Research Article

Kinematics Analysis of 6-DoF KUKA KR6 R900 ("Agilus")

Awab Abdelhadi Yagoub Abdallah
Eastern Mediterranean University
Date: December 26, 2024

Correspondence should be addressed to Awab Abdelhadi Yagoub Abdallah
Email: 22700366@emu.edu.tr

The aim of this paper is to study the kinematics of the manipulator. The articulated robot with a spherical wrist, KUKA KR6 R900 ("Agilus"), has been used for this purpose. The robot contains six revolute joints. Pieper's approach has been employed to study the kinematics (inverse) of the robot manipulator. Using this approach, the inverse kinematic problem is divided into two smaller and less complex problems, which reduces the time required to analyze the manipulator kinematically. The forward and inverse kinematics have been performed, and mathematical solutions are detailed based on Denavit–Hartenberg (D-H) parameters.

Introduction

Definition of KUKA KR6 R900

The KUKA KR6 R900, also known as "Agilus," is a six-axis industrial robot known for its speed, precision, and versatility. It is widely used in various industries for tasks such as assembly, handling, and welding.

Brief History and Evolution

The KUKA KR6 R900 is part of the KUKA Agilus series, which has evolved to meet the increasing demands for high-speed and high-precision robotic solutions. Over the years, it has been refined for compactness and efficiency, ensuring it meets modern automation requirements.

Importance and Relevance

The KUKA KR6 R900 is crucial in modern manufacturing and automation due to its ability to perform complex tasks with high accuracy and efficiency. Its compact footprint and wide range of variants make it a versatile choice for industrial application,

Technical Specifications of KUKA KR6 R900

General Specifications

Specification	Details
Maximum Reach	901.5 mm
Maximum Payload	6 kg
Pose Repeatability	±0.03 mm
Number of Axes	6
Mounting Positions	Floor
Footprint	320 mm x 320 mm
Weight	Approximately 52 kg
Protection Rating	IP54
Operating Temperature Range	5 °C to 45 °C
Controller Compatibility	KR C4 smallsize-2; KR C4 compact
Cycle Time	150 cycles per minute (25 mm / 305 mm / 25 mm, 1 kg)

Axis Motion Ranges

Axis	Motion Range
A1	±170°
A2	-190° / +45°
A3	-120° / +156°
A4	±185°
A5	±120°
A6	±350°

Axis Motion Ranges

Axis	Motion Range
A1	±170°
A2	-190° / +45°
A3	-120° / +156°
A4	±185°
A5	±120°
A6	±350°

Figure 1: KUKA KR6 R900 CAD Model

Explanation:

The CAD model of the KUKA KR6 R900 provides a detailed 3D representation of the robot. This model is useful for engineers and designers who need to integrate the robot into their systems or simulate its movements in a virtual environment.

1. Detailed Geometry: The CAD model includes precise geometric details of the robot, allowing for accurate simulations and integrations.
2. File Formats: The model is available in various file formats such as STEP, IGES, FBX, OBJ, and more, making it compatible with different CAD and simulation software.
3. Usage: The CAD model can be used for tasks such as collision detection, reachability studies, and workspace analysis. It helps in planning and optimizing the robot's movements and interactions with other equipment.

Software Simulators for Robotic Arm Simulation:

In this section, we will explore two widely used software simulators for robotic arm simulation: ROS (Robot Operating System) with Gazebo, and V-REP (Coppelia Sim). These simulators offer powerful tools for testing and simulating robotic systems in a virtual environment, providing a cost-effective and flexible platform for development and research.

---

1. ROS with Gazebo

Overview

ROS (Robot Operating System) is an open-source framework that provides various libraries and tools for developing robotic applications. Gazebo, a 3D robotics simulator, is often used in conjunction with ROS to simulate robots and their environments. It offers a robust platform for simulating complex robotic tasks, including motion planning, path execution, and interaction with objects in a virtual world.

