

# Mathematical Modeling of a Three Axis Robotic Arm

Umut Özen, Aleyna Özdoğan

Department of Mathematics,

Dokuz Eylül University

*Supervisor: Prof. Dr. Başak Karpuz*

2024–2025 – Spring Semester



# Abstract

## Abstract

This presentation models and simulates the spatial motion of a three-degree-of-freedom robotic arm using Python. Joint coordinates are calculated based on link lengths ( $L_1, L_2$ ) and joint angles ( $\theta, \phi_1, \phi_2$ ) through trigonometric expressions. The geometric and trigonometric conditions for the arm's endpoint to reach a target surface (e.g., a whiteboard) are analyzed, including valid joint angle ranges. Vector analysis, trigonometric transformations, and the law of cosines are employed, and the results are visualized and tested in Python. The project offers a theoretical and practical approach to solving inverse kinematics problems in robotics.

# Outline of the Talk

## ① Introduction and Applications

- Real-world applications of 3 variable robotic arms  
(e.g., assembly lines, surgery, 3D printing, education)
- Motivation: Need for precise control in spatial environments

## ② Problem Definition

- What is spatial positioning?
- Why modeling robotic arm kinematics is challenging and important

## ③ Mathematical Representation of the Robotic Arm

- Definition of link lengths ( $L_1, L_2$ ) and joint angles ( $\theta, \phi_1, \phi_2$ )
- Description of coordinate system and base point ( $x_0, y_0, z_0$ )

# Outline of the Talk

## ④ Forward and Inverse Kinematics

- Computing joint positions ( $x_1, y_1, z_1$ ) and end-effector ( $x_2, y_2, z_2$ )
- Deriving inverse kinematic expressions for  $\theta, \phi_1, \phi_2$

## ⑤ Geometric and Trigonometric Constraints

- Applying the law of cosines and triangle relationships
- Constraints for reaching a target (e.g., whiteboard surface)

## ⑥ Python-Based Simulation and Visualization

- Implementation of kinematic model in Python
- 3D graphical representation and animation of the robotic arm

# Applications of Three Axis Robotic Arm

- **Assembly Lines:** 3D positioning and precise assembly of components.
- **Welding and Painting Applications:** Performing tasks at complex angles.
- **Surgical Robots:** Medical operations requiring precise, multi-axis movements.
- **3D Printing and CNC Machining:** Executing complex toolpaths in manufacturing.
- **Research and Education:** Used in advanced robotics control and kinematic studies.

# Applications of Two-Arm Robotic Systems

## Excavation and Industrial Robotics



Excavator system



Industrial robot arm

# Problem Statement

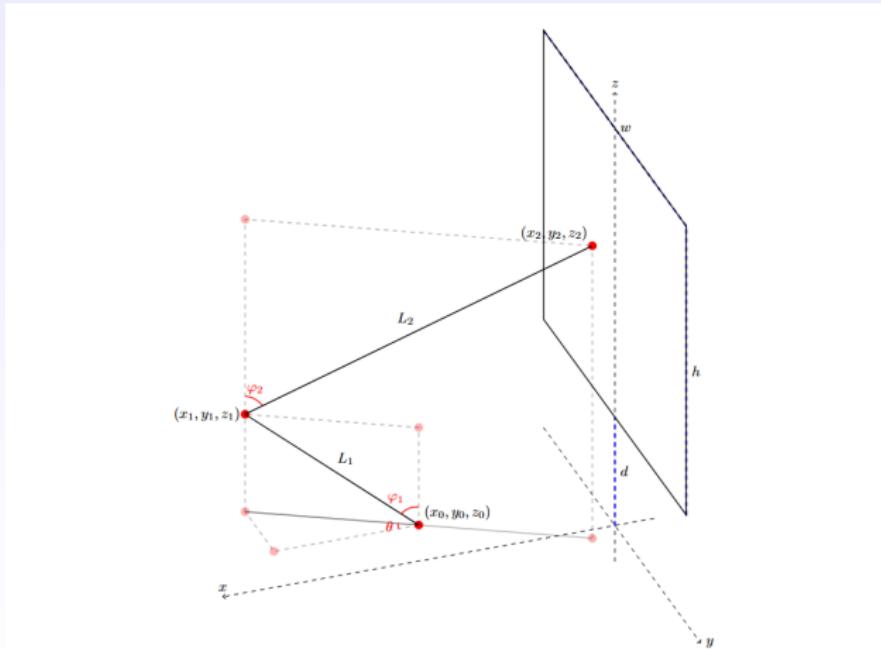
## Kinematic Modeling of a 3 Variable Robotic Arm

The precise control and spatial positioning of robotic arms with multiple variables present significant challenges in robotics.

