# Robotic Welding Arm Assembly for Manufacturing Automation

**Sannjay Balaji**

## Forward Kinematics:

### DH (Denavit-Hartenberg) Table for the Robot Manipulator:

| **Joint *i*** | **$a_i$ (mm)** | **$\alpha_i$** | **$d_i$ (mm)** | **$\theta_i$** |
|---|---|---|---|---|
| **1** | 0 | 0 | 194.95 | $\theta_1$ |
| **2** | 310.77 | 0 | 0 | $\theta_2$ |
| **3** | 800.51 | 90 | 0 | $\theta_3$ |
| **4** | 532.99 | -90 | 0 | $\theta_4$ |
| **5** | 0 | 90 | 371.40 | $\theta_5$ |
| **6** | 0 | 0 | 269.07 | $\theta_6$ |

**Estimated Weld Points Based on Red-Highlighted Areas on the Headstock:**

Workbench and Headstock Position in the Robot's Workspace

1. Workbench Position:
   - The workbench is located 1000 mm from the robot base along the x-axis.
   - Its height is 122 mm.
2. Headstock Position:
   - The headstock is placed on top of the workbench, so it is elevated by the workbench height: $z_{\text{headstock}} = 122\text{ mm} + (\text{Headstock Height}/2) = 122\text{ mm} + 68.32\text{ mm} = 190.32\text{ mm}$
   - Assuming the headstock is centred along the length of the workbench (800 mm), the x-position of the headstock's centre is approximately: $x_{\text{headstock}} = 1000\text{ mm} + (800/2) - (159.98/2) = 1170.01\text{ mm}$

**Weld Points on the Headstock:**

Weld Point 1 (Start of Red-Highlighted Area):

| | |
|---|---|
| Position: | $x \approx 1170.01$ (starting point along headstock's length).<br>$y=0$ (centred on workbench width).<br>$z=190.32$ (top of the workbench and headstock height). |

Weld Point 2 (Middle of Red-Highlighted Area):

| | |
|---|---|
| Position: | ○ $x=1170.01\text{ mm}+80\text{ mm}=1250.01\text{ mm}$.<br>○ $y=0$.<br>○ $z=190.32$. |

Weld Point 3 (End of Red-Highlighted Area):

| | |
|---|---|
| Position: | $x=1170.01\text{ mm}+160\text{ mm}=1330.01\text{ mm}$.<br>$y=0$.<br>$z=190.32$ |

Additional Weld Points (Top and Side Welds):

| Top Weld: | Side Weld: |
|---|---|
| x=1250.01 mm (centre along the headstock's length)<br>y=0 mm<br>z=190.32+68.32=258.64 mm (top of the headstock) | x=1250.01 mmx = 1250.01<br>y= ±21.35 mm (half the width of the headstock).<br>z=190.32 mm |

Firstly, in to compute this in MATLAB we do the following

Function Header: Defines the function named ForwardKinematics with an input q. The function returns T, which is the final transformation matrix.

Input Parameter (q): A vector containing the joint variables (joint angles in this case) for the 6 joints of the robot.

**Defining DH Parameters**

- **DH Parameters**: The table of Denavit-Hartenberg parameters for each of the 6 joints is defined here. Each row represents the DH parameters of a joint. The columns represent:
    1. **a** (Link length): The distance between the z-axes of two consecutive joints along the x-axis.
    2. **alpha** (Link twist): The angle between the z-axes of two consecutive joints, measured about the x-axis.
    3. **d** (Link offset): The distance between the x-axes of two consecutive joints along the z-axis.
    4. **theta** (Joint angle): The rotation angle of the joint along its own z-axis.
- Each row contains the respective DH parameters for one joint. Note that q(i) is used for the theta value of each joint because the joint angles are variable, depending on the input vector q.

**Loop through Each Joint to Calculate the Transformation Matrix**

- **Loop to Calculate Transformations**: This for loop iterates through each joint (each row of the dh_params matrix). The values a, alpha, d, and theta are extracted for each joint from the dh_params matrix.
- **Transformation Matrix for Each Joint (Ti)**: The transformation matrix Ti for each joint is computed using the DH parameters:
    - The matrix Ti represents the transformation between two consecutive coordinate frames in terms of rotation and translation.
    - The elements of the matrix are calculated based on trigonometric functions (cos and sin) applied to the theta (joint angle) and alpha (link twist).

The matrix structure:

- **First Row and Second Row**: These elements represent rotations and translations in the x-y plane.

  - **Third Row**: This row represents the effect of twisting and offsets along the z-axis.
  - **Fourth Row**: This row is [0, 0, 0, 1] and is necessary to keep the matrix compatible for homogeneous transformation.
- **Update the Total Transformation (T)**: The total transformation matrix T is updated by multiplying it with each joint transformation Ti. The multiplication accumulates the transformations for each joint, effectively resulting in the transformation from the base frame to the end-effector.

We assume the following joint angles for forward kinematics:

q1 = [0, pi/4, -pi/4, pi/2, -pi/6, pi/3];

q2 = [pi/6, -pi/3, pi/4, -pi/2, pi/6, -pi/4];

q3 = [-pi/6, pi/3, -pi/4, pi/4, -pi/3, pi/6];

MATLAB output after calculating the Transformation matrix for the following joint angles: -

Forward Kinematics for q1:

```
  -0.8660   -0.5000   -0.0000  648.8576

  -0.2500    0.4330   -0.8660  -13.2739

   0.4330   -0.7500   -0.5000  593.4050

       0         0         0    1.0000
```

Forward Kinematics for q2:

```
  -0.0008    0.0006    0.0002    1.4614

   0.0002    0.0005   -0.0008   -0.0772

  -0.0006   -0.0006   -0.0005   -0.4726

       0         0         0    0.0010
```

Forward Kinematics for q3:

```
  -0.2399   -0.6502   -0.7209  958.7592

  -0.7122    0.6225   -0.3245 -168.6802

   0.6597    0.4356   -0.6124  669.6793

       0         0         0    1.0000
```

***# The program is attached at the appendix with its outputs as screenshots.***

## Inverse Kinematics:

We write a script in MATLAB to perform **inverse kinematics** for a 6-degree-of-freedom (DOF) robotic manipulator using **optimization**. The goal is to determine the joint angles (q_solution) that achieve a specified target transformation matrix (T_target). The solution is found using **non-linear optimization**, specifically using the fmincon function.

