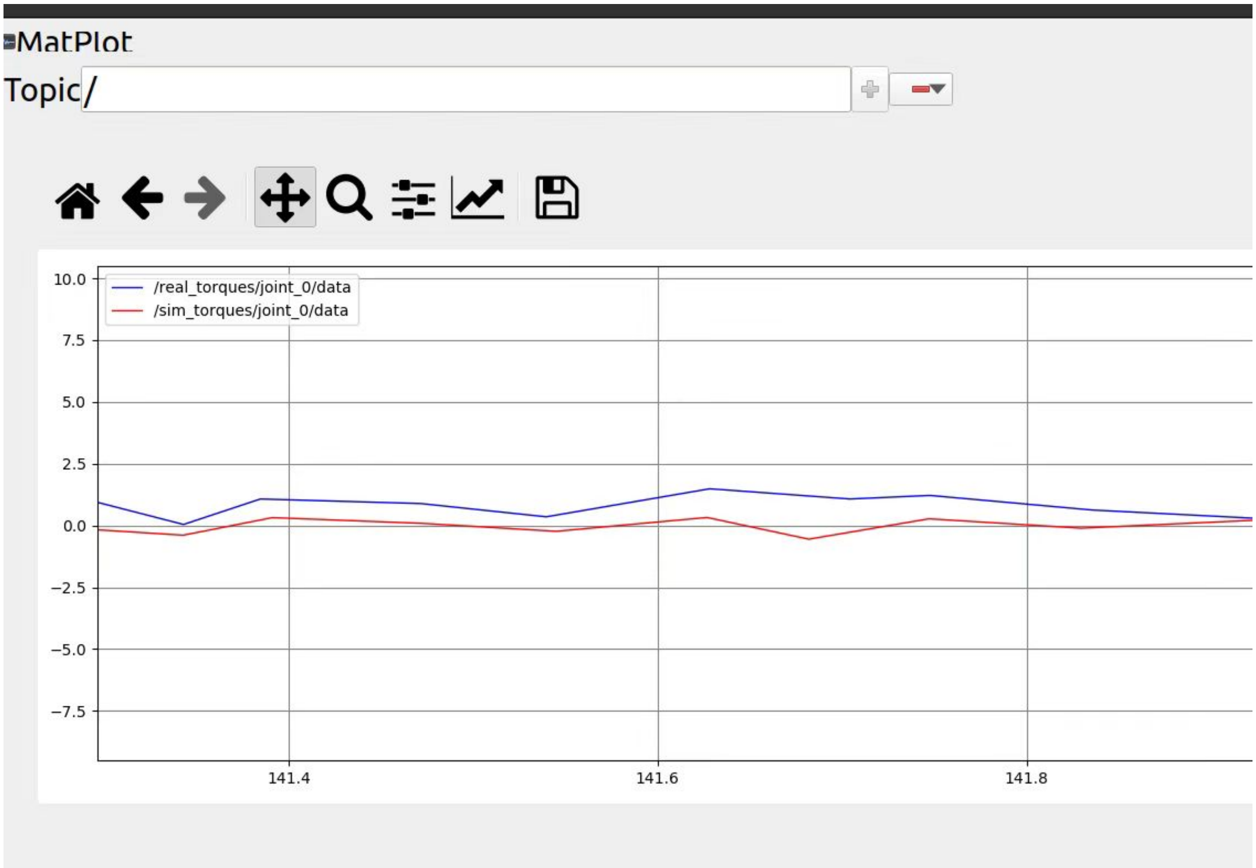

Figure 1: The real torque curve and calculated torque curve of joint0

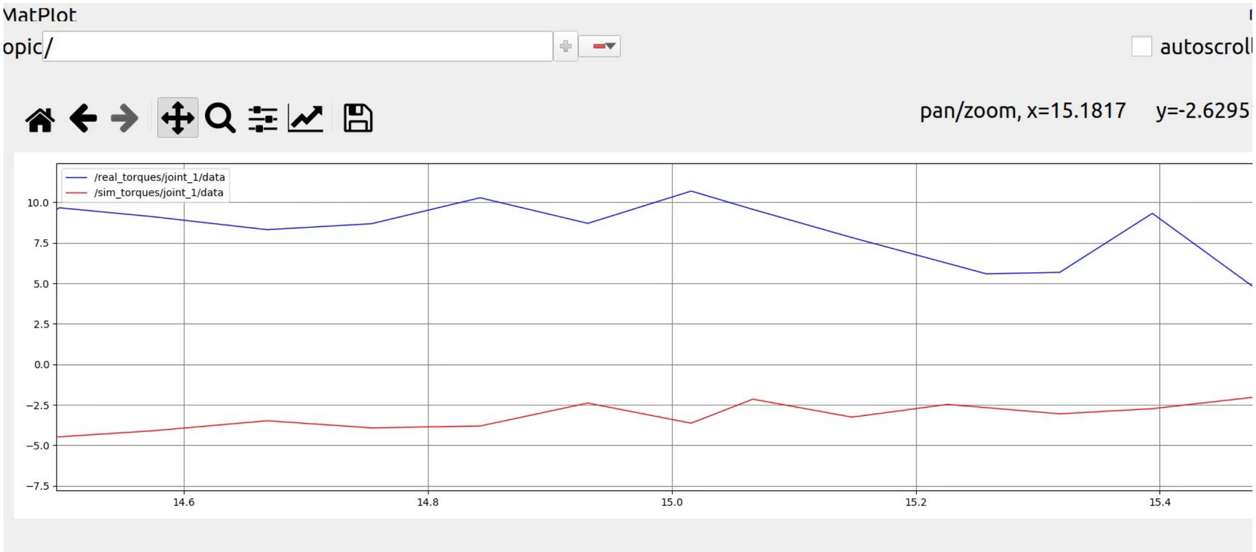

Figure 2: The real torque curve and calculated torque curve of joint1