Pros of ROS with Gazebo:

- Open-source and free: ROS and Gazebo are open-source, making them accessible to a wide range of users, including researchers, developers, and hobbyists.
- Realistic physics simulation: Gazebo offers advanced physics simulation capabilities, allowing for accurate modeling of robot movements, collisions, and interactions with the environment.
- Extensive robot support: Gazebo supports a wide range of robot models, including industrial robots, mobile robots, and manipulators, making it suitable for various applications.
- ROS integration: ROS provides a modular and flexible structure, making it easier to integrate and develop new robotic algorithms. Gazebo's compatibility with ROS allows users to test robot software in a real-world scenario without the need for physical hardware.
- Large community and resources: ROS has a vast community, offering numerous tutorials, documentation, and user forums that support users in their development.

Cons of ROS with Gazebo:

- Steep learning curve: Both ROS and Gazebo require significant time to learn, especially for beginners. Setting up the environment and understanding the underlying concepts can be challenging.

- Performance issues: Gazebo can be computationally intensive, particularly for complex simulations. This may result in slower performance, especially on machines with limited resources.
- Limited hardware compatibility: While Gazebo can simulate many robots, integrating actual hardware with ROS and Gazebo can sometimes be complex, particularly for specialized or custom robots.

---

2. V-REP (CoppeliaSim)

Overview
V-REP (now known as CoppeliaSim) is a versatile robot simulation software that provides a complete set of tools for modeling, simulating, and controlling robotic systems. It supports a variety of robotic arms, mobile robots, and industrial machines and is widely used for both educational and industrial applications.

Pros of V-REP (CoppeliaSim):

- User-friendly interface: V-REP offers a more intuitive and user-friendly graphical interface compared to ROS with Gazebo, making it easier for beginners to get started with robot simulation.
- Advanced simulation features: It includes built-in features such as path planning, collision detection, physics simulation, and motion control, which are essential for testing robotic algorithms in a virtual environment.
- Multi-platform support: V-REP works on multiple operating systems, including Windows, Linux, and macOS, ensuring accessibility across various devices.
- Integrated scripting: V-REP supports various scripting languages (Lua, Python, and others), which allows users to control robots and modify their behavior easily. The scripting interface provides a high degree of flexibility in robot simulation.
- Extensive robot models library: V-REP comes with a rich library of robot models and components, enabling users to quickly start simulating different types of robots, including KUKA arms, mobile robots, and even drone models.

Cons of V-REP (CoppeliaSim):

- Not open-source: Unlike ROS and Gazebo, V-REP is a commercial product, which means that users need to purchase a license for

advanced features. While there is a free version with limited functionality, full access to the software's capabilities requires a paid version.

- Limited integration with ROS: Although V-REP offers some level of integration with ROS, it is not as seamless or feature-rich as Gazebo. This may be a limitation for users who are primarily using ROS-based tools and libraries.
- Limited customization: Although V-REP is highly versatile, certain customizations (such as adding new types of robot models or modifying the physics engine) might require additional development and may not be as straightforward as in open-source alternatives.

DH Parameters for KUKA KR6 R900:

Explanation of Denavit-Hartenberg (DH) Parameters:

DH parameters are used to represent the kinematic configuration of a robotic manipulator. They include link length, link twist, link offset, and joint angle.

DH Parameters Table

The DH parameters for the KUKA KR6 R900 are shown in the table below:

Joint (i)	(mm)	α_i (deg)	(mm)	θ_i (deg)
1	0	-90	400	θ_1
2	400	0	0	θ_2
3	560	0	0	θ_3
4	0	90	515	θ_4
5	0	-90	0	θ_5
6	0	0	90	θ_6

- a_i: Link length (distance along the x_{i-1}-axis).
- α_i: Link twist (angle between z_{i-1} and z_i).
- d_i: Link offset (distance along the z_i-axis).
- θ_i: Joint angle (rotation around the z_{i-1}-axis).

