FACULTY OF ENGINEERING, ALEXANDRIA UNIVERSITY
EPP333-MECHATRONICS

GUI-BASED SERIAL LINK KINEMATICS SOLVER

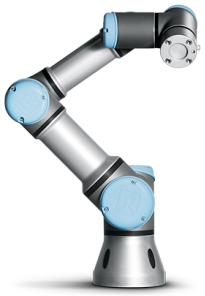
May 5, 2024

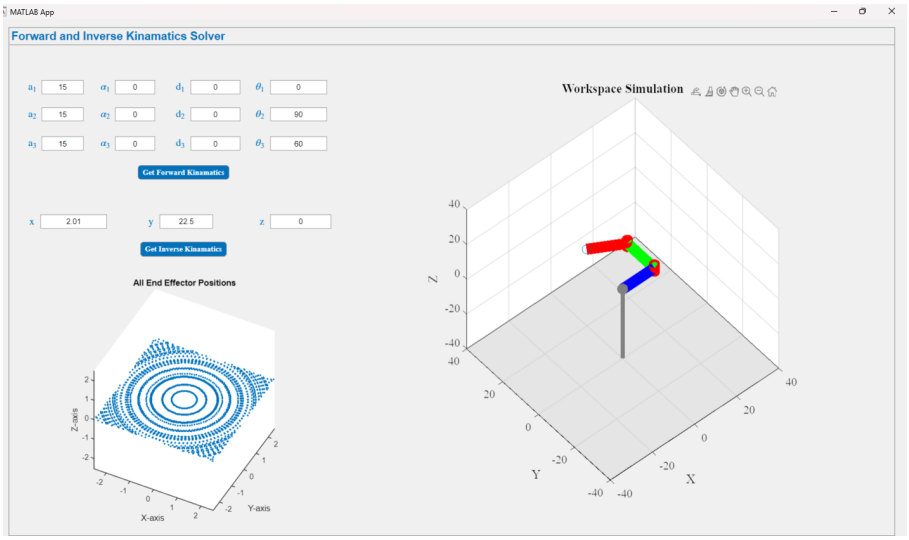
Name: Ahmed Samy Mohammed Elnozahy

ID: 20010099

1 Introduction

In this document, i present the design and implementation of a GUI-Based Serial Link Kinematics Solver developed for the EPP333-Mechatronics course.
The solver is aimed at providing a user-friendly interface for solving serial link kinematics problems, including forward and inverse kinematics, and visualization of robot workspace.





2 GUI Design

The GUI design of the MATLAB App consists of the following components and functionality:

1. **Forward and Inverse Kinematics Solver Panel:** This panel serves as the main interface for users to interact with the forward and inverse kinematics solver.

2. **Workspace Simulation UI (UIAxes):** The `UIAxes` component provides a visual representation of the robot's workspace and movement. It displays the robot's links and end effector position based on the input parameters provided by the user. The `Simulation()` function is responsible for plotting the robot's links and end effector position within this `UIAxes`.

3. **All Possible Points UI (UIAxes2):** The `UIAxes2` component visualizes all possible end effector positions for various combinations of joint angles. It utilizes the `allendeffectors()` function to calculate and display these positions.

4. **Input Fields for Joint Angles and Parameters:** Edit fields (`theta1`, `theta2`, `theta3`, etc.) allow users to input joint angles for the robot manipulator. Edit fields for other parameters (`a1`, `a2`, `a3`, `d1`, `d2`, `d3`, `alpha1`, `alpha2`, `alpha3`) enable users to define the robot's geometry.

5. **Buttons for Kinematics Calculation:** "Get Forward Kinematics" and "Get Inverse Kinematics" buttons trigger the respective kinematics calculations based on the user input. These buttons are linked to the `GetForwardKinematicsButtonPushed()` and `GetInverseKinematicsButtonPushed()` callback functions.

6. **Display Fields for Calculated Results:** Edit fields (`x`, `y`, `z`) display the calculated end effector position after forward kinematics calculation. These fields are updated dynamically based on the results of the kinematics calculations.

