MAE 547 Final Project Report

May 2, 2024

Arizona State University
School for Engineering of Matter, Transportation and Energy

Eric Weissman
(eweissm1@asu.edu)

Timothy Nieves
(tanieves@asu.edu)

Wen Yang
(wyang115@asu.edu)

Willem Grier
(wgrier@asu.edu)

Emad Anvar Siddikui
(esiddiku@asu.edu)

GitHub Repository

This page has been left blank intentionally.

Table of Contents

1 - Introduction

The target of the project is to design a robotics toolbox to explore kinematics and dynamics control challenges in robotic manipulation tasks. It comprises several key modules: a primary GUI window, robot dynamics, and indirect force control mechanisms such as compliance control and impedance control. Leveraging the entirety of knowledge learned from the 547 courses, this project transforms theoretical insights into practical tools aimed at tackling real-world robotic manipulation complexities. Through thorough analysis and debugging, we ensure the functionality and reliability of the toolbox in the simulation environment.

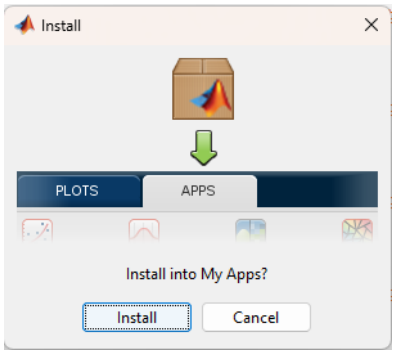
2 - Getting Started

2.1 - Prerequisites

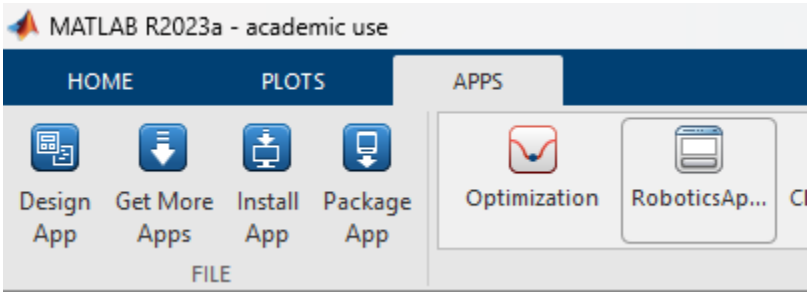
1. MATLAB >= R2023a
   (the following are installed with our RoboticsAppGUI Package)
2. App Designer Migration Tool for MATLAB.
3. Optimization Toolbox
4. Polynomialspirals by Peter Corke
5. Robotics Toolbox by Peter Corke
6. Simulink
7. Symbolic Math Toolbox
8. Triple Angle Visualizer Toolbox
9. Control System Toolbox
10. Mapping Toolbox

2.2 - How to Install the Package

1. Run `RoboticsAppGUIV1.mlappinstall`. The required libraries listed above will be installed automatically in MATLAB. Files in the `src` directory are for viewing only.



