ME 655
Wearable Robotics and Sensors

Fall 2023

Homework Assignment 2

This report has been prepared by:
Jnana Teja Pasagadugula (20018097).

Abstract:

In this report, the control for an RP planar manipulator to follow a user-specific straight-line trajectory using MATLAB and Simulink is explained. The given robot has a revolute and a prismatic joint, and joint torques and forces are given, The center of masses of the links and their locations are given, the objective is to have the end effector track a specific user-defined trajectory under non-linear control dynamics model.

Cartesian Space Trajectory Generation:

The end-effector follows a straight-line trajectory from p1 to p2, following a 5th-order polynomial in the cartesian space, the fifth-order polynomial is:

$$S(t) = a_0 + a_1 \cdot t + a_2 \cdot t^2 + a_3 \cdot t^3 + a_4 \cdot t^4 + a_5 \cdot t^5$$

The coefficients of the equation are determined by the starting and final conditions.
Assuming null velocity and acceleration at t0=0 and tf, s(t) must satisfy the following boundary conditions:

S(0) = 0, S'(0) = 0, S''(0) = 0.
S(tf) = 1, S'(tf) = 0, S''(0) = 0.

So, the coefficients for the fifth-order polynomial will be:

$a_0 = 0,\ a_1 = 0,\ a_2 = 0,$

The remaining coefficients can be extracted by solving the equation:

```
rhs = [1; 0; 0];
lhs = [t_f^3, t_f^4, t_f^5;
3*t_f^2, 4*t_f^3, 5*t_f^4;
6*t_f, 12*t_f^2, 20*t_f^3];
coefficients = lhs \ rhs;
a3 = coefficients(1,1)
a4 = coefficients(2,1)
a5 = coefficients(3,1)
```

This equation is integrated into the Cartesian trajectory generator simulink block, the block gives the current position, velocity and acceleration of the robot with respect to the current time.
The P1 given to the robot is (4,4), and the P2 given to the robot is (3,1).

Trajectory Conversion:

The forward kinematics of the robot are:

$X = (d_2 + l_c) \cdot cos(\theta_1)$

$Y = (d_2 + l_c) \cdot sin(\theta_1)$

And the inverse kinematics of the robot, which can be obtained by manipulating the forward kinematics equations are:

$\theta_1 = tan^{-1}(y/x).$

$d_2 = \sqrt{x^2 + y^2} - l_c.$

These equations are integrated into the Trajectory conversion block in the Simulink and it takes the current positions in the cartesian space as the input to calculate the join space, and they can be used to calculate the inverse dynamics of the robot.

The dynamics equations of the robot are:

$\tau_1 = (m_1 l_1^2 + I_{zz1} + I_{zz2} + m_2 d_2^2) \cdot \theta_1'' + 2 m_2 d_2 \theta_1' d_2'$

$\tau_2 = m_2 d_2'' - m_2 d_2 \theta_1'^2$

Where τ_1 is the torque input at the proximal joint and τ_2 is the force input at the proximal joint.

The inverse dynamics of the equations are

$\Theta_1' = J^{-1}(\Theta) X_d'$

$\Theta_1'' = \delta J^{-1}(\Theta) X_d' + J^{-1}(\Theta) X_d''$

We need the inverse Jacobian matrix and the derivative of the inverse jacobian matrix to find the inverse dynamics of the system.

The Jacobian matrix can be found by partially differentiating the forward kinematics equations with the joint space variables.

The Jacobian matrix is

$$J(q) = \begin{bmatrix} -(d_2 + l_c).\sin(\theta_1) & \cos(\theta_1) \\ (d_2 + l_c).\cos(\theta_1) & \sin(\theta_1) \end{bmatrix};$$

And the Jacobian inverse matrix is:

$$J^{-1}(q) = \begin{bmatrix} (-\sin(\theta_1))/(d_2 + l_c) & (\cos(\theta_1))/(d_2 + l_c) \\ \cos(\theta_1) & \sin(\theta_1) \end{bmatrix};$$

And the derivative of the inverse of the Jacobian is:

$$\delta J^{-1} = \begin{matrix} [\dot{d_2}(sin(\theta_1))/(d_2 + l_c)^2 - \dot{\theta_1}(cos(\theta_1))/(d_2 + l_c) & - (cos(\theta_1)/(d_2 + l_c)^2)\dot{d_2}; \\ (sin(\theta_1))\,\dot{\theta_1} & (cos(\theta_1))\dot{\theta_1}]; \end{matrix}$$

And with these dynamics equations, we can find the desired position, velocity, and acceleration of the joints of the robot.