Modeling the kinematics of a three-axis robotic arm mathematically is essential for understanding its movement, enabling accurate targeting and manipulation within a 3D workspace.

This project aims to develop a comprehensive mathematical model and simulation framework to solve the forward and inverse kinematics problems for such robotic arms, ensuring efficient and reliable operation in real-world applications.

# Representation of Robot Arm Angles and Coordinates



# Mathematical Representation of the Robotic Arm

## Link Lengths and Joint Angles

Let  $(x_0, y_0, z_0)$  denote the base point of the robotic arm. Let  $L_1$  and  $L_2$  be the lengths of the first and second links, respectively.

- $\phi_1$  is the angle between link  $L_1$  and the  $z$ -axis.
- $\phi_2$  is the angle between link  $L_2$  and the  $z$ -axis.
- $\theta$  is the angle between the  $x$ -axis and the  $xy$ -projection of  $L_1$ .

# Mathematical Representation of the Robotic Arm

## Forward Kinematic Equations

Using trigonometric relationships, the coordinates of the joints are:

### Intermediate point (joint 1):

$$x_1 = x_0 + L_1 \sin(\phi_1) \cos(\theta)$$

$$y_1 = y_0 - L_1 \sin(\phi_1) \sin(\theta)$$

$$z_1 = z_0 + L_1 \cos(\phi_1)$$

### End-effector (joint 2):

$$x_2 = x_1 - L_2 \sin(\phi_2) \cos(\theta)$$

$$y_2 = y_1 + L_2 \sin(\phi_2) \sin(\theta)$$

$$z_2 = z_1 + L_2 \cos(\phi_2)$$

# Mathematical Representation of the Robotic Arm

Angle Formulas for  $\theta, \phi_1, \phi_2$

**Angle  $\theta$  (rotation in  $xy$ -plane):**

$$\theta = -\arctan\left(\frac{y_2 - y_0}{x_2 - x_0}\right)$$

**Angle  $\phi_1$  (inclination of first link):**

$$\phi_1 = \arctan\left(\frac{x_1 - x_0}{z_1 - z_0 \cdot \sec(\theta)}\right)$$

**Angle  $\phi_2$  (inclination of second link):**

$$\phi_2 = \arctan\left(\frac{y_2 - y_1}{z_2 - z_1 \cdot \csc(\theta)}\right)$$

These expressions allow us to compute the required joint angles based on spatial coordinates of the arm.

# Animation Program Code

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from matplotlib.animation import FuncAnimation
4
5 # Calculates the planar angle (theta) between two points in the
6 # XY plane
7 def tf(x, y, u, v):
8     return np.pi - np.arctan2(v - y, u - x)
9 # Computes the 3D Euclidean distance between two points (used as
10 # 13)
11 def l3f(x, y, z, u, v, w):
12     return np.linalg.norm([u - x, v - y, w - z])
13 # Calculates the angle (in radians) using the Law of Cosines
14 def af(l, k, h):
15     return np.arccos((k**2 + h**2 - l**2) / (2 * k * h))
16 # Alternative version of af() with different argument order
17 def bf(l, k, h):
18     return af(k, l, h)
```

# Animation Program Code

```
1 # Calculates the vertical angle based on the Z-axis height  
2     difference  
3 def df(z, w, h):  
4     return np.arccos(np.abs(w - z) / h)  
5 # Calculates the first joint angle phi1  
6 def f1f(z, w, l, k, h):  
7     return bf(l, k, h) - df(z, w, h)  
8 # Calculates the second joint angle phi2  
9 def f2f(z, w, l, k, h):  
10    return af(l, k, h) + df(z, w, h)  
11  
12 # parameters  
13 w, h, d = 10, 8, 5  
14 l1, l2 = 5, 10  
15 x0, y0, z0 = 5, 0, 1
```

# Animation Program Code