2. Execute the installed app “RoboticsAppGUI”.



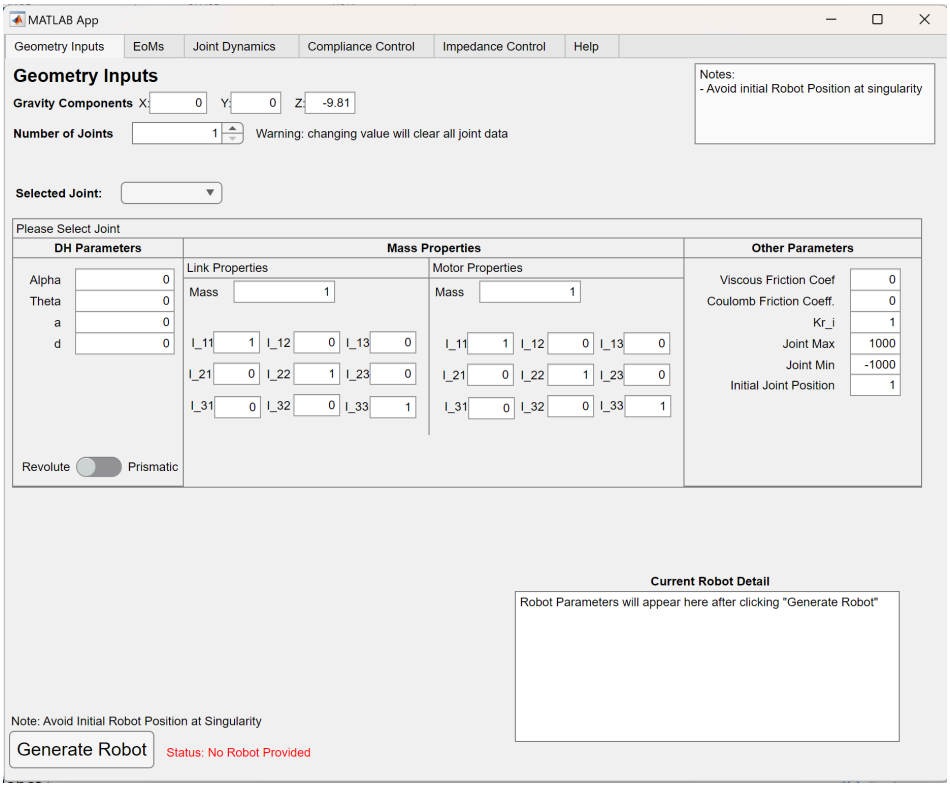


Figure 1: RoboticsAppGUI user view after completing section 2.2

2.3 - How to Use the GUI

1. In the Geometry Inputs page:
   Set the basic configuration for a robot arm, including the number of joints, the DH parameters for each joint, the mass properties such as mass and inertia matrix (defined in link frame), and other parameters like friction coefficient, gear ratio, etc. The initial joint position is in radians for revolute joints. Please note that the initial robot position should not be at a singularity. Then click 'Generate Robot' to create the robot object. The robot is assumed to have homogeneous straight links.
2. Keep the robot figure window open throughout use to enhance your experience and avoid non-critical error messages.
3. Turn to the EoMs page and click 'Generate Analytical Equations of Motion' to get the result.
4. Turn to the Joint Dynamics page, set the desired trajectory by inputting an explicit formula and enable the options. The path is defined as ZYZ euler angles in the base frame. For time-dependent functions, your input must adhere to MATLAB's str2sym documentation. After clicking the 'Generate Joint Dynamics' button, the corresponding plot of the joint state will be displayed on the right side.
5. Turn to the Compliance control page and set the end effector force, click the 'Run Compliance Control' button to display the result. One can also tune the gain matrix (Kp, Kd, etc.) to check the control performance.
6. Turn to the Impedance control page and set the desired trajectory in operational space and enable the options, click the 'Run Impedance Control' button to display the result. One can also tune the gain matrix (Kp, Kd, etc) to check the control performance. (Note that all the trajectory needs to be set within the workspace of the created manipulator!)

3 - The Interactive RoboticsAppGUI

The RoboticsAppGUI we designed aims to provide a convenient and efficient interactive interface, enabling users to quickly simulate and test the dynamics of robotic arms through the parameters and module interfaces provided in the GUI. The GUI assumes that users have a basic level of proficiency in robot modeling,

control, and planning (for example, generating corresponding robotic arms by inputting the DH parameter table directly rather than using other simpler descriptions). Overall, the GUI consists of five modules: geometry input, equations of motion, joint dynamics, compliance control, and impedance control. All technical functions are invoked and linked through the App Designer.

The operational logic of this GUI is as follows: first, set the basic configurations of the robotic arm in the geometry input page and click the button to create the robot. As a result, a 3D plot window will pop up to display the basic configuration of the created manipulator. After successfully creating the robotic arm, users can then proceed to the other four interfaces to conduct related simulation experiments.

