
# ME5402/EE5106 ADVANCED ROBOTICS

## Part 1 CA Robot Simulation

*Submitted by Homework Group 30*

AY2023/2024 (SEMESTER 2)

12th, Apr.,2024

# *Content*

## *List of Figures*

# Introduction

In the realm of advanced robotics, the manipulation and precise control of robotic arms using kinematic theories and simulations represent a foundational aspect of robotics research and practical application. This report details a comprehensive study and simulation undertaken as part of the coursework for ME 5402/EE 5106 Advanced Robotics during the AY23/24 Semester 2. The focus is predominantly on the Universal Robot UR5e, a versatile and widely used robotic arm known for its accuracy and ease of integration into various industrial processes.

The project's primary objective is to model the UR5e manipulator using classical D-H (Denavit-Hartenberg) representation and to derive its forward and inverse kinematics. This involves not just a theoretical understanding of the manipulator's architecture but also practical simulation tasks designed to mimic complex real-world operations. One such simulation involves defining an upright 3D plane that resembles the back of a human body and programming the robot to perform precise needle insertions at predetermined points on this plane.

Throughout this report, we will navigate through our methodologies in setting up the simulation, the challenges faced, and the solutions derived to accomplish the assigned tasks. This endeavor is not only a testament to our understanding of robotic kinematics but also to our ability to translate theoretical concepts into practical, executable simulations that mirror potential real-life applications. By the conclusion of this report, we aim to demonstrate a robust model of the UR5e manipulator executing complex maneuvers with high precision and reliability, as expected in both academic and industrial settings.

# Chapter 1 Literature Review of UR5e

Manipulators are essential tools in modern manufacturing, playing a crucial role in automating various tasks such as assembly, welding, and material handling. These versatile machines come in different configurations, ranging from simple robotic arms to complex multi-axis manipulators capable of intricate movements and precise operations. They are designed to enhance productivity, improve quality, and ensure safety in manufacturing processes. With advancements in robotics and automation technology, manipulators continue to evolve, offering increased flexibility, efficiency, and integration capabilities in manufacturing environments.

## History of manipulators

The history of manipulators dates back to the 1950s, when mechanical devices were employed to perform repetitive tasks. As shown in Fig. 1, UNIMATE is the world's first industrial robot, co-developed by American engineers George Devol and Joseph Engelberger. Early manipulators were primarily mechanical systems powered by hydraulic or pneumatic actuators. These rudimentary devices were limited in functionality and required constant human supervision. Over time, advancements in technology, particularly in robotics and automation, have propelled the evolution of manipulators. With the advent of electric motors and computer control, manipulators became more sophisticated, capable of performing complex tasks with higher precision and efficiency.


***Fig. 1 The First Industrial Robot "UNIMATE"***

The development of manipulators can be divided into the following stages chronologically:

- 1950s: Early Robotic Arms

The first digitally operated industrial robot was created in 1954, known as the UNIMATE. This robot was used on assembly lines in the automotive industry, marking the birth of industrial robotics.

- 1960s: Rise of Industrial Robots

In 1961, American engineer Joseph Engelberger, in collaboration with George Devol, established the Unimation company to commercialize industrial robots. The following year, the world's first industrial robot, UNIMATE, was deployed on an assembly line in an American automotive plant for tasks such as parts handling and assembly.

- 1970s: Advancements in Industrial Robot Technology

Unimation company developed the Programmable Universal Machine for Assembly (PUMA) robot in 1973, the first successful commercialized industrial robot widely used in automotive manufacturing and other industries. By the mid-1970s, industrial robots began transitioning from hard-wired controls to computerized control systems, enhancing flexibility and precision.

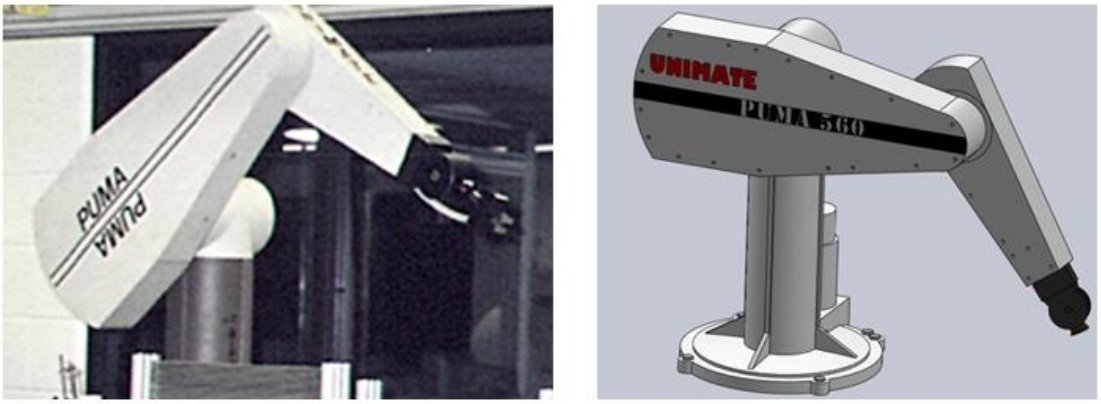


***Fig. 2 UNIMATE PUMA Robot***

- 1980s: Introduction of Machine Vision and Flexible Manufacturing

In 1981, advancements in machine vision technology allowed industrial robots to perceive their environment through visual systems, enabling more complex tasks. In the mid-1980s, Flexible Manufacturing Systems (FMS) emerged, integrating robots, computers, and automation equipment for flexible production line configurations and scheduling.

- 1990s: Emergence of Flexible Robots

In 1997, Stanford University developed flexible robotic arms controlled by pneumatic muscles, exhibiting flexibility and precision similar to human arms, suitable for applications in healthcare and other fields.

- 2000s: Development of Collaborative Robots

Danish company Universal Robots introduced the first collaborative industrial robot in 2008, named UR5, leading the trend of collaborative robotics. These robots could work alongside human operators, enhancing flexibility and safety in industrial production.

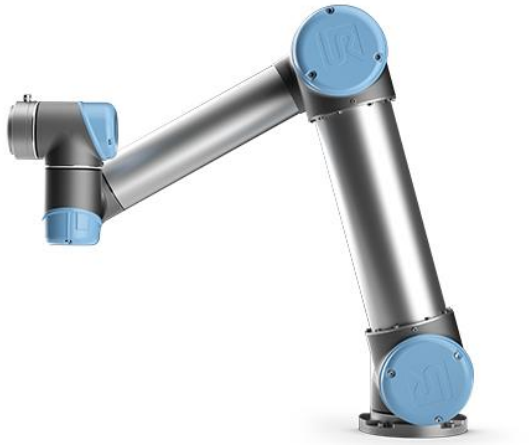

***Fig. 3 UR5 Robot***

- 2010s: Proliferation of Robot Technology

Universal Robots launched UR5e and UR10e in 2012, with higher precision and flexibility, suitable for a wider range of industrial applications. And then in 2014, Amazon introduced large-scale deployment of robots in its warehouses, improving automation in warehousing and logistics.