We initialize a variable T_target which we input the transformation matrix we acquired from the forward kinematics solution to find the optimal joint angles required to perform the task efficiently for the joint angles of, we input the target matrix from the forward kinematics solution.

Additionally, the script uses the fmincon optimization function to find the joint angles that minimize the error between the current pose and the target pose.

optimoptions is used to specify solver settings:

- Algorithm: Sequential Quadratic Programming (sqp), which is suitable for constrained optimization.
- Tolerances and maximum evaluations are also specified to control the precision and duration of optimization.

The poseError function computes the pose error, which the optimization algorithm tries to minimize and solve for the inverse kinematics, resulting in the following joint angles.

**Define Joint Limits**

- **Joint Limits**: Each joint angle has an upper (q_max) and lower (q_min) bound to ensure the solution is physically possible and falls within the robot's joint limits. These values help fmincon stay within the allowed range during optimization.

**Define the Objective Function**

- **objective**: Defines an anonymous function for optimization. The function poseError(q, T_target) computes the error between the current transformation matrix of the robot (calculated using q) and the desired transformation matrix (T_target). The goal is to minimize this error.

**Run the Optimization**

- **fmincon**: The MATLAB function used to perform constrained optimization.
  - **objective**: The function to minimize, which is the pose error.
  - **q_initial**: The initial guess for joint angles.
  - **[ ] (Equality and Inequality Constraints)**: No linear inequality or equality constraints are provided here.
  - **q_min and q_max**: These are joint limits provided as lower and upper bounds.
  - **options**: Optimization options defined earlier.
- **q_solution**: The joint angles that minimize the pose error.
- **fval**: The final value of the objective function (i.e., the minimized pose error).

| $Q_i$ | T_Target | Joint angles |
|---|---|---|
| [0, pi/4, -pi/4, pi/2, -pi/6, pi/3] | [ -0.8660, -0.5000, -0.0000, 648.8576; -0.2500, 0.4330, -0.8660, -13.2739; 0.4330, -0.7500, -0.5000, 593.4050; 0, 0, 0, 1.0000 ] | [-0.0000 0.7854 -0.7854 1.5708 -0.5236 1.0472] |
| [pi/6, -pi/3, pi/4, -pi/2, pi/6, -pi/4] | [ -0.0008, 0.0006, 0.0002, 1.4614; 0.0002, 0.0005, -0.0008, -0.0772; -0.0006, -0.0006, -0.0005, -0.4726; 0, 0, 0, 0.0010 ] | [ -0.6291 -0.4361 -2.1041 2.8516 0.1052 1.0486] |
| [0, pi/4, -pi/4, pi/2, -pi/6, pi/3] | [ -0.2399, -0.6502, -0.7209, 958.7592; -0.7122, 0.6225, -0.3245, -168.6802; 0.6597, 0.4356, -0.6124, 669.6793; 0, 0, 0, 1.0000 ] | [-0.3625 0.3359 0.0611 1.0565 -0.8245 0.4645] |

***Verification of Forward and Inverse Kinematic Solutions:***

We write a new MATLAB script which is intended to **verify the Inverse Kinematics (IK) solution** obtained in a previous IK calculation. The goal is to use the joint angles solution (q_solution) found during the IK process and calculate the **Forward Kinematics (FK)** to check whether the calculated pose matches the desired target pose (T_target). The joint angles solution (q_solution) is provided as input, which is obtained from a previous inverse kinematics calculation. These joint angles are used to compute the forward kinematics.

**T_target**: This 4x4 matrix represents the desired position and orientation of the end-effector. It is the same transformation matrix that was used as a reference in the inverse kinematics problem.

- The **first three columns** represent the rotation matrix that describes the orientation of the end-effector.
- The **fourth column** ([648.8576; -13.2739; 593.4050]) represents the position vector of the end-effector in Cartesian coordinates.

**Define the Solution Obtained from Inverse Kinematics (q_solution)**

- **q_solution**: This vector represents the joint angles obtained from the inverse kinematics optimization. The values in this vector should match those obtained from the fmincon solver used in the inverse kinematics script.
- These angles correspond to the positions of the joints of the robot to achieve the T_target transformation matrix.

**Calculate Forward Kinematics for the Solution (T_check)**

T_check = ForwardKinematicsVerification(q_solution);

- **T_check**: This is the calculated transformation matrix based on the given joint angles (q_solution). The function ForwardKinematicsVerification(q_solution) is used to calculate this matrix.
- This matrix represents the position and orientation of the end-effector computed from the joint angles, which can be compared to T_target to check the correctness of the inverse kinematics solution.

| **Angles provided for FK** | **Angles Calculated from IK** | **T_Forward** | **T_Inverse** |
|---|---|---|---|
| [0, pi/4, -pi/4, pi/2, -pi/6, pi/3] | [-0.0000 0.7854 -0.7854 1.5708 -0.5236 1.0472] | [ -0.8660, -0.5000, -0.0000, 648.8576; -0.2500, 0.4330, -0.8660, -13.2739; 0.4330, -0.7500, -0.5000, 593.4050; 0, 0, 0, 1.0000 ] | [ -0.8660 -0.5000 0.0000 648.8557 -0.2500 0.4330 -0.8660 -13.2733 0.4330 -0.7500 -0.5000 593.4034 0 0 0 1.0000 ] |
| [pi/6, -pi/3, pi/4, -pi/2, pi/6, -pi/4] | [ -0.6291 -0.4361 -2.1041 2.8516 0.1052 1.0486] | [ -0.0008, 0.0006, 0.0002, 1.4614; 0.0002, 0.0005, -0.0008, -0.0772; -0.0006, -0.0006, -0.0005, -0.4726; 0, 0, 0, 0.0010 ] | [ -0.0008 0.0006 0.0002 1.4614 0.0002 0.0005 -0.0008 -0.0772 -0.0006 -0.0006 -0.0005 -0.4726 0 0 0 0.0010 ] |
| [0, pi/4, -pi/4, pi/2, -pi/6, pi/3] | [-0.3625 0.3359 0.0611 1.0565 -0.8245 0.4645] | [ -0.2399, -0.6502, -0.7209, 958.7592; -0.7122, 0.6225, -0.3245, -168.6802; 0.6597, 0.4356, -0.6124, 669.6793; 0, 0, 0, 1.0000 ] | [ -0.2399 -0.6502 -0.7209 958.7592 -0.7122 0.6225 -0.3245 -168.6802 0.6597 0.4356 -0.6124 669.6793 0 0 0 1.0000 ] |