```
1 # The logo is parameterized over the interval t \in [0, 14],  
2 def batmanlogo_scalar(t):  
3     pi = np.pi  
4     # Transforms and scales the shape to fit the drawing area  
5     def transform(x, y):  
6         return [x / 2, y / 2 + 8]  
7     # Piecewise construction of the Batman logo using  
8     # trigonometric and polynomial segments  
9     if 0 <= t < 1:  
10        return transform(0.6 * t, 4)  
11    elif 1 <= t < 2:  
12        return transform(0.6 + 0.6 * (t - 1), 4 + 0.94 * (t - 1))  
13    elif 2 <= t < 3:  
14        return transform(1.2, 4.9434 - 1.4434 * (t - 2))  
15    elif 3 <= t < 4:  
16        s = (0.576 + pi) * (t - 3) - pi  
17        return transform(1.52 * np.cos(s) + 2.72, 1.52 * np.sin(s)  
18        ) + 3.5)
```

# Animation Program Code

```

1    elif 4 <= t < 5:
2        return transform(8 * np.cos(1.83 * (5 - t) - 0.785), 5 *
3        np.sin(1.83 * (5 - t) - 0.785))
4    elif 5 <= t < 6:
5        s = 4.4 + 2.26 * (6 - t)
6        return transform(-12.43 + 5.212 * s - 0.375 * s**2,
7        -21.52 + 7.03 * s - 0.65 * s**2)
8    elif 6 <= t < 7:
9        s = 3.25 * (7 - t)
10       return transform(s, -2 - 0.75 * (s - 2)**2)
11   elif 7 <= t < 8:
12       s = 3.25 * (t - 7)
13       return transform(-s, -2 - 0.75 * (s - 2)**2)
14   elif 8 <= t < 9:
15       s = 4.4 + 2.26 * (t - 8)
16       return transform(12.43 - 5.212 * s + 0.375 * s**2, -21.52
17       + 7.03 * s - 0.65 * s**2)
18   elif 9 <= t < 10:
19       s = 1.83 * (t - 9) - 0.785
20       return transform(-8 * np.cos(s), 5 * np.sin(s))

```

# Animation Program Code

```
1 elif 10 <= t < 11:
2     s = (0.576 + pi) * (11 - t) - pi
3     return transform(-1.52 * np.cos(s) - 2.72, 1.52 * np.sin(
4         s) + 3.5)
5 elif 11 <= t < 12:
6     return transform(-1.2, 4.9434 - 1.4434 * (12 - t))
7 elif 12 <= t < 13:
8     return transform(-0.6 - 0.6 * (13 - t), 4 + 0.94 * (13 -
9         t))
10 elif 13 <= t < 14:
11     return transform(-0.6 * (14 - t), 4)
else:
    return [0, 0]
```

# Animation Program Code

```
1 # Returns x and y coordinates for the given time(s) t by
2 # evaluating the Batman logo path.
3 def batmanlogo(t):
4     t = np.asarray(t)
5     if t.ndim == 0:
6         return batmanlogo_scalar(t)
7     else:
8         result = np.array([batmanlogo_scalar(ti) for ti in t])
9     return result[:, 0], result[:, 1]
10 # Converts the 2D Batman logo path into 3D by assigning x = 0.
11 def batmanlogo3d(t):
12     x, y = batmanlogo(t)
13     return [0 * np.array(x), x, y]
14 # Helper functions to extract the X, Y, and Z coordinates from
15 # the 3D path.
16 def x_t(t): return batmanlogo3d(t)[0]
17 def y_t(t): return batmanlogo3d(t)[1]
18 def z_t(t): return batmanlogo3d(t)[2]
```

# Python Simulation Codes and Graphical Representation

## Python Animations

# Python Simulation Codes and Graphical Representation

## Python Animations

# Python Simulation Codes and Graphical Representation

## Python Animations

# Python Simulation Codes and Graphical Representation

## Python Animations

# Thank You



Thank you for listening to our presentation!  
If you're interested,  
you can explore more by scanning the QR code.