- 2020s: Advancements in Intelligence and Adaptive Manufacturing

In recent years, with advancements in artificial intelligence and machine learning, industrial robots began to exhibit greater autonomy and adaptability, capable of real-time adjustments and optimizations based on environmental and task requirements. Moreover, robots found increasing applications in the healthcare sector, including surgical robots and rehabilitation robots, providing new solutions for healthcare.

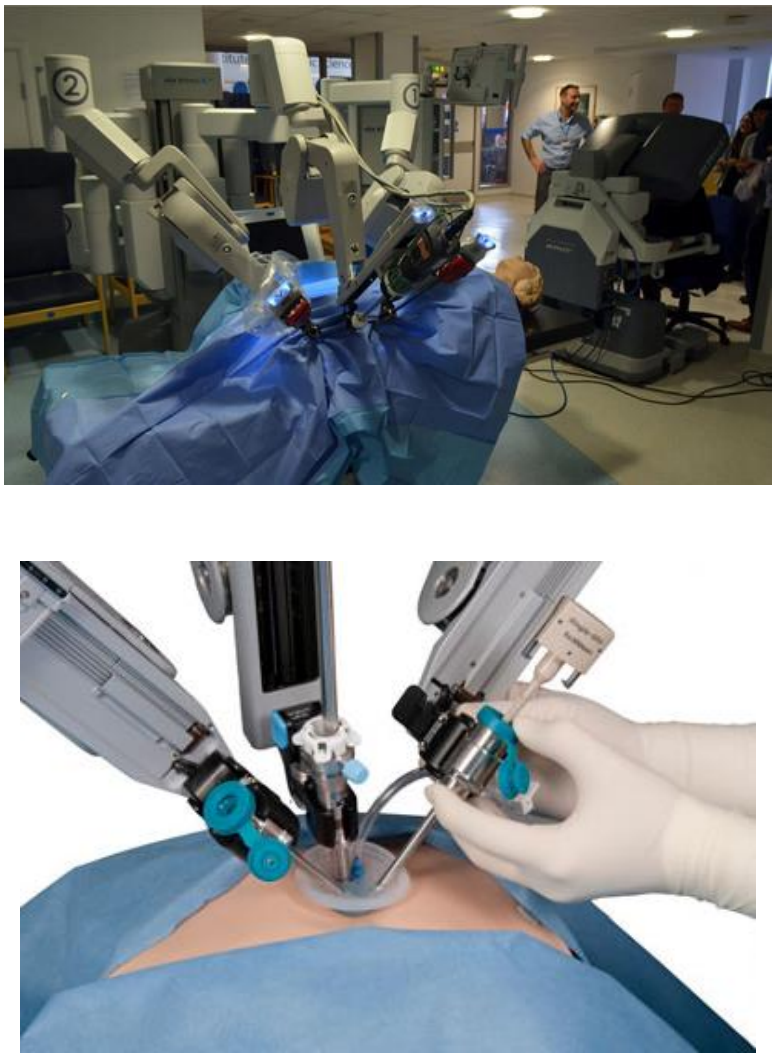

***Fig. 4 Robot-Assisted Surgery***

Some Key Characteristics of Manipulators:

- **Degrees of Freedom (DOF):**

The degrees of freedom of a manipulator refer to the number of independent movements it can perform. This parameter determines the manipulator's flexibility and versatility. Manipulators with higher DOF can access a broader range of positions and orientations, allowing them to handle a wider variety of tasks.

- **Workspace:**

The workspace of a manipulator refers to the volume of space within which the end-effector can reach. The size and shape of the workspace are critical factors in

determining the manipulator's suitability for specific applications. Manipulators with larger workspaces can handle larger objects and accommodate a wider range of tasks.

- **Payload Capacity:**

The payload capacity of a manipulator refers to the maximum weight it can lift and manipulate. Higher payload capacity enables the manipulator to handle heavier objects, expanding its range of applications. Payload capacity is a crucial consideration in industries such as manufacturing, where large and heavy objects need to be manipulated.

- **Accuracy and Repeatability:**

Precision and repeatability are essential characteristics of manipulators, particularly in applications requiring precise positioning and manipulation. Manipulators with high accuracy and repeatability ensure consistent performance, leading to improved productivity and quality of output.

- **Control System:**

The control system of a manipulator determines its autonomy, responsiveness, and adaptability. Advanced control algorithms enable manipulators to perform complex tasks autonomously while responding to changes in the environment or task requirements. Control systems play a crucial role in optimizing the performance of manipulators and ensuring safe and efficient operation.

Industrial robots have revolutionized manufacturing processes by offering unparalleled precision, efficiency, and versatility. Among these, the Universal Robots UR5e robotic arm (as shown in Fig. ) stands out as a pioneering solution in the realm of collaborative robotics. Engineered by Universal Robots, the UR5e embodies cutting-edge technology designed to seamlessly integrate into various industrial environments, working alongside human operators to enhance productivity and safety.

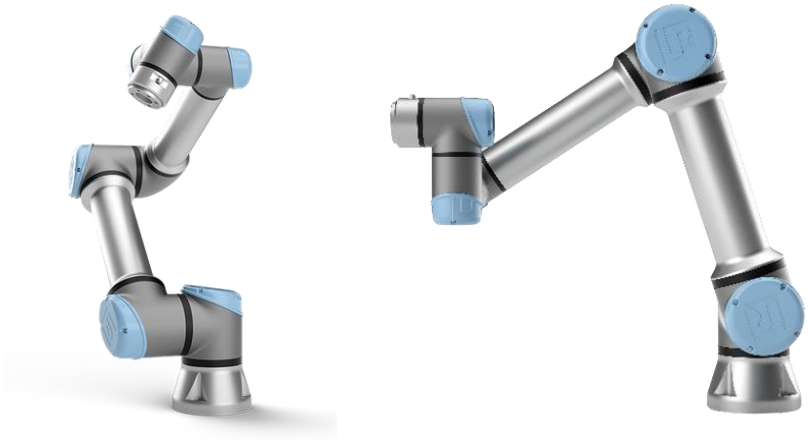

***Fig. 5 UR5e Robot***

The Universal Robots UR5e robotic arm is equipped with a plethora of performance features, ensuring optimal functionality across various industrial applications. With a payload capacity of 5 kg (11 lbs) at full Center of Gravity (CoG) offset and throughout its entire workspace, the UR5e offers versatility in handling objects of different sizes and weights. Its impressive reach of 850 mm (33.5 in) provides ample coverage within the workspace, facilitating efficient task execution.