We can conclude that the initial angles taken for the 6 DOF Robotic manipulator are true and henceforth can perform the necessary welding operations.

## Differential Kinematics:

This script calculates the **Jacobian matrix** for a 6-degree-of-freedom (DOF) robotic manipulator using **Denavit-Hartenberg (DH) parameters**. The **Jacobian** is essential for understanding how changes in joint variables (q) affect the end-effector's velocity in Cartesian space.

**Calculate the Jacobian Matrix (J)**

J = Jacob(q3);

- **J = Jacob(q3)**: Calls the local function Jacob(q) to calculate the Jacobian matrix based on the given joint angles (q3).
- The function returns a **6x6 Jacobian matrix**, which contains information about both linear and angular velocity contributions.

**Initializing Transformation Matrices and Jacobian**

- **T**: The initial transformation matrix is set as the **identity matrix** (4x4). It is used to keep track of the transformation from the base frame to each successive joint.
- **J**: The **Jacobian matrix** is initialized as a 6x6 zero matrix. The Jacobian contains six rows:
    - The **first three rows** represent the **linear velocity** components.
    - The **last three rows** represent the **angular velocity** components.

**Loop to Compute the Jacobian**

1. **Loop to Compute Transformation for Each Joint (for i = 1:num_joints)**:
    - A for loop iterates over each joint to calculate the transformation matrix (T) and extract the necessary vectors for Jacobian computation.
2. **DH Parameter Extraction**:
    - For each joint i, the DH parameters (a, alpha, d, theta) are extracted from the matrix dh_params.
3. **Transformation Matrix for Joint i (Ti)**:
    - The transformation matrix Ti is calculated based on the DH parameters, which gives the transformation from frame i-1 to frame i.
4. **Update Transformation from Base to Joint i (T = T * Ti)**:
    - The cumulative transformation (T) is updated by multiplying it with Ti.
5. **Extracting Position Vector (p) and Z-Axis (z)**:
    - **p**: Extracts the **position vector** of the current joint from the transformation matrix (T).
    - **z**: Extracts the **z-axis** of the current joint from the transformation matrix (T). The z-axis is important because it defines the axis of rotation for revolute joints.
6. **Compute Linear Velocity Contribution (Jv)**:

- **Jv**: Computes the **linear velocity part** of the Jacobian using the **cross product** of z and p. This represents the linear velocity component of the end-effector due to the rotation of joint i.

7. **Fill the Jacobian Matrix (J)**:
   - The **linear part** (Jv) is assigned to the **first three rows** of column i in J.
   - The **angular part** (z) is assigned to the **last three rows** of column i in J.

**Analyzing the Jacobian Matrices for q1, q2, and q3**:

1. General Observations:
   - Each Jacobian matrix shows different values for linear (Jv) and angular (Jω) components, which makes sense because each configuration (q1, q2, q3) represents a different posture of the manipulator.
   - Changes in joint angles alter the position of the end-effector as well as the orientation of each link, which impacts the contribution of each joint to the overall velocity.
2. Comparison Between the Configurations:

First Configuration (q1):

Jacobian Matrix for q1:

```
  1.0e+03 *

         0   -0.2197   -0.1949   -0.0000   -0.5205   -0.5205
         0    0.2197    0.0000    0.7279   -0.3244   -0.3244
         0         0    1.0203   -0.2197    0.5619    0.5619
         0         0         0   -0.0010   -0.0000   -0.0000
         0         0   -0.0010   -0.0000   -0.0009   -0.0009
    0.0010    0.0010    0.0000    0.0000   -0.0005   -0.0005
```

- The linear components in the Jacobian for q1 show significant non-zero values, indicating that most joints are contributing to the overall movement of the end-effector.
- The angular components are also non-zero, reflecting contributions from rotations around the z-axis of each joint.
- The large values in the linear velocity components suggest that the end-effector is undergoing a movement where positional changes dominate, meaning the manipulator may be extending its links to reach a specific target.

Second Configuration (q2):

```
  1.0e+03 *

         0    0.1554   -0.1883   -0.0875    0.3567    0.3567
```

```
        0    0.2691   -0.0505    0.3265   -0.6248   -0.6248
        0         0    1.0203   -0.2197    1.2052    1.2052
        0         0    0.0003    0.0010    0.0002    0.0002
        0         0   -0.0010    0.0003   -0.0008   -0.0008
   0.0010    0.0010    0.0000    0.0000   -0.0005   -0.0005
```

- The Jacobian matrix for q2 has different values compared to q1, reflecting a change in the end-effector's configuration and the contributions of each joint.
- Here, we notice slight variations in the linear and angular velocities compared to q1. This indicates that the manipulator has changed its posture to adjust its reach and orientation.
- The structure and magnitude of the components vary because different joint angles mean that different axes and link lengths have a greater or lesser effect on the end-effector's pose.