Trajectory Tracking:

The robot uses an asymptotically stable state-feedback control law that allows the closed-loop system to track an arbitrary trajectory,
The state space equations can be found using the force equations.

$$\tau_1 = (m_1 l_1^2 + I_{zz1} + I_{zz2} + m_2 d_2^2) \cdot \ddot{\theta_1} + 2m_2 d_2 \dot{\theta_1} \dot{d_2}$$

$$\tau_2 = m_2 \ddot{d_2} - m_2 d_2 \dot{\theta_1}^2$$

The state space equations are:

$$\dot{X} = AX + Bu$$

$$x_1 = \theta_1$$

$$x_2 = d_2$$

$$x_3 = \dot{x_1} = \dot{\theta_1}$$

$$x_4 = \dot{x_2} = \dot{d_2}$$

By solving the torque equations, we get the state space equations as,

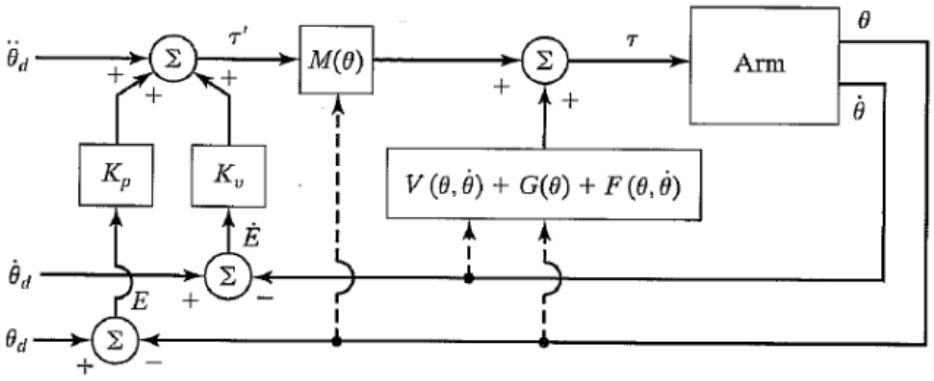
$$\dot{x_3} = 3\tau_1/(4(m_2 x_2^2 + m_1 l_1^2)) - 3x_2 x_4 m_2 x_3/(2(m_2 x_2^2 + m_1 l_1^2))$$

$$\dot{x_4} = x_2 x_3^2 + \tau_2/m_2$$

The controller design is as follows

Use **Computed Torque Method**

$$\tau = M(\Theta)\ddot{\Theta} + V(\Theta, \dot{\Theta}) + G(\Theta) + F(\Theta, \dot{\Theta})$$



The mass matrix M is given by:

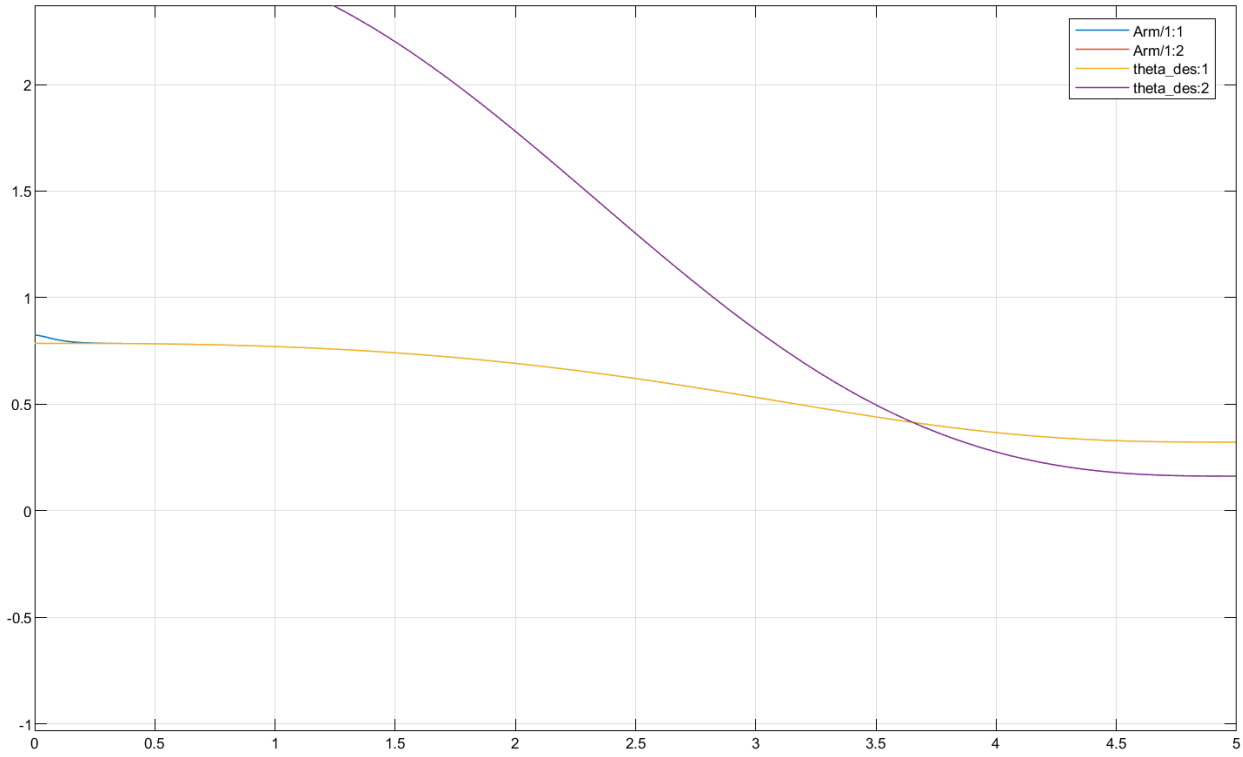
$$M = [m_1 l_1^2 + \tfrac{1}{3}.m_1 l_1^2 + \tfrac{1}{3} m_1 d_2^2 + m_2 d_2^2 \quad 0;$$
$$0 \quad m_2];$$