Featuring 6 rotating joints, the UR5e offers exceptional degrees of freedom, allowing for precise and flexible movement. Its power consumption is rated at 570 W, with moderate operating settings consuming only 200 W, optimizing energy efficiency without compromising performance. Additionally, the UR5e boasts a comprehensive working range of its base, shoulder, elbow, and all three wrist joints, each capable of rotating a full 360°. Furthermore, it achieves maximum speeds of ±180° across all joints, ensuring rapid and agile motion for enhanced productivity[1].

The Universal Robots UR5e robotic arm is a versatile collaborative robot with a wide range of applications and unique features. The UR5e is widely used in manufacturing, logistics and warehousing, healthcare, and other industries, providing efficient and flexible automation solutions for businesses. Fig.  shows its industrial application in glue or paint dispensing and cargo sorting.

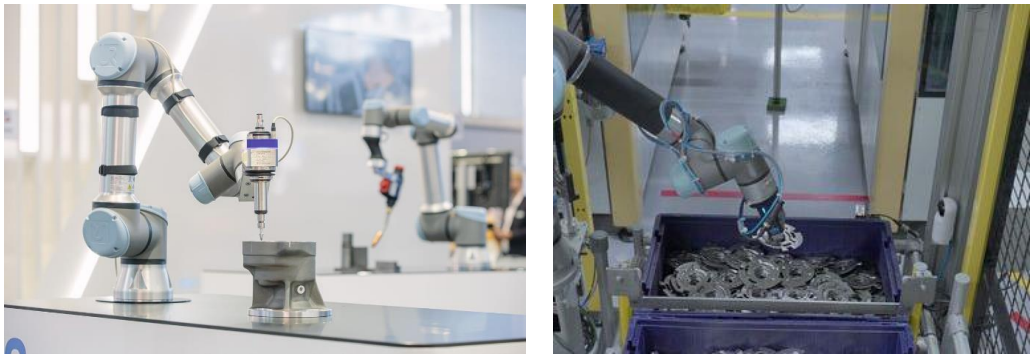

***Fig. 6 Industrial Applications of UR5e***

Key features of the UR5e include flexibility, safety, ease of deployment, efficiency, and versatility. It features 6 rotating joints, enabling multi-axis freedom of movement suitable for various complex work scenarios. Equipped with advanced safety features including torque sensors and a collaborative safety system, the UR5e ensures safe collaboration with human operators.

With a simple and intuitive programming interface, the UR5e is easy to deploy and operate without the need for specialized programming knowledge. With fast movements and precise positioning, it enhances production efficiency and reduces production costs. The UR5e can be applied to a variety of tasks including assembly, picking and placing, packaging, quality inspection, and more, making it suitable for different industries and work scenarios.

# Chapter 2 Modeling of UR5e Manipulator

For task2, we utilized the Robotics System Toolbox in MATLAB to construct the UR5e manipulator model, employing code based on the classical Denavit-Hartenberg (D-H) representation. It is a systematic method to representing kinematic relationship between two adjacent links of robotic arm. First and foremost, the D-H parameters of the UR5e are essential, and they are obtainable from its official website [2]. These parameters define the transformation between coordinate frames of each link, including Joint Angle (θ), link length (a), Link Offset (d) and Link Twist (α).

Link Length (a) represents the length of the link, specifically the distance between adjacent joint axes along the axis of rotation of the preceding joint. Link Twist (α) indicates the angle of rotation of the link around the axis of rotation of the preceding joint. Typically, it is the angle of rotation from the vertical position around the axis of rotation of the preceding joint. Link Offset (d) denotes the displacement along the axis of rotation of the preceding link to the axis of rotation of the current link, essentially the translation along the axis of rotation. Joint Angle (θ) represents the rotation angle of the current link relative to the preceding link around the axis of rotation of the current link, defining the rotation of the link. The UR5e robotic arm comprises six joints, each with corresponding D-H parameters as listed in Table 1.

Table 1 D-H Parameters of UR5e

| **Kinematics** | θ [rad] | a [m] | d [m] | α [rad] |
|---|---|---|---|---|
| Joint 1 | 0 | 0 | 0.1625 | π/2 |
| Joint 2 | 0 | -0.425 | 0 | 0 |
| Joint 3 | 0 | -0.3922 | 0 | 0 |
| Joint 4 | 0 | 0 | 0.1333 | π/2 |
| Joint 5 | 0 | 0 | 0.0997 | -π/2 |
| Joint 6 | 0 | 0 | 0.0996 | 0 |

The manipulator model is built following these steps:

(1) Specifying UR5e parameters:

Using the parameter listed in Table 1, we define a D-H parameter matrix named "dhparams" as follows:

```
% Specify the parameters for the UR5e as a matrix.
a = [0, -0.42500, -0.3922, 0, 0, 0]';
d = [0.1625, 0, 0, 0.1333, 0.0997, 0.0996]';
alpha = [pi/2, 0, 0, pi/2, -pi/2, 0]';
theta = zeros(6,1);
dhparams = [a,alpha,d,theta];
```

(2) Creating rigid body tree object:

Use the "rigidBodyTree" function to create a rigid body tree object, which represents the structure of the UR5e robotic arm.

```
% Create a rigid body tree object.
robot = rigidBodyTree;
```

(3) Adding D-H parameters to the robot:

Use a for loop to add DH parameters to each joint of the robot model. Create rigid body and joint objects and sets the fixed transformation for each joint using the "setFixedTransform" function.

(4) Adding bodies to the tree:

Adds each body to the robot model, connecting it to the previous body or the base.

```
% Add DH parameters into robot
bodies = cell(6,1);
joints = cell(6,1);
for i = 1:6
    bodies{i} = rigidBody(['body' num2str(i)]);
```

```
    joints{i} = rigidBodyJoint(['jnt' num2str(i)],"revolute");
    setFixedTransform(joints{i},dhparams(i,:),"dh");
    bodies{i}.Joint = joints{i};
    if i == 1   % Add first body to base
        addBody(robot,bodies{i},"base")
    else   % Add current body to previous body by name
        addBody(robot,bodies{i},bodies{i-1}.Name)
    end
end
```

(5) Displaying the robot model:

Use “showdetails” and “show” functions to display the model.

```
% Show robot model
showdetails(robot)
show(robot);
title("original_UR5e Robot Model");
disp(bodies)
```

Execute the code, and the manipulator model utilizing the classical D-H representation of UR5e is depicted in Fig. .