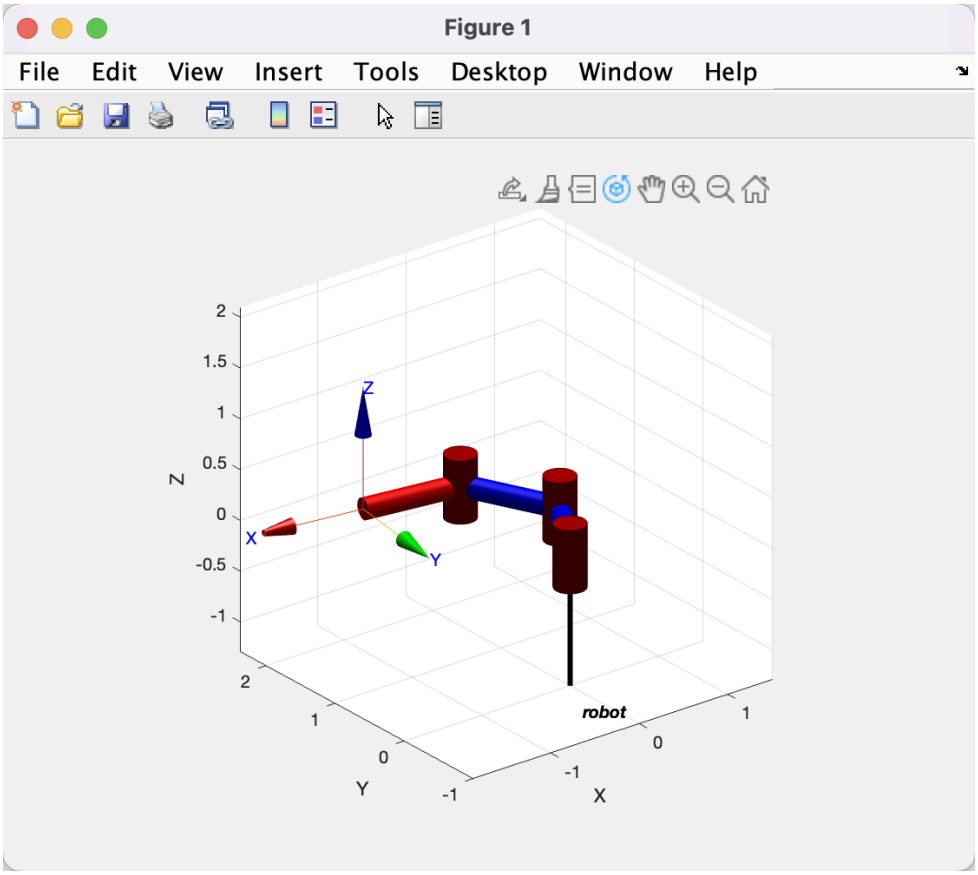


Figure 2: Generated Robotics Manipulator

3.1 - Forward Kinematics

A manipulator is a kinematic chain of rigid bodies (links) connected by actuated joints. The target of the forward kinematics of a manipulator is to derive the relationship between the joint states and the end-effector state. We use Peter Corke's Robotics Toolbox to construct a `SerialLink` object and plot the robot in the GUI. This robot is used throughout the application to plot results.

The configuration of the robot is defined using D-H parameters for each joint. The joints are assumed to be either revolute or prismatic. After locating the coordinate frames at each joint, the parameters are defined as follows:

a: length of the common normal (which is the line perpendicular to both the previous z-axis and the current one)

Alpha: angle about the common normal, measured from the previous z-axis to the current z-axis

d: offset distance along the previous z to the common normal

Theta: angle about the previous z-axis, measured from the previous x-axis to the current x-axis.

The parameters (a, alpha, d and theta) are then substituted into the following matrix:

$$A_i^{i-1} = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The matrix A is generated for each joint separately. After that, the following product is calculated to obtain the final form of the forward kinematics equation:

$$T_3^0(q) = A_1^0 A_2^1 A_3^2$$

3.2 - Equations of Motion

Our Equations of Motion (EoM) GUI uses the homogeneous transformation matrices produced through forward kinematics to create the rotational and translation Jacobian matrices for the links and motors of the robot, using the following equations:

$$J_{Pj}^{Li} = \begin{cases} z_{j-1} & \text{Prismatic} \\ z_{j-1} \times (P_{Li} - P_{j-1}) & \text{Revolute} \end{cases} \quad ; \quad J_{oj}^{Li} = \begin{cases} 0 & \text{Prismatic} \\ z_{j-1} & \text{Revolute} \end{cases}$$

$$J_{Pj}^{mi} = \begin{cases} z_{j-1} & \text{Prismatic} \\ z_{j-1} \times (P_{mi} - P_j - 1) & \text{Revolute} \end{cases} \quad ; \quad J_{0j}^{mi} = \begin{cases} J_{oj}^{Li} & j = 1,2,\dots,i-1 \\ kr_i z_{mi} & j = i \end{cases}$$