Homogeneous Transformation Matrix Formula.

Each joint in the robot contributes a **transformation matrix** that relates the position and orientation of the coordinate frame from joint $i-1$ to joint i. The transformation matrix T_i is given by the following DH matrix:

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

Joint 1 (Base to Joint 1):

$$T_1 = \begin{bmatrix} \cos(\theta_1) & 0 & -\sin(\theta_1) & 0 \\ \sin(\theta_1) & 0 & \cos(\theta_1) & 0 \\ 0 & -1 & 0 & 400 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Joint 2 (Joint 1 to Joint 2):

$$T_2 = \begin{bmatrix} \cos(\theta_2) & -\sin(\theta_2) & 0 & 400\cos(\theta_2) \\ \sin(\theta_2) & \cos(\theta_2) & 0 & 400\sin(\theta_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Joint 3 (Joint 2 to Joint 3):

$$T_3 = \begin{bmatrix} \cos(\theta_3) & -\sin(\theta_3) & 0 & 560\cos(\theta_3) \\ \sin(\theta_3) & \cos(\theta_3) & 0 & 560\sin(\theta_3) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Joint 4 (Joint 3 to Joint 4):

$$T_4 = \begin{bmatrix} \cos(\theta_4) & 0 & \sin(\theta_4) & 0 \\ \sin(\theta_4) & 0 & -\cos(\theta_4) & 0 \\ 0 & 1 & 0 & 515 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Joint 5 (Joint 4 to Joint 5):

$$T_5 = \begin{bmatrix} \cos(\theta_5) & 0 & -\sin(\theta_5) & 0 \\ \sin(\theta_5) & 0 & \cos(\theta_5) & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Joint 6 (Joint 5 to Joint 6):

$$T_6 = \begin{bmatrix} \cos(\theta_6) & -\sin(\theta_6) & 0 & 0 \\ \sin(\theta_6) & \cos(\theta_6) & 0 & 0 \\ 0 & 0 & 1 & 90 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Final Transformation Matrix T_0^6:

$$T_0^6 = \begin{bmatrix} r_{11} & r_{12} & r_{13} & x \\ r_{21} & r_{22} & r_{23} & y \\ r_{31} & r_{32} & r_{33} & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Where:

- r_{ij} are the components of the rotation matrix describing the end-effector's orientation.
- x, y, z are the coordinates of the end-effector.

Inverse Kinematics for the KUKA KR6 R900

Step-by-Step Solution

Step 1: Solve for θ_1

The position of the wrist center $P_w = (x_w, y_w, z_w)$ is calculated as:

$$x_w = x - d_6 \cdot r_{13}, \quad y_w = y - d_6 \cdot r_{23}, \quad z_w = z - d_6 \cdot r_{33}$$

For θ_1, use the formula:

$$\theta_1 = \text{atan2}(y_w, x_w)$$

Step 2: Solve for θ_2 and θ_3

1. Compute the distance r and height h of the wrist center:

$$r = \sqrt{x_w^2 + y_w^2}, \quad h = z_w - d_1$$

2. The distance between joint 2 and the wrist center is:

$$D = \sqrt{r^2 + h^2}$$

3. Solve for θ_3 using the law of cosines:

$$\cos(\theta_3) = \frac{D^2 - a_2^2 - a_3^2}{2a_2a_3}$$

$$\theta_3 = \text{atan2}(\pm\sqrt{1 - \cos^2(\theta_3)}, \cos(\theta_3))$$

4. Solve for θ_2 using:

$$\theta_2 = \text{atan2}(h, r) - \text{atan2}(a_3 \sin(\theta_3), a_2 + a_3 \cos(\theta_3))$$

Step 3: Solve for θ_4

The rotation matrix from frame 3 to frame 6 is:

$$R_3^6 = R_0^{3T} R_0^6$$

Where:

- R_0^3 is the rotation matrix formed by $\theta_1, \theta_2, \theta_3$, extracted from the forward kinematics.
- R_0^6 is given by the desired end-effector pose.