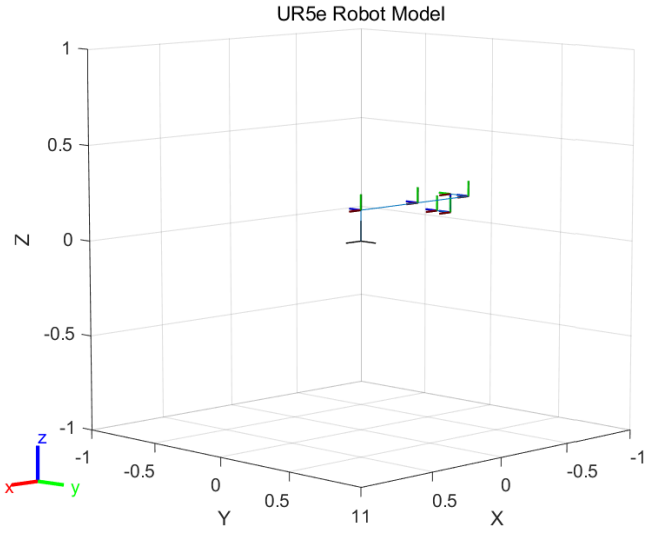


***Fig. 7 Original UR5e Robot Model***

In order to make the model more concrete, we loaded the built-in UR5e model in MATLAB. The reconstructed model is shown in Fig.5.

```
% Load
openExample('urseries/MotionPlanningRBTSimulationUR5eBinPickingManipulatorRRTExample');
ur5eRBT = loadrobot('universalUR5e','DataFormat','row');
ur5e = exampleHelperAddGripper(ur5eRBT);
```

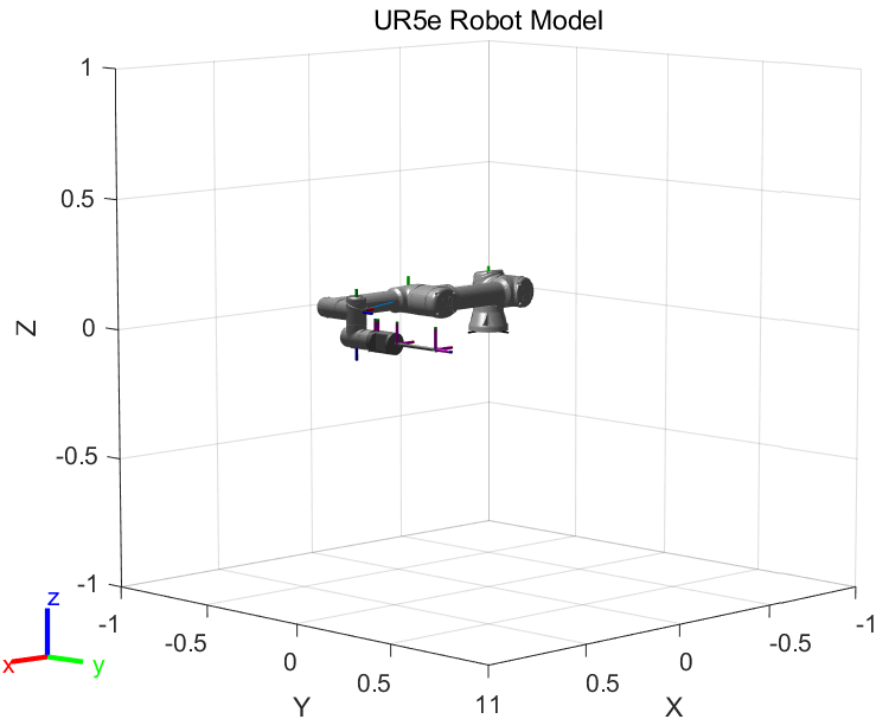


***Fig. 8 UR5e Robot Model***

# Chapter 3 Forward Kinematics Analysis

Forward kinematics focuses on determining the position and orientation of the end-effector of a robotic arm given the joint angles of each joint. Specifically, given the positions of each joint (joint angles or displacements), the lengths of the links, and the initial pose of the robot, the objective is to calculate the transformation matrix of the end-effector relative to the base coordinate system.

The pose of each link relative to the preceding one can be accurately represented using transformation matrices. These matrices are commonly derived from D-H parameters, providing a comprehensive description of the geometric relationships within the robotic arm. Specifically, for the ith link, its transformation matrix is denoted as formula (1). By meticulously multiplying all transformation matrices together, we derive the comprehensive forward kinematics transformation matrix of the robotic arm, expressed as formula (2), where n represents the total number of joints in the robot. From this consolidated transformation matrix ${}^{0}_{n}T$, we can extract both the position and orientation details of the end-effector, allowing for precise control and manipulation of the robotic arm.

$$ {}^{i-1}_{i}A = \begin{bmatrix} cos\theta_i & -sin\theta_i cos\alpha_i & sin\theta_i sin\alpha_i & a_i cos\theta_i \\ sin\theta_i & cos\theta_i cos\alpha_i & -cos\theta_i sin\alpha_i & a_i sin\theta_i \\ 0 & sin\alpha_i & cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1} $$

$$ {}^{0}_{n}T = {}^{0}_{1}A(\theta_1)\,{}^{1}_{2}A(\theta_2) \dots {}^{n-1}_{n}A(\theta_n) \tag{2} $$

For the UR5e, only the joint angles are adjustable. Assume all of the θ angles remain in their initial state, which is equal to 0. Other parameters follow the classical D-H table. Then the forward kinematics transformation matrix of the UR5e is computed using the following MATLAB code:

```
% DH parameters
a = [0, -0.42500, -0.3922, 0, 0, 0]';
d = [0.1625, 0, 0, 0.1333, 0.0997, 0.0996]';
alpha = [pi/2, 0, 0, pi/2, -pi/2, 0]';
% Adjustable parameters
theta = [0,0,0,0,0,0];


T=zeros(4,4);
for i=1:6
   T(:,:,i) = [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
% T06 is the forward kinematic matrix
T06 = T(:,:,1)*T(:,:,2)*T(:,:,3)*T(:,:,4)*T(:,:,5)*T(:,:,6);
disp(T06);
```

As an example, when all joint angles (θ) are set to 0 degrees, the forward kinematics transformation matrix ${}^{0}_{6}T$ is equal to (3). The matrix T varies based on changes in joint angle values.

$$ {}^{0}_{6}T = \begin{bmatrix} 1 & 0 & 0 & -0.8172 \\ 0 & 0 & -1 & -0.2329 \\ 0 & 1 & 0 & 0.0628 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3} $$

# Chapter 4 Inverse Kinematics Analysis

In this chapter, we explore the analytical inverse kinematics problem for robotic arms, focusing on determining the joint configurations that achieve a specified position and orientation of the robot's end effector. The inverse kinematics (IK) problem is pivotal in robotics, particularly when precise control of the end effector is required for tasks such as assembly, painting, or surgical operations.