These jacobians combined with the mass and inertia properties as well as the "other parameters" described in the Geometry input tab are then used to symbolically generate the robot's Analytical Equations of Motion via the implementation of the following governing equations:

$$\boldsymbol{B(q)\ddot{q} + C(q,\dot{q})\dot{q} + F_v\dot{q} + F_s\mathbf{sgn}(\dot{q}) + g(q) = \tau - J^T(q)h_e}$$

$$B(q) = \sum_{i=1}^{n} (m_{Li} {J_P^{Li}}^T J_P^{Li} + {J_o^{Li}}^T R_i I_{Li}^i R_i^T J_o^{Li} + m_{mi} {J_P^{mi}}^T J_P^{mi} + {J_o^{mi}}^T R_{mi} I_{mi}^{mi} R_{mi}^T J_o^{mi})$$

$$C_{ij} = \sum_{k=1}^{n} c_{ijk}\dot{q}_k \quad ; \quad c_{ijk} = \frac{1}{2}\left(\frac{\partial b_{ij}}{\partial q_k} + \frac{\partial b_{ik}}{\partial q_j} - \frac{\partial b_{jk}}{\partial q_i}\right)$$

$$g_i(q) = -\sum_{j=1}^{n} (m_{Li} g_o^T J_{Pi}^{Lj}(q) + m_{mj} g_o^T J_{Pi}^{mj}(q))$$

3.3 - Joint Dynamics

Our joint dynamics GUI leverages the forward and inverse kinematics functions from Peter Corke's Robotics Toolbox. We build out the input trajectories for the six degrees of freedom as a time series of transformation matrices, which are inputted into the SerialLink `ikine` function to obtain the joint variable trajectory. We use numerical differentiation to obtain the corresponding joint velocity and acceleration plots. Finally, the SerialLink `fkine` function is used to compute the

end effector pose from the joint poses. Plotted alongside the desired pose, we can compare the result effectively.

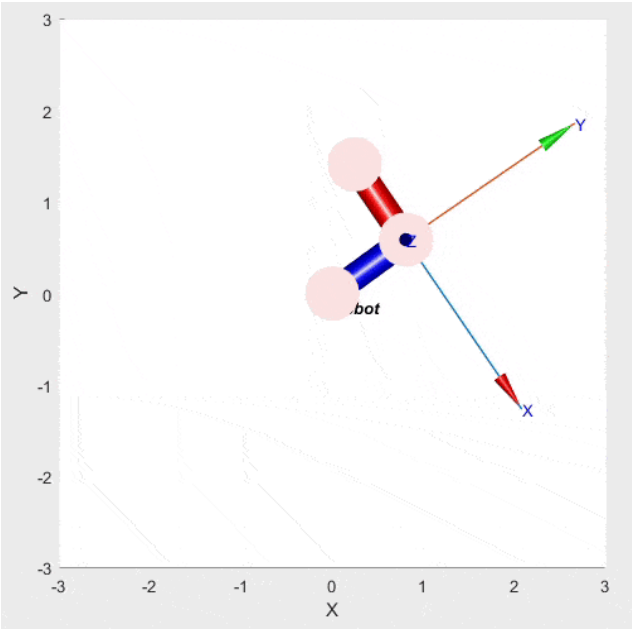


Figure 3: Dynamics Demo

3.4 - Compliance Control

Our compliance control GUI is built on a PD controller with gravity compensation. The error between the robot's pose and the desired end-effector pose is utilized to compute the required generalized torques to move the joints and correct this error. The `ManipulatorMechanicsSubsystem` models the robot's forward dynamics and gives us the joint variables required to recompute the error and make iterative corrections to the system. We use the operational space to compute error and the analytical Jacobian is in the RYP frame. Due to time constraints, we do not account for singularities in our implementation.