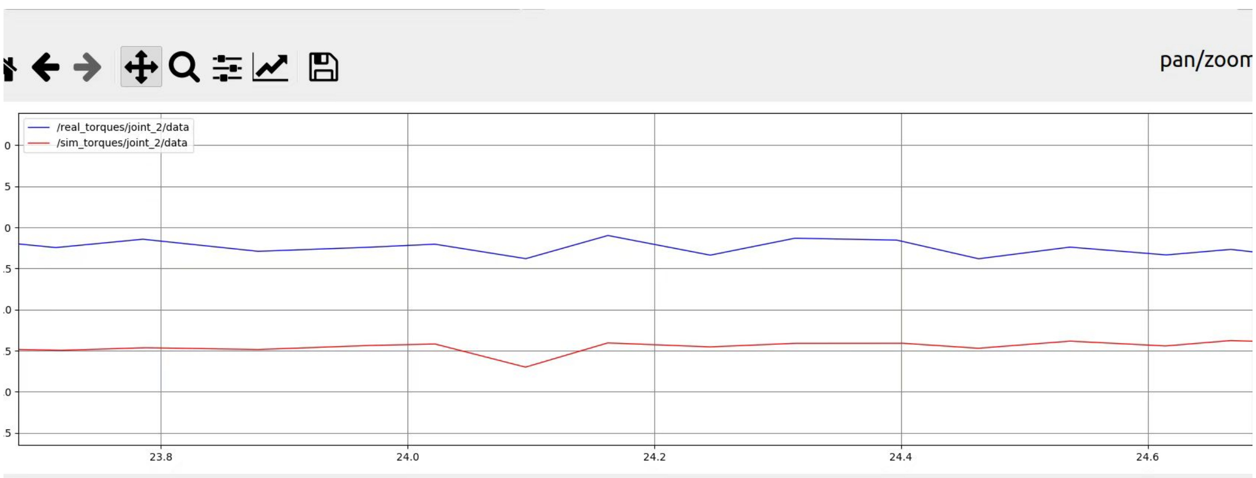

Figure 3: The real torque curve and calculated torque curve of joint2

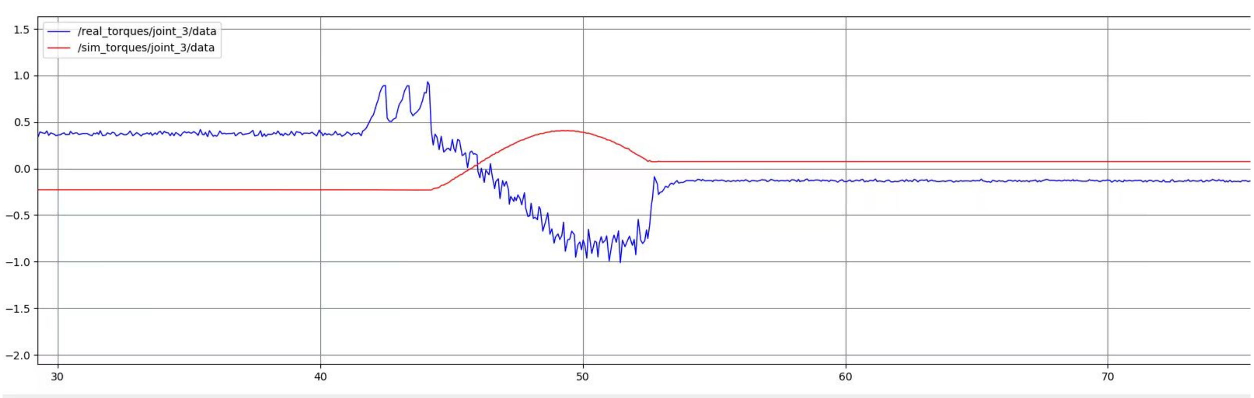

Figure 4: The real torque curve and calculated torque curve of joint3