## 1. Traditional Geometric Approach

Traditionally, the inverse kinematics problem has been tackled using geometric methods. This approach is highly intuitive, relying on the physical dimensions and configuration of the robot. It involves applying fundamental geometric and trigonometric principles to deduce the necessary joint angles that result in the desired end effector position. While effective for robots with a simpler structure or fewer degrees of freedom (DOF), this method may not be suitable for more complex robotic systems.

Here, to calculate the inverse kinematics for the UR5e robot using the traditional geometric approach, let's focus on a specific target position for the end effector. Given that we have the dimensions of the robot and the target, we can proceed to solve for the joint angles θ1 to θ6 using geometric relations.

Let's define a simplified scenario:

1. **Target Position:** Assume we have a target position in space for the end effector, such as $[x_t, y_t, z_t]$.

2. Robot Configuration: We will consider the significant dimensions of the UR5e from the D-H parameters you provided, focusing on lengths and offsets between joints.

**Steps to Solve for Inverse Kinematics:**

1. **Calculate $\boldsymbol{\theta_1}$:** The first joint rotates around the z-axis. $\theta_1$ can be determined by projecting the target position onto the xy-plane and calculating the angle from the x-axis to this projection.

2. **Wrist Center Calculation:** We need to calculate the position of the wrist center, which depends on the last joint (wrist joint). For simplicity, let's consider the length from the last joint to the end effector as a known constant $d_6$.
   The wrist center (WC) position is then:
   $$WC = [x_t, y_t, z_t] - d_6 \times \text{end effector orientation}$$

3. **Calculate $\boldsymbol{\theta_2}$ and $\boldsymbol{\theta_3}$:** Using the position of the wrist center, we calculate $\theta_2$ and $\theta_3$ by solving the triangle formed by the first three joints. Employ the Law of Cosines and Law of Sines in the plane formed by these joints, considering the known lengths of the arm segments a2 and a3.

4. **Orientation calculation for $\boldsymbol{\theta_4}$, $\boldsymbol{\theta_5}$, and $\boldsymbol{\theta_6}$:**These angles are determined based on the orientation of the end effector, which involves decomposing the desired end effector orientation relative to the orientation achieved by the first three joints.

Let's calculate the inverse kinematics for a simplified target position, assuming a hypothetical target and orientations.

Here's a hypothetical MATLAB snippet that would implement part of this logic:

```
% Assume target position (in meters)
targetPosition = [0.5, -0.3, 0.7];
% Assume end effector length (last segment)
d6 = 0.1;  % 10 cm from the last joint to the end of the effector
% Assume no rotation for simplification (end effector orientation vector)
```

```
end_effector_orientation = [0, 0, 1];  % Pointing straight down
% Wrist center position
wrist_center = targetPosition - d6 * end_effector_orientation;
% Calculate theta1
theta1 = atan2(wrist_center(2), wrist_center(1));
% Display results
disp(['Theta 1: ', num2str(theta1)]);
……
```

This snippet calculates θ1 based on the projection of the wrist center on the xy-plane. The remaining angles would need additional geometric calculations involving the specific arm segment lengths and the desired orientation, which typically would need to take into account the joint limits and possible collisions. Therefore, the actual calculation process is more complicated.

## 2. Challenges with Higher Degrees of Freedom

For robots with six degrees of freedom (6 DOF), which are common in both industrial and research applications, the geometric approach can become cumbersome and inadequate due to the increased complexity and potential for multiple solutions or kinematic redundancies. Hence, numerical methods are preferred due to their flexibility and capability to handle high-dimensional configuration spaces.

## 3. Numerical Approach to Inverse Kinematics

The numerical approach utilizes iterative algorithms that can efficiently converge on a solution even for complex robots. The process can be outlined as follows:

Step 1. Define the Target Pose:

Position: `targetPosition = [0.5, -0.3, 0.6]` represents the desired location of the end effector in three-dimensional space.

Orientation: `targetOrientation = [1, 0, 0, 0]`, a quaternion that denotes no rotation from the base orientation.

Step 2. Setup the Inverse Kinematics Solver:

An inverse kinematics solver, which code is ‘ik = robotics.InverseKinematics ('RigidBodyTree', robot)’, is initialized. This solver is designed to interact with the robot's `RigidBodyTree` model, which includes a comprehensive definition of its structural components, joints, and linkages.

Step 3. Solve the Inverse Kinematics:

The command `[configSoln, solnInfo] = ik(robot.BodyNames{end}, tform, weights, initialGuess)` calculates the joint configurations. Here, the variables are: `tform`: The transformation matrix that incorporates both `targetPosition` and `targetOrientation`. `weights`: A set of parameters that prioritize the importance of achieving position over orientation or vice versa. `initialGuess`: An initial set of joint angles that serves as a starting point for the iterative process.

Step 4. Common Algorithms Used in Numerical IK Solutions:

Several algorithms are typically employed in numerical IK solvers: Gradient Descent: Searches for a minimum error in joint space by following the steepest descent based on the gradient of a cost function. Newton-Raphson Method: An iterative approach that uses derivatives to find zeros of a function, applied here to minimize the difference between current and desired end effector states. Jacobian Inverse Method: Uses the inverse of the Jacobian matrix of partial derivatives to find increments in joint angles that yield a desired increment in end effector position and orientation.

## 4. Implementing the Solution in Robotics System Toolbox

In practice, these techniques are implemented in environments like MATLAB's Robotics System Toolbox, which provides tools and functions for modeling and

simulating robotic systems. This toolbox allows for a robust application of numerical methods, enabling the adjustment of parameters like weights and initial guesses to optimize the performance of the IK solver for different scenarios.

In Figure 8, we employ the UR5e model from the Robotics System Toolbox to solve for the target position (0, 0.6, 0.1) with the target orientation specified by the quaternion (0.707 0 0.707 0).