Third Configuration (q3):

```
  1.0e+03 *

        0   -0.1554   -0.1883    0.2103   -0.3206   -0.3206
        0    0.2691    0.0505    1.3850   -0.1043   -0.1043
        0         0    1.0203   -0.1554    0.4327    0.4327
        0         0   -0.0003   -0.0007   -0.0007   -0.0007
        0         0   -0.0010    0.0002   -0.0003   -0.0003
   0.0010    0.0010    0.0000    0.0007   -0.0006   -0.0006
```

- For q3, the Jacobian matrix again shows unique values compared to q1 and q2, indicating a new posture.
- The linear velocity components are more prominent, suggesting a change in the spatial position of the end-effector.
- There are also notable changes in the angular velocity components, indicating that joint rotations are influencing the end-effector's orientation differently compared to the previous configurations.

Linear Velocity (Jv) Components:

- The linear velocity components indicate how much the end-effector's position changes with respect to changes in each joint angle.
- In these matrices, we see relatively large values for linear velocities, which means that even small changes in some joint angles can significantly affect the position of the end-effector.
- This makes sense for an articulated manipulator: joints positioned further from the base will have a larger effect on the end-effector's position due to leverage and link lengths.

    - The different values for the linear velocity components across the three configurations suggest that different postures have different effectiveness in achieving a specific direction of movement.

Angular Velocity (Jω) Components:

    - The angular velocity components represent how much the end-effector's orientation changes with respect to changes in each joint angle.
    - Non-zero values in the angular components indicate that a rotation in a specific joint contributes to the rotation of the end-effector.
    - Comparing q1, q2, and q3, the variations in the angular velocity components reflect how the end-effector's orientation is affected by different configurations.
    - Higher values indicate that the joint in question has a significant influence on the rotation of the end-effector, which can help in achieving precise orientation requirements during a task (e.g., welding or placing an object).

Interpretations of Large Values:

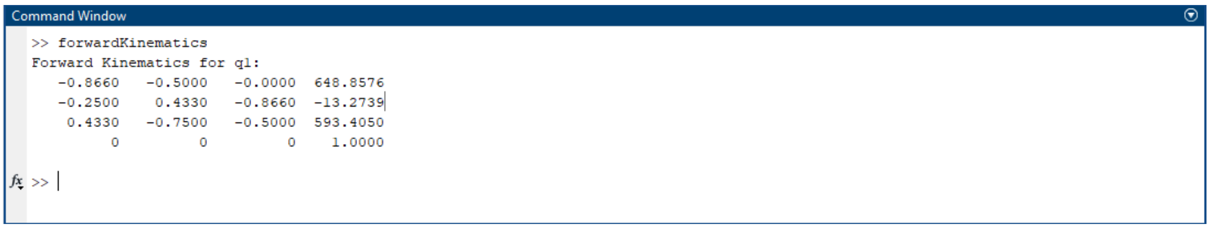
- Large values in the linear velocity components suggest that the manipulator is extended or positioned in a way where joint movement leads to a large change in the position of the end-effector.
- Large angular velocity components indicate that small changes in some joint angles will result in significant changes in the orientation of the end-effector. This is crucial in tasks that require precise orientation, especially for the case of welding as it is a job that requires high precision.
- The overall magnitude of the values in the Jacobian reflects the effectiveness of the manipulator configuration in affecting the end-effector's movement. Certain configurations will make the manipulator more efficient in moving in a particular direction, while others may not be as effective due to joint limitations or the physical geometry of the manipulator.

## Appendix:

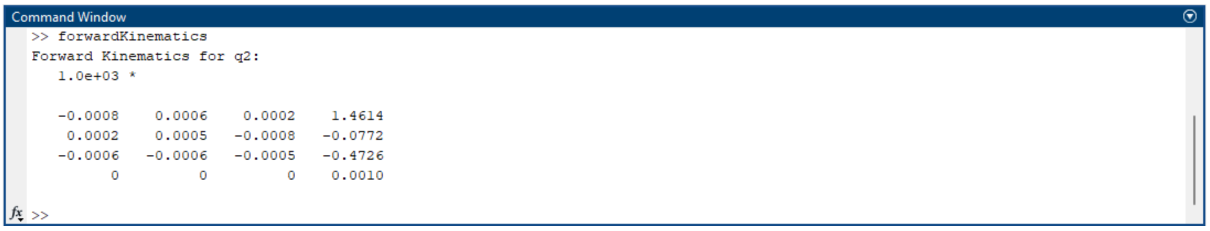
**Forward Kinematics Program:-**

```
function T =ForwardKinematics(q)
   % DH parameters for each joint
   dh_params = [
      0,      0,        194.95, q(1);
      310.77,  0,        0,      q(2);
      800.51,  pi/2,      0,      q(3);
      532.99,  -pi/2,     0,      q(4);
      0,      pi/2,      371.40, q(5);
      0,      0,        269.07, q(6)
   ];
   % Initialize the transformation matrix as identity
   T = eye(4);
   % Loop through each joint to calculate the total transformation matrix
   for i = 1:size(dh_params, 1)
      a = dh_params(i, 1);
      alpha = dh_params(i, 2);
      d = dh_params(i, 3);
      theta = dh_params(i, 4);
      % Compute individual transformation matrix using DH parameters
      Ti = [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];

      % Update the total transformation matrix
      T = T * Ti;
   end
end
q1 = [0, pi/4, -pi/4, pi/2, -pi/6, pi/3];
%q2 = [pi/6, -pi/3, pi/4, -pi/2, pi/6, -pi/4];
```