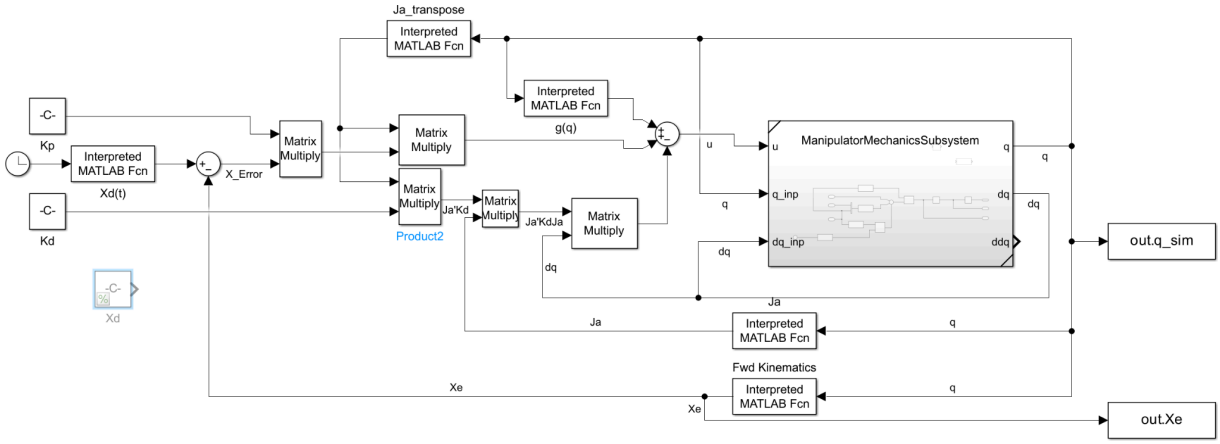


Figure 4: Indirect Compliance Control with PD Controller and Gravity Compensation