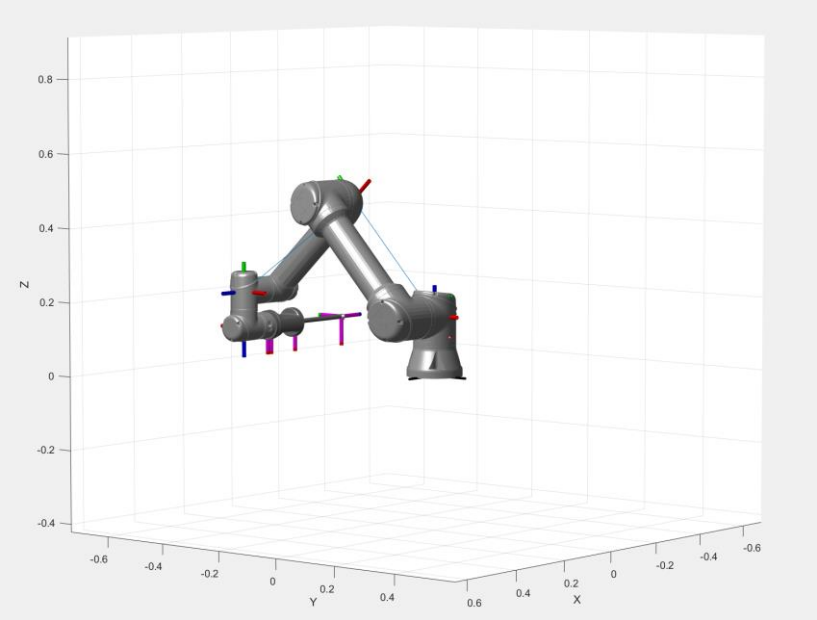


***Fig. 9 IK solution example***

By using these advanced numerical techniques, engineers and researchers can efficiently solve the inverse kinematics for robots with multiple degrees of freedom, enhancing the robot's ability to perform complex tasks with high precision.

# Chapter 5 Upright 3D Plane Definition

In this chapter, we detail the configuration of a three-dimensional upright plane that simulates the rear view of a human torso within the workspace. This plane is meticulously crafted based on typical human body proportions to provide a realistic model for robotic interactions. The dimensions are set as follows: a torso length of 0.525 meters, a shoulder width of 0.45 meters, and a waist width of 0.35 meters. Positioned 0.6 meters away from the robot, this plane serves as the stage for the robot to recognize and interact with strategically designated target points.

### 1. Target Points Defined

The target points on this plane are carefully selected to represent key anatomical landmarks, enhancing the robot's ability to perform tasks that mimic human interaction, such as in physical therapy or ergonomic assessments. Each point is described with its coordinates in metric units (meters) as follows:

- **Above the Left Shoulder:**

Coordinates: (-0.225, 0.6, 0.625)

Significance: Represents the area just above the left shoulder, crucial for applications like targeted muscle therapy or placement of wearable medical devices.

- **Above the Right Shoulder:**

Coordinates: (0.225, 0.6, 0.625)

Significance: Symmetric to the left shoulder point, important for balanced therapeutic interventions and assessments across both shoulders.

- **Left Waist:**

Coordinates: (-0.175, 0.6, 0.1)

Significance: Located at the left waist level, vital for procedures that involve the lower torso such as kidney stone therapy or lower rib examination.

- **Right Waist:**

Coordinates: (0.175, 0.6, 0.1)

Significance: Mirrors the left waist, ensuring symmetrical access to both sides of the lower torso for comprehensive therapeutic coverage.

- **Upper Spine:**

Coordinates: (0, 0.6, 0.49375)

Significance: Positioned halfway up the spine, key for central spinal treatments and can serve as a reference point for alignment in posture correction.

- **Middle Spine:**

Coordinates: (0, 0.6, 0.3625)

Significance: Indicates the mid-spinal region, crucial for targeting mid-back issues and spinal therapies.

- **Lower Spine:**

Coordinates: (0, 0.6, 0.23125)

Significance: Represents the lower spinal area, essential for addressing lower back pain and disorders such as lumbar arthritis.

- **Bottom of the Neck:**

Coordinates: (0, 0.6, 0.625)

Significance: Directly at the base of the neck, a pivotal area for neck therapies, cervical spine assessments, and the application of neck braces or other medical devices.

**2. Justification for Point Selection**

The selection of these points is driven by their anatomical and therapeutic relevance. By targeting these specific locations, the robot can simulate and practice a range of motions and procedures that are commonly required in medical and ergonomic applications. This spatial arrangement not only allows the robot to navigate and adapt to human proportions accurately but also provides a comprehensive framework for developing algorithms that handle complex movements, ensuring precise and safe interactions with human-like figures.

## 3. Application in Robotics

Incorporating these specific target points into robotic programming enables the development of sophisticated robotic assistants that can perform delicate tasks, such as administering injections, placing electrodes, or conducting specific manual therapy techniques. This approach significantly enhances the robot's utility in clinical settings, offering a safe and controlled environment to refine therapeutic techniques before direct application on patients, thereby improving both the safety and efficacy of robotic-assisted procedures.

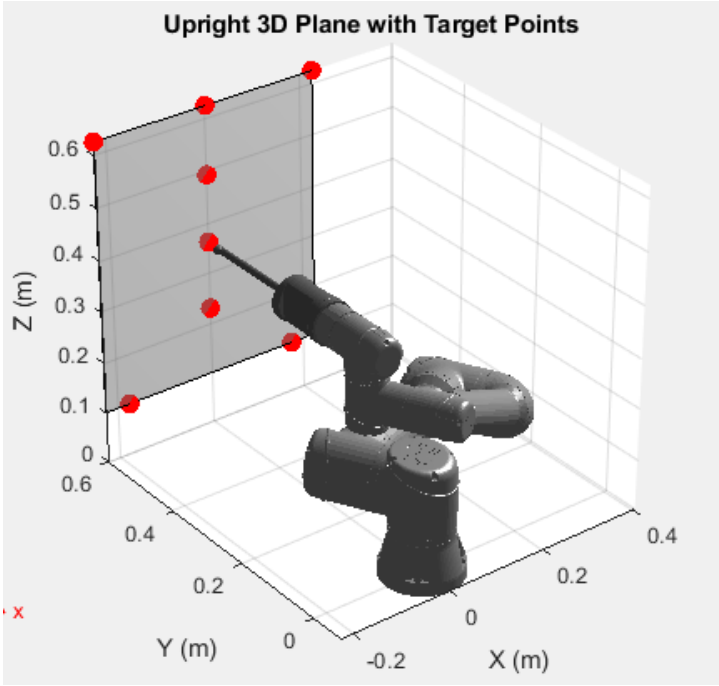


*Fig. 10 Upright 3D Plane with Target points*

# Chapter 6 Move the Manipulator to Insert a Needle

In this section, our task is to program the manipulator to precisely move its distal end to insert a needle at each designated point. We start from the zero position of the manipulator, establishing a baseline for all movements. To ensure accuracy and safety, the program will command the manipulator to pause at the start and end of each needle insertion, which is critical for the piercing process. Additionally, we must define the speed at which the piercing should occur, ensuring that the insertion is consistent and controlled. To calculate the necessary movements and achieve the required speed, we will employ the **inverse Jacobian** method, which allows us to translate desired end-effector velocities into corresponding joint actuation. This approach ensures that the manipulator operates smoothly and efficiently, achieving the intended piercing with precision.