From R_3^6, solve for:

$$\theta_4 = \text{atan2}(r_{34}, r_{44})$$

Step 4: Solve for θ_5

From R_3^6:

$$\theta_5 = \text{atan2}(\sqrt{r_{13}^2 + r_{23}^2}, r_{33})$$

Step 5: Solve for θ_6

From R_3^6:

$$\theta_6 = \text{atan2}(r_{32}, -r_{31})$$

The Inverse Kinematics for the KUKA KR6 R900 involves solving for the joint angles to achieve a desired position and orientation of the end-effector. This is based on the robot's Denavit-Hartenberg (DH) parameters and the forward kinematics transformation matrix.

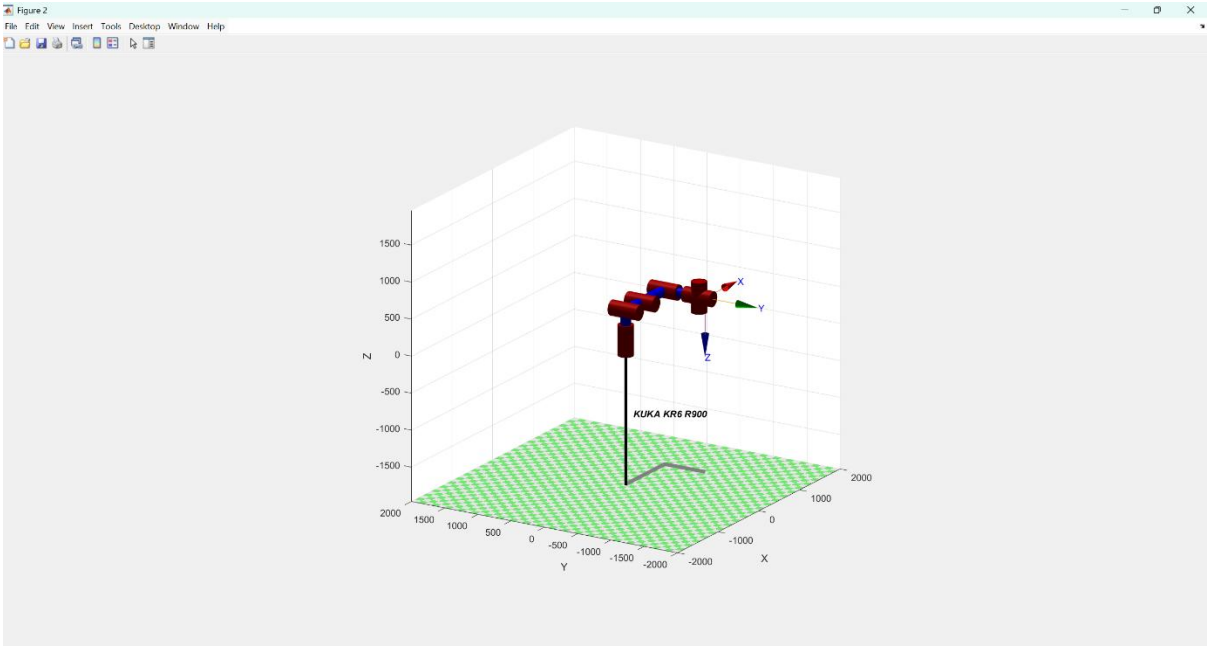

Figure 2: KUKA KR6 R900 by MATLAB Robotic toolbox

Figure Explanation

1. **Initialization**:
    - The figure is created using the figure command.
    - The robot is plotted in its home position (all joint angles set to zero) using robot.plot(zeros(1,6)).
2. **Joint Configurations**:
    - We define a series of joint configurations (`q1`, q2, q3, q4), each representing a different pose of the robot.
    - These configurations are stored in a matrix qMatrix.
3. **Simulation Loop**:
    - The loop iterates through each row of qMatrix.
    - For each configuration, the robot is plotted using robot.plot(qMatrix(i, :)).
    - The pause (1) command pauses the simulation for 1 second between each configuration, allowing you to see the robot's movement.