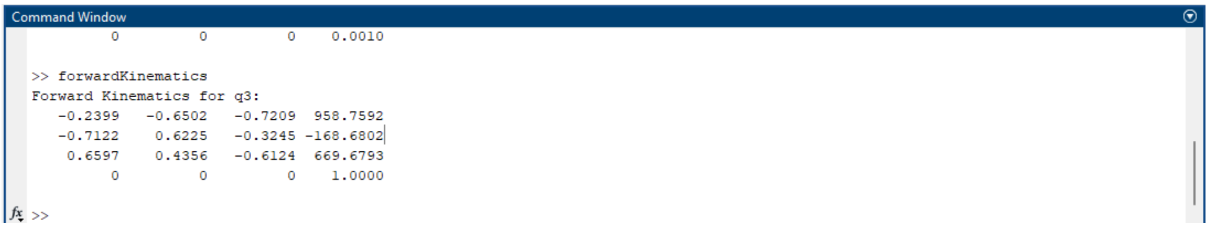
```
%q3 = [-pi/6, pi/3, -pi/4, pi/4, -pi/3, pi/6];
disp('Forward Kinematics for q1:');
disp(ForwardKinematics(q1));
%disp('Forward Kinematics for q2:');
%disp(ForwardKinematics(q2));
%disp('Forward Kinematics for q3:');
%disp(ForwardKinematics(q3));
```



```
Command Window
>> forwardKinematics
Forward Kinematics for q1:
   -0.8660   -0.5000   -0.0000  648.8576
   -0.2500    0.4330   -0.8660  -13.2739
    0.4330   -0.7500   -0.5000  593.4050
         0         0         0    1.0000

fx >>
```



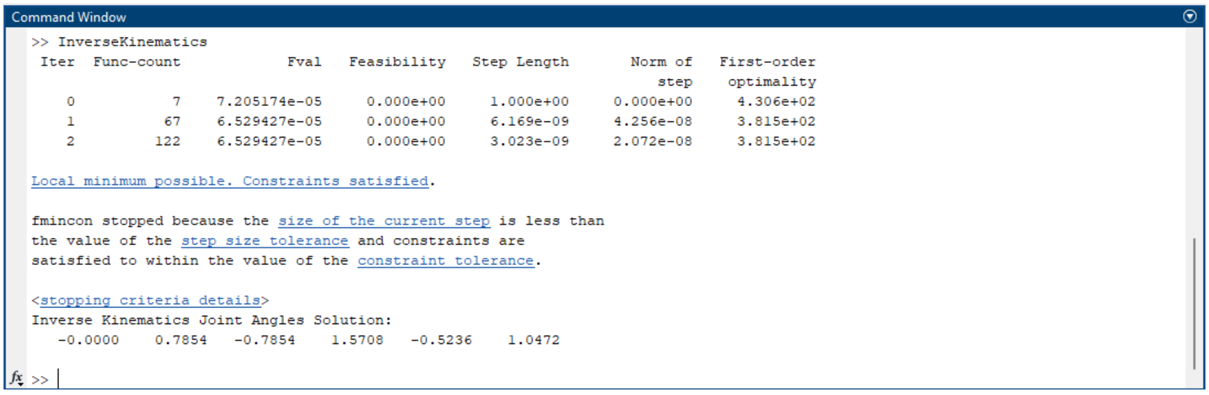
```
Command Window
>> forwardKinematics
Forward Kinematics for q2:
   1.0e+03 *

   -0.0008    0.0006    0.0002    1.4614
    0.0002    0.0005   -0.0008   -0.0772
   -0.0006   -0.0006   -0.0005   -0.4726
         0         0         0    0.0010

fx >>
```



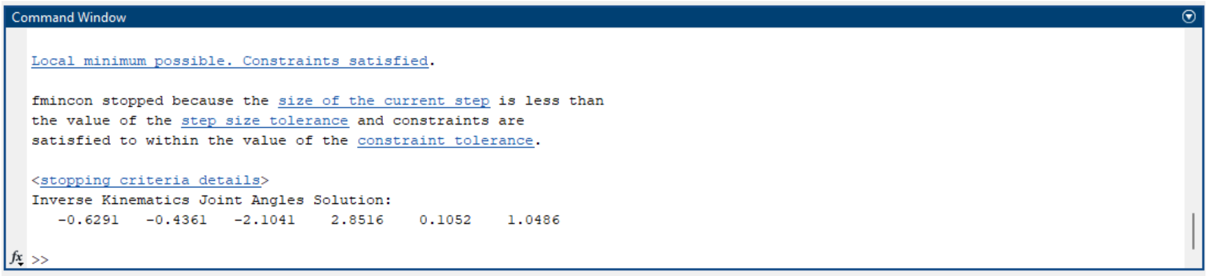
```
Command Window
         0         0         0    0.0010

>> forwardKinematics
Forward Kinematics for q3:
   -0.2399   -0.6502   -0.7209  958.7592
   -0.7122    0.6225   -0.3245 -168.6802
    0.6597    0.4356   -0.6124  669.6793
         0         0         0    1.0000

fx >>
```