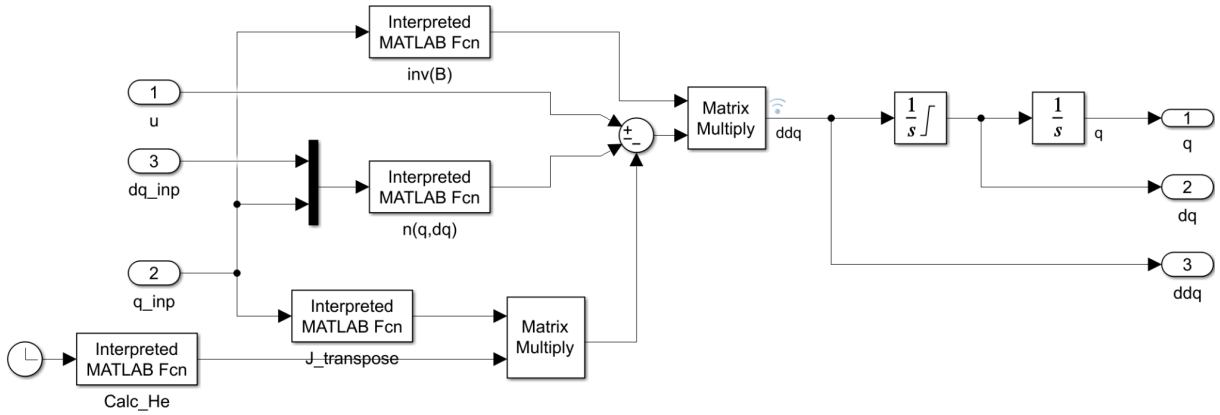


Figure 5: Generalized Manipulator Model

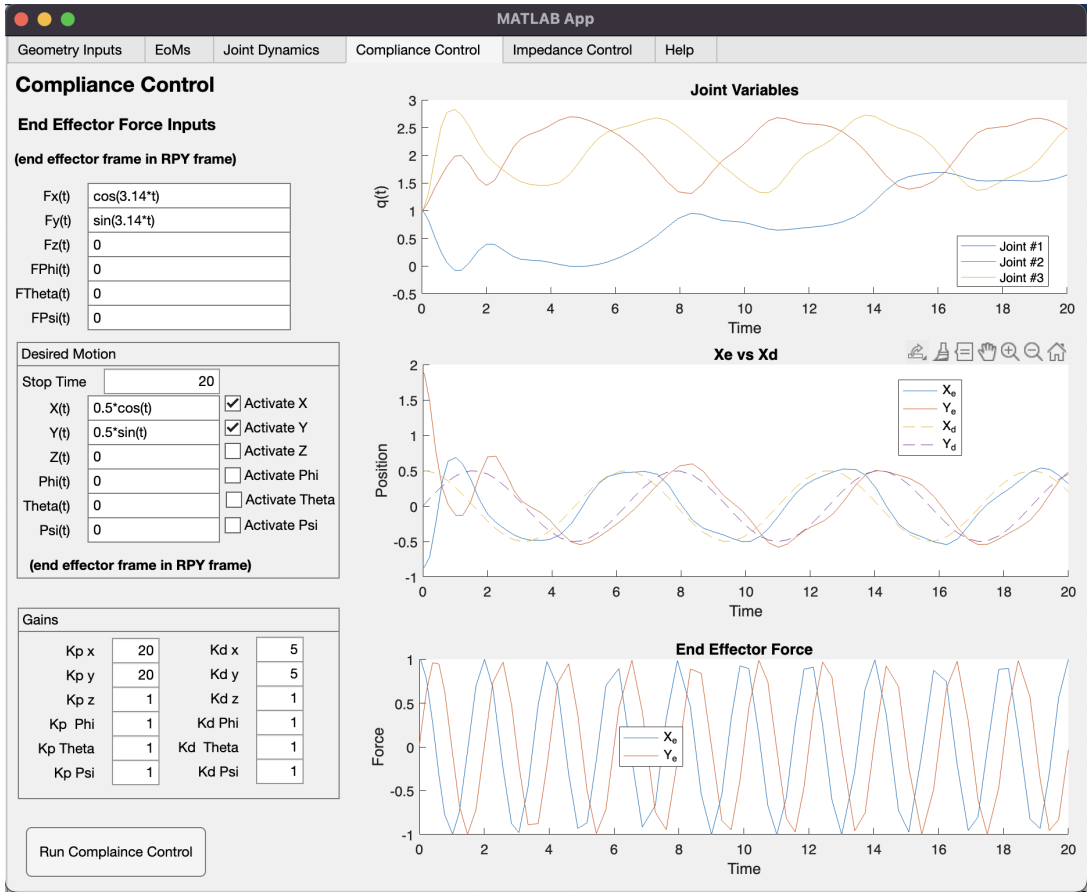


Figure 6: Compliance Control Tracking Plot

3.5 - Impedance Control

We represent the indirect impedance control law (Equation 2) as a Simulink model (Figure 7). As with our compliance control solution, we use the `ManipulatorMechanicsSubsystem` for forward dynamics. Impedance control offers more fine-tuned control than compliance control because it has three tuneable "knobs" M_d, k_D, *and* k_P for inertial/mass, damping, and compliance control respectively. The mass matrix knob provides acceleration control. y is computed from the pose error, velocity error, and desired acceleration. We then use inverse dynamics (Equation 1) to find the desired generalized torques to apply to our manipulator model. We use the operational space to compute error and the analytical Jacobian is in the RYP frame. Due to time constraints, we do not account for singularities in our implementation.

$$(1)\ u = B(q)y + n(q,\dot{q})$$
$$(2)\ y = J_A^{-1}(q)M_d^{-1}(M_d\ddot{x}_d + k_D\dot{\tilde{x}} + k_P\tilde{x} - \dot{M_d}\dot{J_A}(q,\dot{q})\dot{q})$$