**Trajectory Planning**

Firstly, we use the quintic polynomial to generate the motion trajectory according to Chapter 3. This trajectory planning method allows for the precise control of position, velocity, and acceleration at the start and end of a motion path, and is commonplace in robotics, especially for the motion planning of robotic arms where precise positioning and smooth motion are desired. It is particularly suited for applications where it is important to avoid jerks and vibrations during movement, such as in manufacturing, automated assembly, or in precision lab instruments.

The trajectory is represented as a function of time using a fifth-order polynomial as equation (4). This allows for the control of position, velocity, and acceleration throughout the planning process. The six coefficients $a_0$ to $a_5$ of the polynomial are determined by six constraints, typically two positions, two velocities, and two accelerations at the initial and final times. A quintic polynomial is preferred over a cubic

polynomial because it avoids discontinuities in acceleration at the beginning and end of the trajectory, which can lead to reduced vibrations. A cubic polynomial does not allow for the specification of constraints on acceleration at the start and end points.

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

**Practical Implementation:**

1. Defining initial and final conditions, including positions, velocities, and accelerations.
2. Solving for the coefficients of the polynomial using these boundary conditions shown as below.

$$a_0 = \theta_0$$

$$a_1 = \dot{\theta}_0$$

$$a_2 = \frac{\ddot{\theta}_0}{2}$$

$$a_3 = \frac{20\theta_f - 20\theta_0 - (8\dot{\theta}_f + 12\dot{\theta}_0)t_f - (3\ddot{\theta}_0 - \ddot{\theta}_f)t_f^2}{2t_f^3}$$

$$a_4 = \frac{30\theta_0 - 30\theta_f + (14\dot{\theta}_f + 16\dot{\theta}_0)t_f + (3\ddot{\theta}_0 - 2\ddot{\theta}_f)t_f^2}{2t_f^4}$$

$$a_5 = \frac{12\theta_f - 12\theta_0 - (6\dot{\theta}_f + 6\dot{\theta}_0)t_f - (\ddot{\theta}_0 - \ddot{\theta}_f)t_f^2}{2t_f^5}$$

3. Computing the desired position, velocity, and acceleration at any point in time by substituting the time t into the polynomial.
4. Controlling the motion of the robotic arm or other robots using these trajectory points.

Here's a hypothetical MATLAB snippet that would implement part of this logic:

```
ik = robotics.InverseKinematics('RigidBodyTree', robot);
weights = [0.25 0.25 0.25 1 1 1]; % Weight of joint position and attitude
```

```
initialGuess = robot.homeConfiguration;

%% Execution
home_theta_values = homePosition;

for idx=(1:8)
    Final_pos=targetPositions(idx,:);
    tform = trvec2tform(targetPositions(idx, :)) * quat2tform(targetOrientation);
    Final_Theta = ik(robot.BodyNames{end}, tform, weights, initialGuess);
    final_theta_values = zeros(1, 6);
    for i = 1:6
        final_theta_values(i) = Final_Theta(i).JointPosition;
    end
    step = 100;

    % Time parameters
    T = 1; % Assume the time from start to end is 1 second
    t = linspace(0, T, step); % Generate time vector

    % Assume initial velocity and acceleration are both zero, and final velocity and acceleration are
also zero
    theta_dot_0 = zeros(1, 6); % Initial velocity
    theta_ddot_0 = zeros(1, 6); % Initial acceleration
    theta_dot_f = zeros(1, 6); % Final velocity
    theta_ddot_f = zeros(1, 6); % Final acceleration
    t_f = 1; % Total time for interpolation

    % Compute coefficients of the quintic polynomial
    a0 = home_theta_values;
    a1 = theta_dot_0;
    a2 = theta_ddot_0 / 2;
    a3 = (20*(final_theta_values - home_theta_values) - (8*theta_dot_f + 12*theta_dot_0)*t_f ...
        - (3*theta_ddot_0 - theta_ddot_f)*t_f^2) / (2*t_f^3);
    a4 = (30*(home_theta_values - final_theta_values) + (14*theta_dot_f + 16*theta_dot_0)*t_f ...
        + (3*theta_ddot_0 - 2*theta_ddot_f)*t_f^2) / (2*t_f^4);
    a5 = (12*(final_theta_values - home_theta_values) - 6*(theta_dot_f + theta_dot_0)*t_f ...
        - (theta_ddot_0 - theta_ddot_f)*t_f^2) / (2*t_f^5);
```

```
% Use quintic polynomial to compute joint angles at each time point
Theta = zeros(step, 6); % Initialize joint angle matrix
for i = 1:step
    t_i = t(i);
    Theta(i,:) = a0 + a1*t_i + a2*t_i^2 + a3*t_i^3 + a4*t_i^4 + a5*t_i^5;
end
```

**Velocity Control**

To compute the Jacobian matrix for a robotic manipulator involves deriving the relationship between the joint velocities and the end effector's linear and angular velocities. And here's an outline of steps for us to compute:

1. **Define the Kinematics:** Start by defining the forward kinematics of the robot using the Denavit-Hartenberg convention or another method. This will establish the relationship between the joint angles and the position and orientation of the end effector.
2. **Derive Partial Derivatives:** Calculate the partial derivatives of the end effector's position with respect to each joint angle. These derivatives express how a small change in each joint angle can affect the position of the end effector in space.
3. **Assemble the Jacobian Matrix:** The top three rows of the Jacobian correspond to the linear velocities and are obtained by taking the derivatives of the end effector's position concerning the joint angles. The bottom three rows correspond to the angular velocities and are derived from the rotational part of the kinematic equations.
4. **Linear Velocity Components:** For each joint, calculate how changes in the joint variable affect the end effector's linear velocity along the x, y, and z axes.
5. **Angular Velocity Components:** Similarly, compute the contributions of each joint's motion to the end effector's angular velocity about each of the axes.
6. **Verify Dimensions:** Ensure that the Jacobian matrix is correctly sized. For a robotic arm with n joints, the Jacobian matrix will be a 6 x n matrix where n is the **number of joints**.
7. **Consider the Physical Limits:** Take into account any physical constraints of the

robotic system that might affect the movement, such as joint limits, to avoid singularities in the Jacobian.

8. **Implement and Test:** Once the Jacobian is computed, it should be implemented in the control software and tested to ensure that it correctly predicts the end effector's behavior in response to joint movements.