**Inverse Kinematics Program:-**

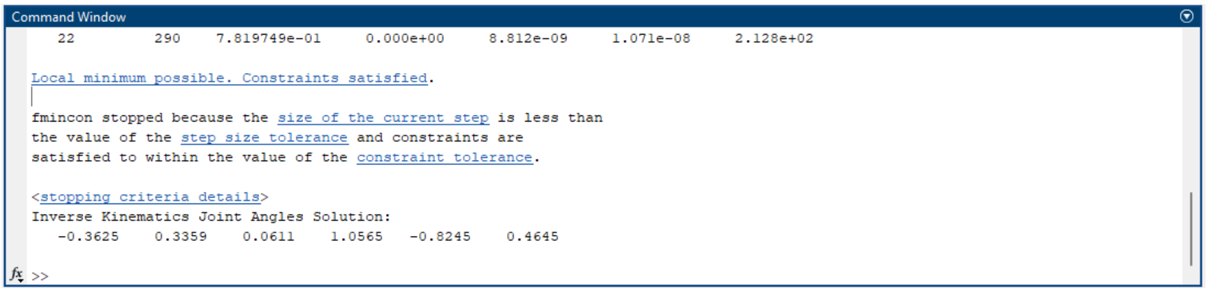
```
% Define the target transformation matrix T_target
T_target = [
   -0.8660, -0.5000, -0.0000, 648.8576;
   -0.2500,  0.4330, -0.8660, -13.2739;
    0.4330, -0.7500, -0.5000, 593.4050;
         0,       0,       0,      1.0000
];
% q1 values for joint angles
q_initial = [0, pi/4, -pi/4, pi/2, -pi/6, pi/3];
% Set optimization options
options = optimoptions('fmincon', 'Display', 'iter', 'Algorithm', 'sqp', ...
    'MaxFunctionEvaluations', 2000, 'OptimalityTolerance', 1e-8, ...
    'StepTolerance', 1e-8, 'FunctionTolerance', 1e-8);
% Define joint limits (optional)
q_min = [-pi, -pi/2, -pi, -pi, -pi/2, -pi];
q_max = [pi, pi/2, pi, pi, pi/2, pi];
% Objective function to minimize the pose error
objective = @(q) poseError(q, T_target);
% Run optimization to solve for joint angles that reach T_target
[q_solution, fval] = fmincon(objective, q_initial, [], [], [], [], q_min, q_max, [], options);
% Display the solution
disp('Inverse Kinematics Joint Angles Solution:');
disp(q_solution);
% --- Local function for Pose Error Calculation ---
function error = poseError(q, T_target)
    % Define DH parameters for each joint
    dh_params = [
        0,       0,         194.95, q(1);
        310.77,  0,         0,      q(2);
        800.51,  pi/2,      0,      q(3);
        532.99,  -pi/2,     0,      q(4);
        0,      pi/2,     371.40, q(5);
        0,      0,        269.07, q(6)
    ];
    % Initialize the transformation matrix as identity
    T_actual = eye(4);
    % Loop through each joint to calculate the total transformation matrix
    for i = 1:size(dh_params, 1)
        a = dh_params(i, 1);
        alpha = dh_params(i, 2);
        d = dh_params(i, 3);
        theta = dh_params(i, 4);
        % Compute individual transformation matrix using DH parameters
        Ti = [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];
        % Update the total transformation matrix
        T_actual = T_actual * Ti;
    end
end
```



```
>> InverseKinematics
 Iter  Func-count            Fval   Feasibility   Step Length       Norm of   First-order
                                                                       step    optimality
    0           7    7.205174e-05     0.000e+00     1.000e+00     0.000e+00     4.306e+02
    1          67    6.529427e-05     0.000e+00     6.169e-09     4.256e-08     3.815e+02
    2         122    6.529427e-05     0.000e+00     3.023e-09     2.072e-08     3.815e+02

Local minimum possible. Constraints satisfied.

fmincon stopped because the size of the current step is less than
the value of the step size tolerance and constraints are
satisfied to within the value of the constraint tolerance.

<stopping criteria details>
Inverse Kinematics Joint Angles Solution:
   -0.0000    0.7854   -0.7854    1.5708   -0.5236    1.0472

>> 
```



```
Local minimum possible. Constraints satisfied.

fmincon stopped because the size of the current step is less than
the value of the step size tolerance and constraints are
satisfied to within the value of the constraint tolerance.

<stopping criteria details>
Inverse Kinematics Joint Angles Solution:
   -0.6291   -0.4361   -2.1041    2.8516    0.1052    1.0486

>> 
```

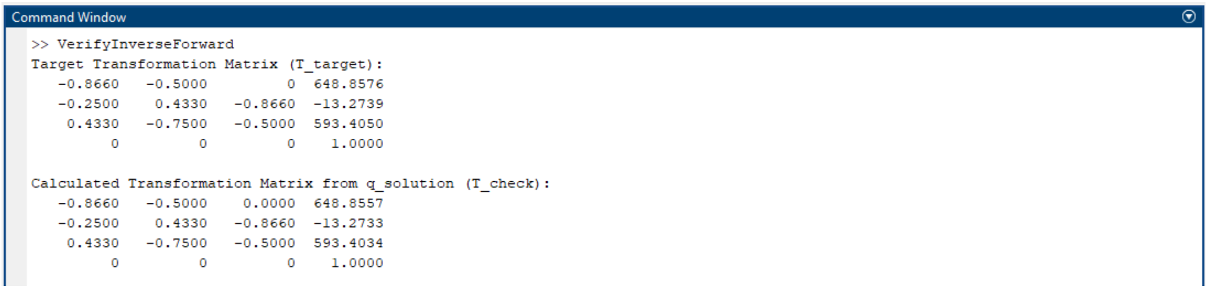


```
   22         290    7.819749e-01     0.000e+00     8.812e-09     1.071e-08     2.128e+02

Local minimum possible. Constraints satisfied.

fmincon stopped because the size of the current step is less than
the value of the step size tolerance and constraints are
satisfied to within the value of the constraint tolerance.

<stopping criteria details>
Inverse Kinematics Joint Angles Solution:
   -0.3625    0.3359    0.0611    1.0565   -0.8245    0.4645

>>
```