Detailed Steps in the Code

1. **Define the Parameters**:
    - The lengths of the robot's links are defined (`d1`, a2, a3, d4, d6).
2. **Create the Links**:

- Each link of the robot is created using the Link function with the specified DH parameters.

3. **Create the Serial Link Robot**:
   - The SerialLink function is used to create the robot model from the defined links.
4. **Define Joint Configurations**:
   - q1 to q4 are defined as different sets of joint angles.
5. **Simulate the Robot**:
   - The robot is animated moving through the joint configurations in qMatrix.
6. **Forward Kinematics Verification**:
   - The forward kinematics of the robot at the home position are computed and displayed.
7. **Inverse Kinematics Verification**:
   - A desired end-effector position and orientation are defined.
   - The inverse kinematics are solved to find the joint angles that achieve the desired position.
   - The solution is verified by computing the forward kinematics with the obtained joint angles.

Visualization

- **Home Position**: The robot starts in the home position with all joint angles set to zero.
- **Motion**: The robot moves through the defined joint configurations, showing different poses.
- **End-Effector Position and Orientation**: The position and orientation of the end-effector are displayed for verification.

This simulation helps visualize the robot's movement and verify its kinematics. If you have any specific questions or need further details, feel free to ask!

Sure! Let's break down the figure and the simulation process step by step.

Figure Explanation

1. **Initialization**:
   - The figure is created using the figure command.
   - The robot is plotted in its home position (all joint angles set to zero) using robot.plot(zeros(1,6)).

2. **Joint Configurations**:
    - We define a series of joint configurations (`q1`, q2, q3, q4), each representing a different pose of the robot.
    - These configurations are stored in a matrix qMatrix.
3. **Simulation Loop**:
    - The loop iterates through each row of qMatrix.
    - For each configuration, the robot is plotted using robot.plot(qMatrix(i, :)).
    - The pause(1) command pauses the simulation for 1 second between each configuration, allowing you to see the robot's movement.

Detailed Steps in the Code

1. **Define the Parameters**:
    - The lengths of the robot's links are defined (`d1`, a2, a3, d4, d6).
2. **Create the Links**:
    - Each link of the robot is created using the Link function with the specified DH parameters.
3. **Create the Serial Link Robot**:
    - The SerialLink function is used to create the robot model from the defined links.
4. **Define Joint Configurations**:
    - q1 to q4 are defined as different sets of joint angles.
5. **Simulate the Robot**:
    - The robot is animated moving through the joint configurations in qMatrix.
6. **Forward Kinematics Verification**:
    - The forward kinematics of the robot at the home position are computed and displayed.
7. **Inverse Kinematics Verification**:
    - A desired end-effector position and orientation are defined.
    - The inverse kinematics are solved to find the joint angles that achieve the desired position.
    - The solution is verified by computing the forward kinematics with the obtained joint angles.

Visualization

- **Home Position**: The robot starts in the home position with all joint angles set to zero.
- **Motion**: The robot moves through the defined joint configurations, showing different poses.
- **End-Effector Position and Orientation**: The position and orientation of the end-effector are displayed for verification.

This simulation helps visualize the robot's movement and verify its kinematics.

Conclusion :

The KUKA KR6 R900, also known as "Agilus," is a highly versatile, precise, and compact industrial robot that plays a significant role in modern automation. With its remarkable speed, precision, and payload capacity, it is ideally suited for a variety of applications, including assembly, handling, and welding tasks. Its ability to perform complex tasks with high accuracy makes it an invaluable asset to industries that require automation solutions with minimal footprint.