Here's a hypothetical MATLAB snippet that would implement part of this logic:

```
function [q] = Insert(ur5e, Start_point, End_point, init_q, dt)
    % Calculate displacement vector
    displacement = (End_point - Start_point) / dt;
    % Compute the Jacobian matrix
    J = geometricJacobian(ur5e, init_q, ur5e.BodyNames{end});
    J_inv = pinv(J(4:6, :));  % Invert the Jacobian for the last three rows
    % Displacement should be a column vector
    d = displacement';
    % Inverse kinematics calculation for dq
    dq = J_inv * d;  % Note that the displacement should be a column vector
    % Update joint angles
    q = init_q + dq'*dt;
end
```

Due to the complexity involved in computing the Jacobian matrix, we have limited its use to controlling the forward movement of the end effector (needle head) using the inverse Jacobian method. For all other movements, we employ inverse kinematics combined with quaternion calculus. By repeatedly sampling positions between two puncture points, we calculate multiple trajectories to form a smooth motion path. Additionally, it is imperative to maintain a uniform velocity while the needle head is in motion and to ensure that the end effector consistently moves along the x-axis, maintaining orientation in the other directions. Therefore, **no rotational vector is involved**, allowing us to simplify the calculations to just translational movements. This means we only need to utilize the **upper part** of the Jacobian matrix, which deals with linear velocities, to determine the necessary joint velocities for the desired movement

of the end effector. This approach streamlines the computational process and optimizes the efficiency of the robotic system's operations.

The specific effect is demonstrated in the video. In the report, only the start and end points of the needle insertion for the first target point are presented as an example.

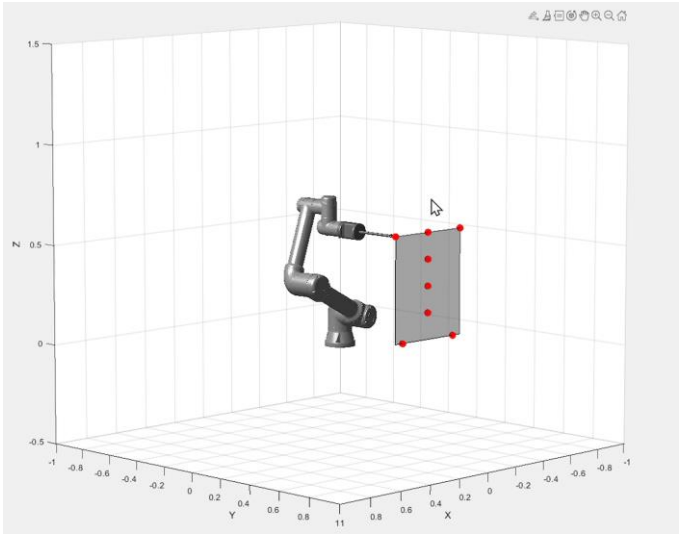


***Fig. 11 Start Point of piercing***

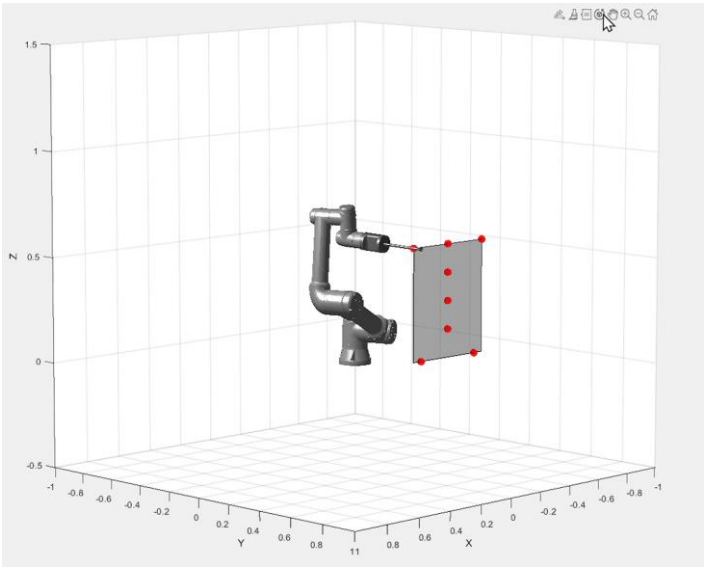


***Fig. 12 End Point of piercing***

# Discussion and Conclusion

During our inverse kinematics calculations, we observed that due to the multiple degrees of freedom in robotic arms, there can be multiple solutions for a single operational state within the workspace. To address this, we can impose additional constraints locally to narrow down the solutions and achieve the desired operational state more reliably.

Expanding on this, implementing local constraints can help in guiding the robotic arm to avoid certain positions or configurations that may lead to inefficiencies or potential hazards. For instance, constraints can be set to minimize joint movements, avoid singularities, or keep the arm within safe operational zones. This method enhances precision in task execution and increases the predictability of the robot's behavior, making it more suitable for complex tasks where precision and safety are paramount.

In the video, it's observable that some points are unable to move along a straight line, which we suspect might be due to the following reasons:

1. Singularity: The Jacobian matrix can lose full rank near or at singular configurations of the robotic arm, making it impossible to accurately compute its inverse. In such cases, even small desired velocities at the end-effector can lead to large or undefined joint velocities.
2. Constraints: If the robotic arm has physical constraints, such as joint angle limits or collision avoidance requirements, using just the inverse Jacobian may result in these constraints being violated, as the inverse solution may not consider these factors.
3. Numerical Issues: Numerical precision issues or computational errors can also lead to deviations between actual and expected movements.
4. End-Effector Speed: If the commanded end-effector speed is too high, the actual robot joints might not be able to keep up, especially in rapidly changing trajectories.

5. Choice of Inverse Jacobian Method: Different methods for computing the inverse Jacobian, such as using the pseudo-inverse, have distinct characteristics and limitations. For example, the pseudo-inverse can handle non-square matrices but may still encounter issues near singularities.
6. Kinematics V.S. Dynamics: Considering only the kinematics might not be sufficient to describe real scenarios accurately. The dynamics of the robotic arm, including factors like acceleration, mass, and friction, also affect motion.

To avoid such issues, we suggest that if a robotic arm is required to perform tasks similar to needle insertion, an additional solution could be to equip the end-effector with a needle mechanism capable of controlled forward and backward movements. This adjustment would allow for precise control over the needle's position, circumventing the issues related to the robotic arm's inability to move along a predetermined path as expected. This setup ensures that even if the arm itself cannot precisely follow the intended linear trajectory, the needle can still reach its target accurately and perform its function effectively.

## Reference

[1] UR5e Technical Specification

ur5e-rgb-fact-sheet-landscape-a4.pdf (universal-robots.com)

[2] DH Parameters for calculations of kinematics and dynamics

Universal Robots - DH Parameters for calculations of kinematics and dynamics (universal-robots.com)