**Verification of Forward and Inverse Kinematics Program:-**

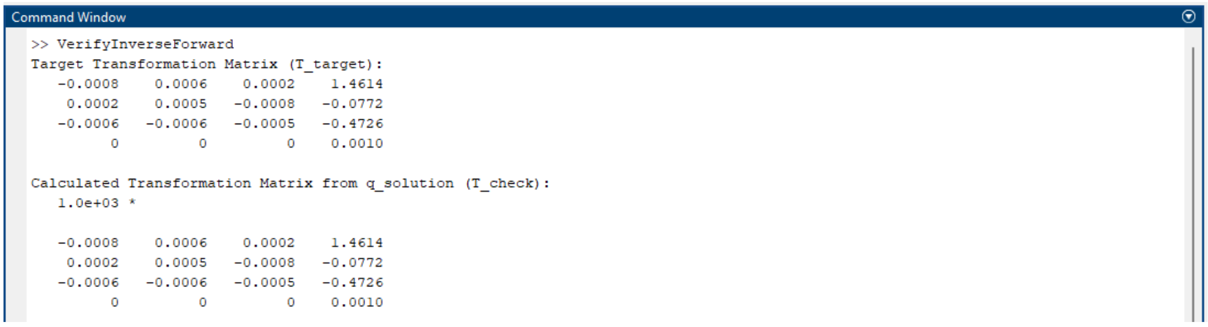
```
% Define the target transformation matrix T_target (from the inverse kinematics problem)
T_target = [
   -0.8660, -0.5000, -0.0000, 648.8576;
   -0.2500,  0.4330, -0.8660, -13.2739;
    0.4330, -0.7500, -0.5000, 593.4050;
         0,       0,       0,      1.0000
];
% Solution obtained from inverse kinematics (q_solution)
% These values should match the output from InverseKinematics.m
q_solution = [-0.0000, 0.7854, -0.7854, 1.5708, -0.5236, 1.0472];
% Calculate the forward kinematics for the solution to get T_check
T_check = ForwardKinematicsVerification(q_solution);
% Display the target and calculated transformation matrices
disp('Target Transformation Matrix (T_target):');
disp(T_target);
disp('Calculated Transformation Matrix from q_solution (T_check):');
disp(T_check);
% Calculate position and orientation differences
position_difference = abs(T_target(1:3,4) - T_check(1:3,4));
orientation_difference = abs(T_target(1:3,1:3) - T_check(1:3,1:3));
% Display the position and orientation differences
disp('Position Difference:');
disp(position_difference);
disp('Orientation Difference:');
disp(orientation_difference);
function T = ForwardKinematicsVerification(q)
```

```
    % Define DH parameters for each joint based on q_solution values
    dh_params = [
        0,       0,         194.95, q(1);
        310.77,  0,         0,      q(2);
        800.51,  pi/2,      0,      q(3);
        532.99,  -pi/2,     0,      q(4);
        0,       pi/2,      371.40, q(5);
        0,       0,         269.07, q(6)
    ];
    % Initialize transformation matrix as identity
    T = eye(4);
    % Loop through each joint to calculate the total transformation matrix
    for i = 1:size(dh_params, 1)
        a = dh_params(i, 1);
        alpha = dh_params(i, 2);
        d = dh_params(i, 3);
        theta = dh_params(i, 4);

        % Compute individual transformation matrix using DH parameters
        Ti = [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];
        % Update the total transformation matrix
        T = T * Ti;
    end
end
```



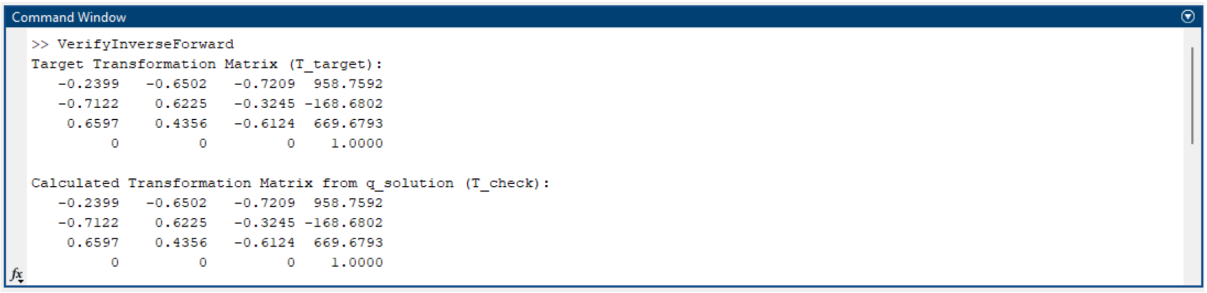
Command Window

```
>> VerifyInverseForward
Target Transformation Matrix (T_target):
   -0.8660   -0.5000         0  648.8576
   -0.2500    0.4330   -0.8660  -13.2739
    0.4330   -0.7500   -0.5000  593.4050
         0         0         0    1.0000

Calculated Transformation Matrix from q_solution (T_check):
   -0.8660   -0.5000    0.0000  648.8557
   -0.2500    0.4330   -0.8660  -13.2733
    0.4330   -0.7500   -0.5000  593.4034
         0         0         0    1.0000
```



Command Window

```
>> VerifyInverseForward
Target Transformation Matrix (T_target):
   -0.0008    0.0006    0.0002    1.4614
    0.0002    0.0005   -0.0008   -0.0772
   -0.0006   -0.0006   -0.0005   -0.4726
         0         0         0    0.0010

Calculated Transformation Matrix from q_solution (T_check):
   1.0e+03 *

   -0.0008    0.0006    0.0002    1.4614
    0.0002    0.0005   -0.0008   -0.0772
   -0.0006   -0.0006   -0.0005   -0.4726
         0         0         0    0.0010
```



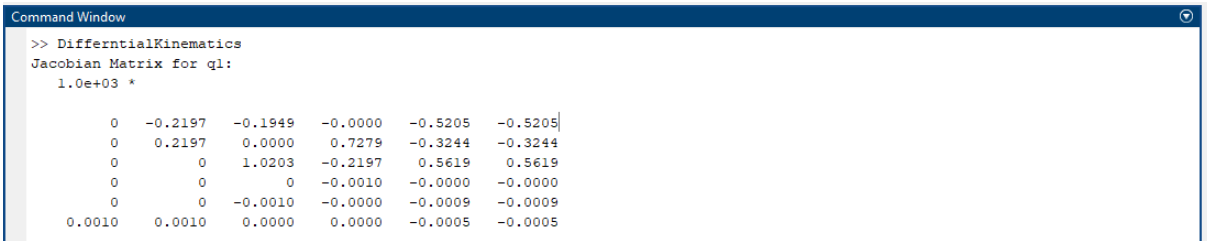
Command Window

```
>> VerifyInverseForward
Target Transformation Matrix (T_target):
   -0.2399   -0.6502   -0.7209  958.7592
   -0.7122    0.6225   -0.3245 -168.6802
    0.6597    0.4356   -0.6124  669.6793
         0         0         0    1.0000

Calculated Transformation Matrix from q_solution (T_check):
   -0.2399   -0.6502   -0.7209  958.7592
   -0.7122    0.6225   -0.3245 -168.6802
    0.6597    0.4356   -0.6124  669.6793
         0         0         0    1.0000
```