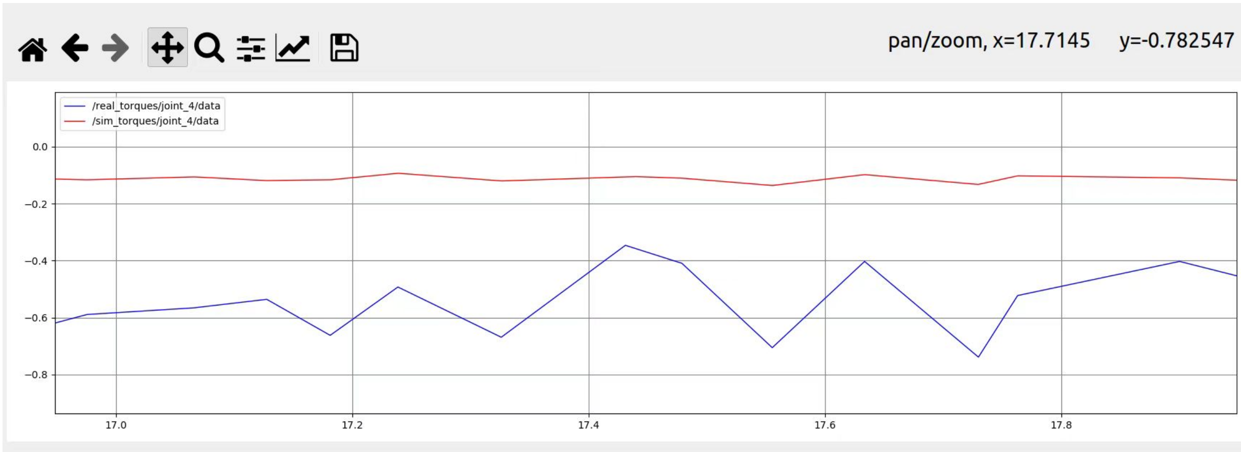

Figure 5: The real torque curve and calculated torque curve of joint4

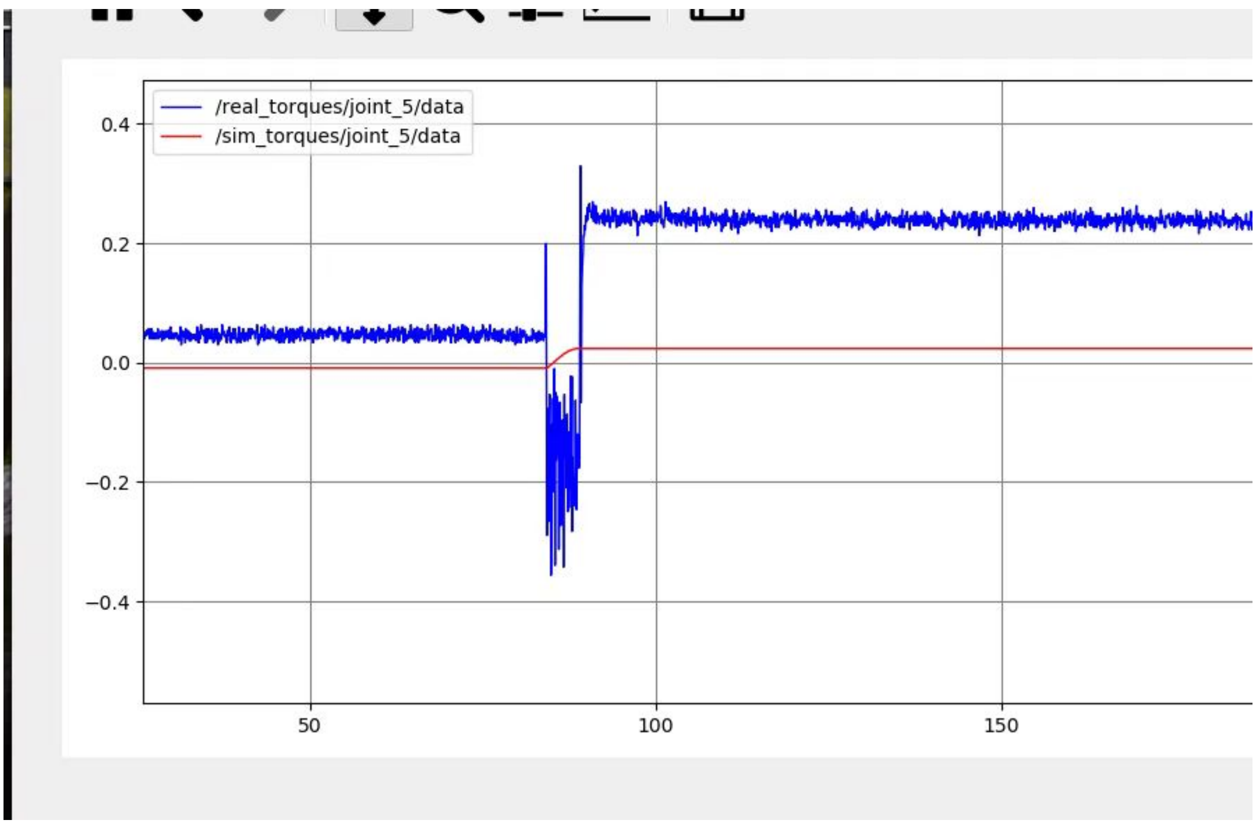

Figure 6: The real torque curve and calculated torque curve of joint5