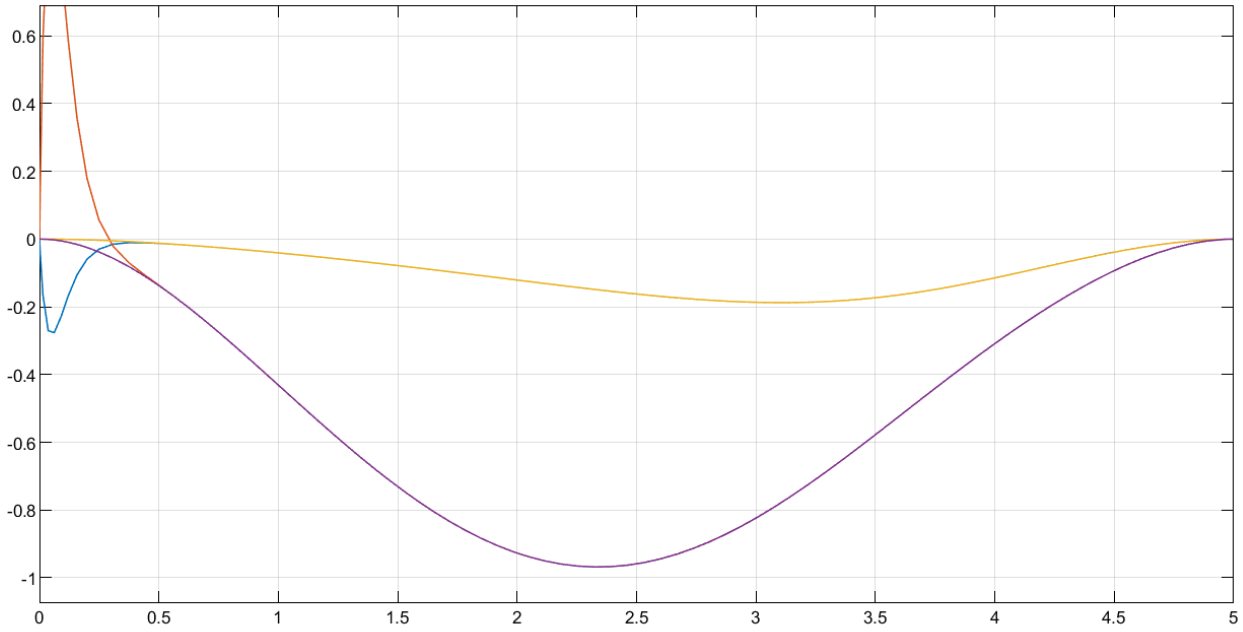
And the velocity matrix V is given by:

$$V = [2m_2 d_2 \dot{\theta}_1 \dot{d}_2;$$
$$-m_2 d_2 \dot{\theta}_1^2];$$

Kp and Kv can be calculated by ω_n, and for this lab, ω_n is taken as 20.

The output of the system after running is:





We can see there's an error in the beginning as we implement little error in the beginning to see whether our system is working or not, and gradually the error converges.