Equation 1: Inverse Dynamics
Equation 2: Impedance Control Law

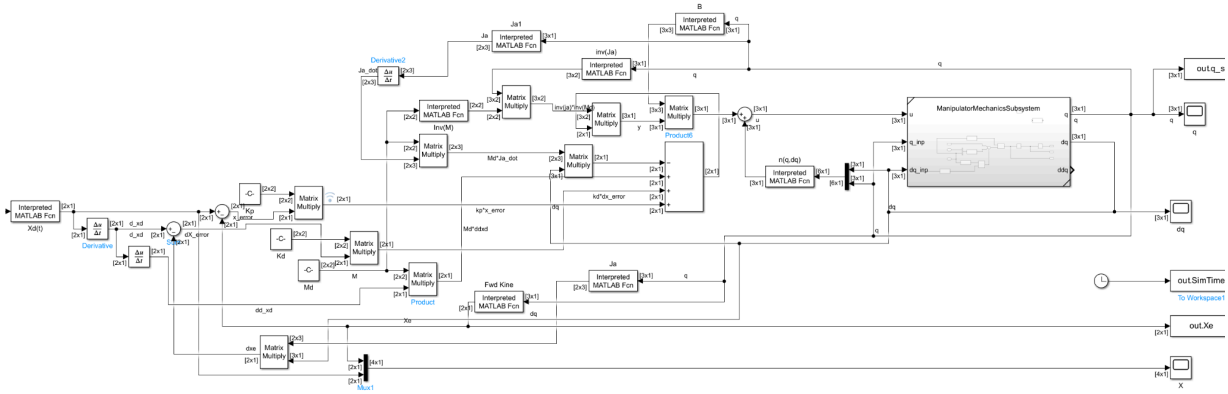


Figure 7: Impedance Control Block Diagram

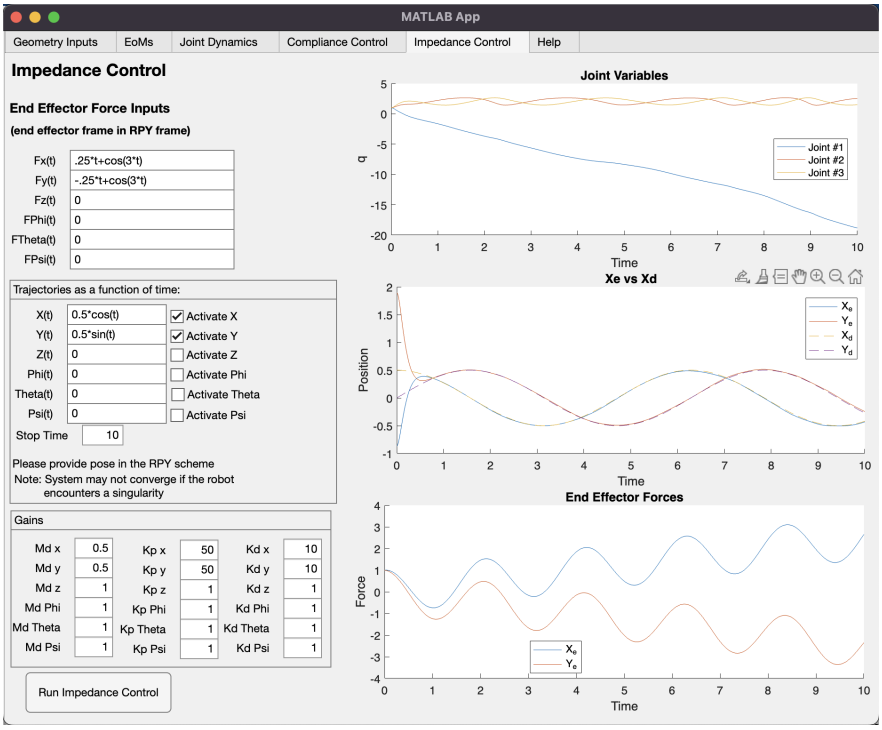


Figure 8: Impedance Control Tracking Plot

4 - Conclusion

In this project, we present a Robotics Dynamics Simulation and Control platform with GUI, providing users with a convenient interaction for simulating and testing robotics manipulation tasks. The project comprises five key modules: robot configuration, equations of motion, joint dynamics, compliance control, and impedance control, through which users can conduct various simulation experiments on robotic arms.

In the project, we utilize techniques and theoretical knowledge learned from the class, such as the DH parameter modeling method, and control methods like compliant control and impedance control. With these techniques and methods, users can quickly create robotic manipulators and perform control experiments in a simulation environment. Overall, we have deepened our understanding of the theoretical knowledge learned in the classroom. We also offer a practical tool to assist users in experimentation and research within the robotics domain, intuitively comprehend the behavior of robots, and explore the effectiveness of various control algorithms.

5 - Individual Contributions

Eric Weissman:

- GUI
- Generate EoMs
- Generate Joint Dynamics
- Generate Compliance Control
- Generate Impedance Control
- Force Control Block Functions

Willem Grier:

- Function to Generate Robotics Toolbox SerialLink Object
- Forward Dynamics Debug
- GUI Debug and Improvements
- User Manual

Wen Yang:

- Compliance Control Simulink Model
- Force Control Debug
- User Manual

Timothy Nieves:

- User Manual

Emad Anvar Siddikui:

- User Manual

REFERENCES

[1] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, Giu, seppe Oriolo. \Robotics Modelling, Planning and Control". Springer. 2009.