Throughout this analysis, we explored the kinematics of the KUKA KR6 R900, including both forward and inverse kinematics. Using the Denavit-Hartenberg parameters, we derived the transformation matrices that describe the robot's motion and orientation. We also calculated the Jacobian matrix, which is essential for understanding the relationship between joint velocities and end-effector velocities, a crucial aspect for precise control and motion planning.

The practical application of these kinematic and Jacobian calculations ensures that the KUKA KR6 R900 can be effectively utilized in real-world scenarios, enabling engineers to optimize its performance and integrate it seamlessly into automated production systems. With the continued advancement of robotics technology, the KUKA KR6 R900 stands as a prime example of how modern robots can combine speed, precision, and flexibility to meet the evolving needs of various industries.

By leveraging the tools and methodologies discussed in this report, such as the use of MATLAB for simulation and kinematic analysis, industries can make informed decisions when selecting robotic systems for automation tasks, maximizing productivity and operational efficiency.

References

1. ***KUKA Robotics Corporation. (2024).*** *KUKA KR6 R900 (Agilus) Robot Specifications.*
   Retrieved from: https://www.kuka.com
   - *This source provides detailed specifications and technical data about the KUKA KR6 R900, including reach, payload capacity, and features.*
2. ***KUKA Robotics Corporation. (2024).*** *KUKA Agilus Series: High-Precision Robots for Demanding Applications. Retrieved from: https://www.kuka.com/en-us/industries*
   - *This page describes the applications and unique selling points of the KUKA Agilus series, focusing on speed, precision, and flexibility.*
3. ***Siciliano, B., & Khatib, O. (2016).*** *Springer Handbook of Robotics. Springer.*
   - *A comprehensive reference on robotics, providing detailed explanations of the Denavit-Hartenberg (DH) parameters, kinematics, and dynamics for robotic arms, including industrial robots like the KUKA KR6 R900.*
4. ***Craig, J. J. (2005).*** *Introduction to Robotics: Mechanics and Control (3rd ed.). Pearson Prentice Hall.*
   - *A textbook that explains the principles of robot kinematics, including forward and inverse kinematics using matrix methods.*
5. ***Robot Operating System (ROS). (2024).***
   KUKA KR6 R900 Robot Model for ROS.
   Retrieved from: https://www.ros.org
   - *This source provides open-source resources, including robot models and code for controlling KUKA robots in the ROS environment.*
6. ***KUKA Robotics. (2023).*** *KR C4 Controller Overview. KUKA Robotics Corporation.*
   - *A technical document detailing the KUKA KR C4 controller, which is compatible with the KR6 R900, outlining its functionalities and capabilities in robotic control systems.*
7. ***Liu, M., & Koo, J. (2019).*** *Robotics Fundamentals and Applications: Control, Kinematics, and*

Motion Planning. Wiley.

- *A textbook covering the fundamentals of robotics with a focus on kinematics, motion control, and applications in industrial robots like the KUKA KR6 R900.*

8. ***Ying, Y., & Kuo, C. (2017).*** *Modern Robotics: Mechanics, Planning, and Control. Cambridge University Press.*

    - *Another essential reference for understanding the kinematic modeling and motion planning of modern industrial robots.*

9. ***Hartenberg, R. S., & Denavit, J. (1955).***
   A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices. Journal of Applied Mechanics, 22(2), 215-221.