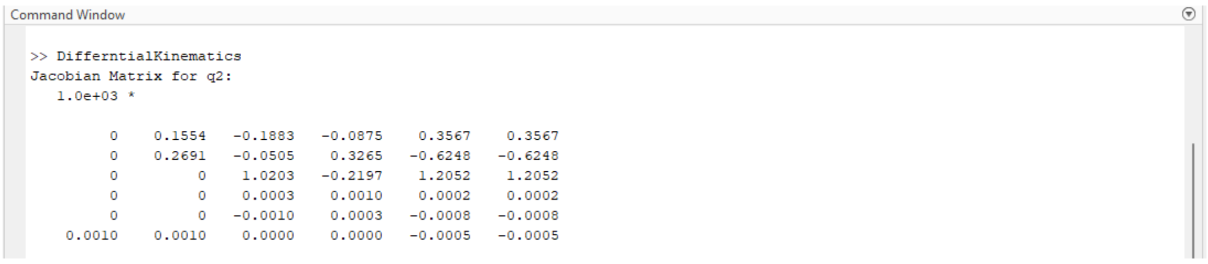
**Differential Kinematics Program:-**

```
% --- Jacobian Calculation ---
% Define joint angles for q1
q1 = [0, pi/4, -pi/4, pi/2, -pi/6, pi/3];
% Calculate the Jacobian matrix for q1
J = Jacob(q1);
% Display the Jacobian matrix for q1
disp('Jacobian Matrix for q1:');
disp(J);
% --- Local function to compute the Jacobian matrix ---
function J = Jacob(q)
   % Number of joints
   num_joints = length(q);
   % DH parameters
   dh_params = [
      0,      0,        194.95, q(1);
      310.77,  0,        0,      q(2);
      800.51,  pi/2,      0,      q(3);
      532.99,  -pi/2,     0,      q(4);
      0,      pi/2,      371.40, q(5);
      0,      0,        269.07, q(6)
   ];
   % Initialize transformation matrices and Jacobian
   T = eye(4); % Initial transformation matrix as identity
   J = zeros(6, num_joints); % Jacobian matrix (6x6 for 6 DOF)
   % Calculate the transformation matrix up to each joint and the Jacobian
   for i = 1:num_joints
      a = dh_params(i, 1);
      alpha = dh_params(i, 2);
      d = dh_params(i, 3);
      theta = dh_params(i, 4);
      % Compute the transformation matrix from i-1 to i
```

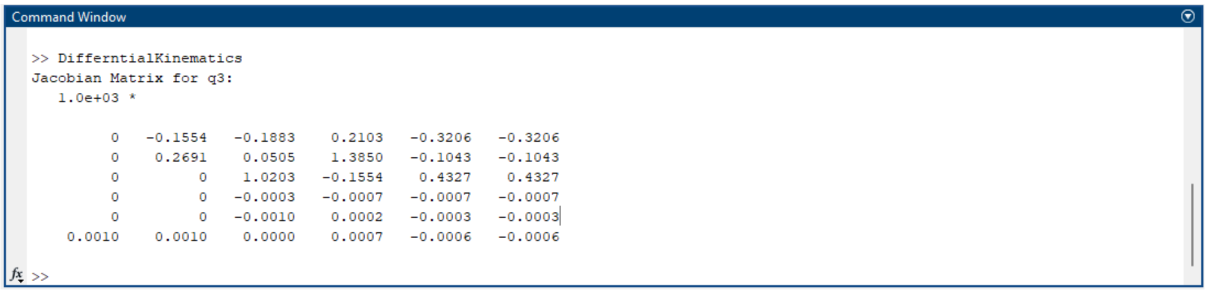
```
        Ti = [cos(theta), -sin(theta)*cos(alpha),  sin(theta)*sin(alpha), a*cos(theta);
              sin(theta),  cos(theta)*cos(alpha), -cos(theta)*sin(alpha), a*sin(theta);
              0,           sin(alpha),             cos(alpha),            d;
              0,           0,                      0,                     1];
        % Update the transformation matrix from base to current joint i
        T = T * Ti;
        % Extract the position vector and z-axis for the current joint
        p = T(1:3, 4);
        z = T(1:3, 3);
        % Compute linear velocity part (position cross z-axis)
        Jv = cross(z, p);
        % Fill the Jacobian matrix (linear and angular parts)
        J(1:3, i) = Jv;  % Linear part
        J(4:6, i) = z;   % Angular part
    end
end
```



```
Command Window
>> DifferntialKinematics
Jacobian Matrix for q1:
   1.0e+03 *

         0   -0.2197   -0.1949   -0.0000   -0.5205   -0.5205
         0    0.2197    0.0000    0.7279   -0.3244   -0.3244
         0         0    1.0203   -0.2197    0.5619    0.5619
         0         0         0   -0.0010   -0.0000   -0.0000
         0         0   -0.0010   -0.0000   -0.0009   -0.0009
    0.0010    0.0010    0.0000    0.0000   -0.0005   -0.0005
```



```
Command Window

>> DifferntialKinematics
Jacobian Matrix for q2:
   1.0e+03 *

         0    0.1554   -0.1883   -0.0875    0.3567    0.3567
         0    0.2691   -0.0505    0.3265   -0.6248   -0.6248
         0         0    1.0203   -0.2197    1.2052    1.2052
         0         0    0.0003    0.0010    0.0002    0.0002
         0         0   -0.0010    0.0003   -0.0008   -0.0008
    0.0010    0.0010    0.0000    0.0000   -0.0005   -0.0005
```



```
Command Window

>> DifferntialKinematics
Jacobian Matrix for q3:
   1.0e+03 *

         0   -0.1554   -0.1883    0.2103   -0.3206   -0.3206
         0    0.2691    0.0505    1.3850   -0.1043   -0.1043
         0         0    1.0203   -0.1554    0.4327    0.4327
         0         0   -0.0003   -0.0007   -0.0007   -0.0007
         0         0   -0.0010    0.0002   -0.0003   -0.0003
    0.0010    0.0010    0.0000    0.0007   -0.0006   -0.0006

>>
```