3 Forward Kinematics Solver

Forward kinematics determines the end effector's position and orientation based on the joint angles of a robot manipulator. In this GUI application, we use the Denavit-Hartenberg (DH) parameters provided by the user to calculate the end effector's position.

DH parameters consist of joint angles (θ_i), link lengths (a_i), link offsets (d_i), and link twists (α_i). The transformation matrix T is computed iteratively using these parameters and the DH transformation function.

The DH transformation matrix T is given by:

$$T = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i)\cos(\alpha_i) & \sin(\theta_i)\sin(\alpha_i) & a_i\cos(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i)\cos(\alpha_i) & -\cos(\theta_i)\sin(\alpha_i) & a_i\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The MATLAB code snippet below demonstrates how the forward kinematics calculation is performed:

Listing 1: Forward Kinematics Solver()

```
function GetForwardKinamaticsButtonPushed(app, event)
    ...
%%%%%%%%%%%%%%%%%%  DH transformation function %%%%%%%%%%%%%%%%%%
function T = dh_transform(theta, d, a, alpha)
    % Convert degrees to radians
    theta = deg2rad(theta);
    alpha = deg2rad(alpha);

% Calculate transformation matrix
T = [
cos(theta) -sin(theta)*cos(alpha) sin(theta)*sin(alpha) a*cos(theta);
sin(theta) cos(theta)*cos(alpha) -cos(theta)*sin(alpha) a*sin(theta);
0 sin(alpha) cos(alpha) d;
0 0 0 1
];
end
%%%%%%%% Calculate transformation matrices for each joint %%%%%%%%
    for i = 1:size(dh_params, 1)
        theta = dh_params(i, 1);
        d = dh_params(i, 2);
        a = dh_params(i, 3);
        alpha = dh_params(i, 4);

        % Calculate transformation matrix
        T_i = dh_transform(theta, d, a, alpha);
        T = T * T_i;
    end
end
```

This code snippet defines the ‘GetForwardKinamaticsButtonPushed‘ callback function, which is executed when the user clicks the "Get Forward Kinematics" button. It retrieves the joint angles and DH parameters from the GUI, calculates the transformation matrix T using the DH parameters, and extracts the end effector position from the transformation matrix. Finally, it updates the GUI with the calculated end effector position.

4 Inverse Kinematics Solver

- Inverse kinematics determines joint angles for the end effector's desired position and orientation. This GUI calculates these angles (θ_i) to reach a specified goal point ($x_{\text{goal}}, y_{\text{goal}}, z_{\text{goal}}$).
- We use an iterative Jacobian transpose method, adjusting joint angles until the end effector reaches the goal position within a specified error margin.
- To compute the Jacobian matrix, we perturb each joint angle (ξ) and observe the resulting change in end effector position.

$$J = \begin{bmatrix} \frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} & \frac{\partial x}{\partial \theta_3} \\ \frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2} & \frac{\partial y}{\partial \theta_3} \\ \frac{\partial z}{\partial \theta_1} & \frac{\partial z}{\partial \theta_2} & \frac{\partial z}{\partial \theta_3} \end{bmatrix}$$

By using the Jacobian matrix to compute the change in joint angles needed to move the end effector towards the goal point. Also Update joint angles iteratively until convergence or the maximum number of iterations is reached.

5 Robot Workspace Drawing

- The `allendeffectors` function iterates through the specified ranges of joint angles (Θ_1, Θ_2, Θ_3) to compute all possible end effector positions. It stores the calculated positions in arrays ($X_{\text{end effector}}$, $Y_{\text{end effector}}$, $Z_{\text{end effector}}$).
- The `calculateEndEffectorPosition` function computes the end effector position based on the given joint angles Θ_1, Θ_2, and Θ_3. It uses the forward kinematics equations to determine the end effector's X, Y, and Z coordinates.
- The `Simulation` function simulates the robot's movement by plotting its links and joints. It calculates the positions of each joint based on the given joint angles and link lengths. For each joint, it plots a cylinder representing the joint and its corresponding link. Additionally, it plots a sphere at the end effector's position to visualize its location.