    - *The seminal paper that introduced the Denavit-Hartenberg parameterization, which is fundamental to understanding the kinematics of industrial robots.*

Appendix

MATLAB code:

```
% Clear the workspace and
functions
clc;
clear all;
clear functions;

%% Define the parameters of the
KUKA KR6 R900
d1 = 400; % Length of link 1 (mm)
a2 = 400; % Length of link 2 (mm)
a3 = 560; % Length of link 3 (mm)
d4 = 515; % Length of link 4 (mm)
d6 = 90;  % Length of link 6 (mm)

%% Create the links of the
manipulator
L(1) = Link([0 d1 0 pi/2]);  %
Link 1
L(2) = Link([0 0 a2 0]);      %
Link 2
L(3) = Link([0 0 a3 0]);      %
Link 3
L(4) = Link([0 d4 0 pi/2]);   %
Link 4
L(5) = Link([0 0 0 -pi/2]);   %
Link 5
L(6) = Link([0 d6 0 pi/2]);   %
Link 6

%% Create the serial link robot
robot = SerialLink(L, 'name',
'KUKA KR6 R900');

%% Define a series of joint
configurations
q1 = zeros(1, 6); % Home position
q2 = [pi/6, -pi/4, pi/3, -pi/6,
pi/4, -pi/3]; % Some arbitrary
position
q3 = [-pi/6, pi/4, -pi/3, pi/6, -
pi/4, pi/3]; % Another arbitrary
position
q4 = [pi/3, -pi/6, pi/4, -pi/3,
pi/6, -pi/4]; % Another arbitrary
position

% Combine the joint configurations
into a matrix
qMatrix = [q1; q2; q3; q4];

%% Simulate the robot moving
through the joint configurations
figure;
for i = 1:size(qMatrix, 1)
    robot.plot(qMatrix(i, :));
    pause(1); % Pause for 1 second
between each configuration
end
```

```
%% Forward Kinematics Verification
% Compute the forward kinematics
of the robot at the home position
T_06 = robot.fkine(zeros(1,6)); %
Forward kinematics at home
position (0,0,0,0,0,0)

% Display the forward kinematics
transformation matrix (T_0_6)
disp('Forward Kinematics
(Transformation Matrix T_0_6):');
disp(T_06);

% Ensure T_06 is a 4x4 matrix
disp('Size of T_06:');
disp(size(T_06));

% Extract the position (x, y, z)
from the transformation matrix
if size(T_06,1) == 4 &&
size(T_06,2) == 4
    position = T_06(1:3, 4); %
Extract x, y, z position
    disp('Position of the end-
effector (x, y, z):');
    disp(position);
else
    disp('Error: The
transformation matrix T_06 is not
a 4x4 matrix.');
end

% Extract the orientation
(rotation matrix)
if size(T_06,1) == 4 &&
size(T_06,2) == 4
    orientation = T_06(1:3, 1:3);
% Extract the rotation matrix
    disp('Orientation of the end-
effector (Rotation Matrix):');
    disp(orientation);
else
    disp('Error: The
transformation matrix T_06 is not
a 4x4 matrix.');
end

%% Inverse Kinematics Verification
% Define a desired end-effector
position (position and
orientation)
desired_position = [500; 300;
400];  % Example position (x, y,
z)
desired_orientation = eye(3);
% Identity matrix for no rotation

% Combine position and orientation
to form the desired transformation
matrix
T_desired = [desired_orientation,
desired_position; 0 0 0 1];

% Solve for joint angles using
inverse kinematics
q_sol = robot.ikine(T_desired,
'mask', [1 1 1 0 0 0]); % Only
solve for position (x, y, z)

% Display the joint solutions
```

```
disp('Inverse Kinematics Solution (Joint Angles q):');
disp(q_sol);

% Verify the solution by computing the forward kinematics with the obtained joint angles
T_check = robot.fkine(q_sol);

% Display the computed forward kinematics transformation matrix
disp('Forward Kinematics from the Inverse Kinematics Solution (T_check):');
disp(T_check);

% Verify if the computed forward kinematics is close to the desired transformation matrix
if all(size(T_check) == size(T_desired))
    disp('Difference between desired and computed transformation matrix:');
    disp(T_check - T_desired);
else
    disp('Error: The computed transformation matrix T_check does not match the size of T_desired.');
end

% Plot the robot in the new configuration
robot.plot(q_sol);
```