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 (θ, ϕ_1, ϕ_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 (θ, ϕ_1, ϕ_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 θ, ϕ_1, ϕ_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

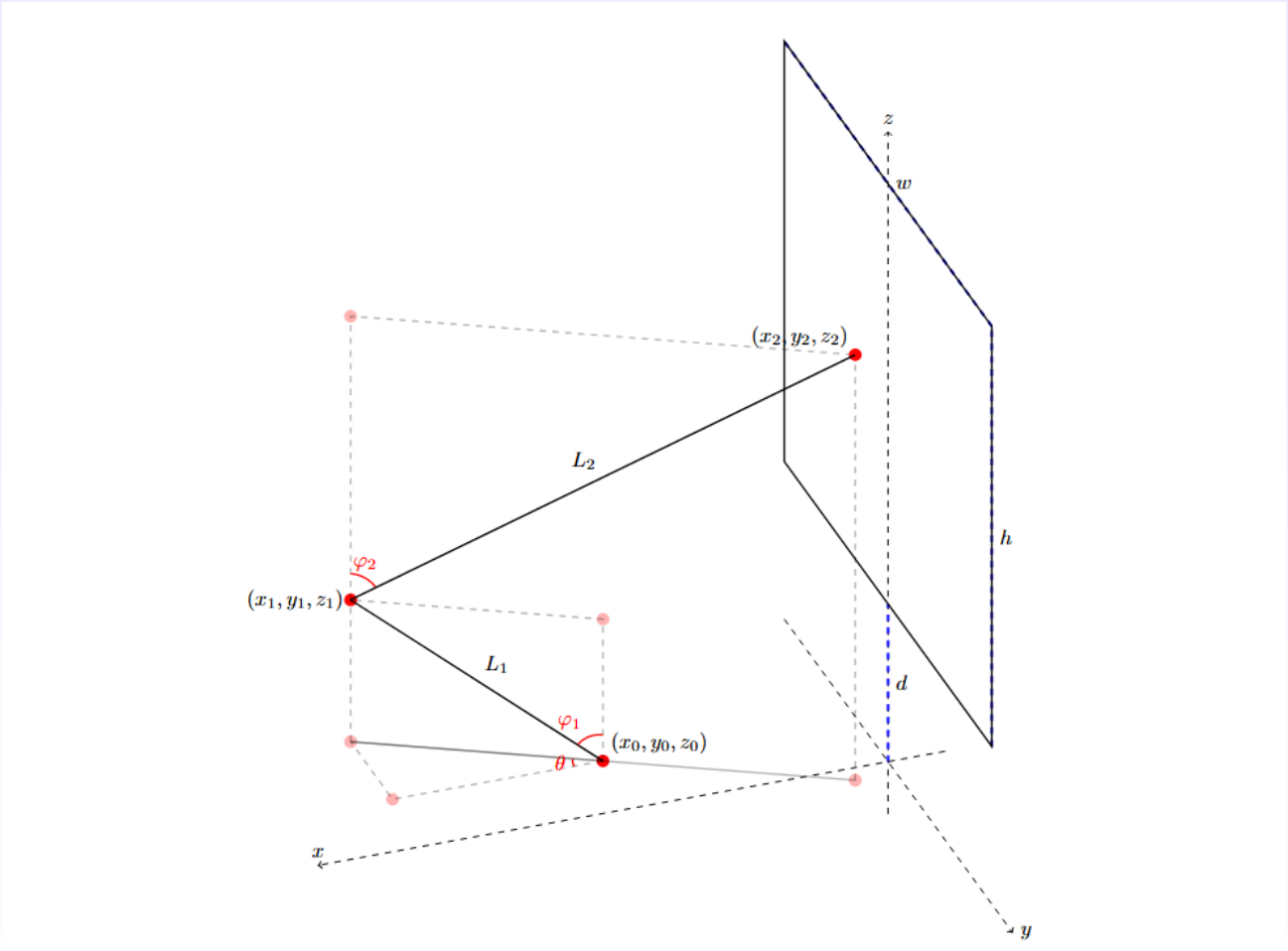
Kinematic Modeling of a 3 Variable Robotic Arm

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.

- ϕ_1 is the angle between link L_1 and the z-axis.
- ϕ_2 is the angle between link L_2 and the z-axis.
- θ 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 θ, ϕ_1, ϕ_2

Angle θ (rotation in xy-plane):

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

Angle ϕ_1 (inclination of first link):

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

Angle ϕ_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

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

# Calculates the planar angle (theta) between two points in the XY plane
def tf(x, y, u, v):
    return np.pi - np.arctan2(v - y, u - x)
# Computes the 3D Euclidean distance between two points (used as l3)
def l3f(x, y, z, u, v, w):
    return np.linalg.norm([u - x, v - y, w - z])
# Calculates the angle (in radians) using the Law of Cosines
def af(l, k, h):
    return np.arccos((k**2 + h**2 - l**2) / (2 * k * h))
# Alternative version of af() with different argument order
def bf(l, k, h):
    return af(k, l, h)
```

Animation Program Code

```
# Calculates the vertical angle based on the Z-axis height difference
def df(z, w, h):
    return np.arccos(np.abs(w - z) / h)
# Calculates the first joint angle phi1
def f1f(z, w, l, k, h):
    return bf(l, k, h) - df(z, w, h)
# Calculates the second joint angle phi2
def f2f(z, w, l, k, h):
    return af(l, k, h) + df(z, w, h)

# parameters
w, h, d = 10, 8, 5
l1, l2 = 5, 10
x0, y0, z0 = 5, 0, 1
```

Animation Program Code

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

Animation Program Code

```
    elif 4 <= t < 5:
        return transform(8 * np.cos(1.83 * (5 - t) - 0.785), 5 *
    np.sin(1.83 * (5 - t) - 0.785))
    elif 5 <= t < 6:
        s = 4.4 + 2.26 * (6 - t)
        return transform(-12.43 + 5.212 * s - 0.375 * s**2,
    -21.52 + 7.03 * s - 0.65 * s**2)
    elif 6 <= t < 7:
        s = 3.25 * (7 - t)
        return transform(s, -2 - 0.75 * (s - 2)**2)
    elif 7 <= t < 8:
        s = 3.25 * (t - 7)
        return transform(-s, -2 - 0.75 * (s - 2)**2)
    elif 8 <= t < 9:
        s = 4.4 + 2.26 * (t - 8)
        return transform(12.43 - 5.212 * s + 0.375 * s**2, -21.52
     + 7.03 * s - 0.65 * s**2)
    elif 9 <= t < 10:
        s = 1.83 * (t - 9) - 0.785
        return transform(-8 * np.cos(s), 5 * np.sin(s))
```

Animation Program Code

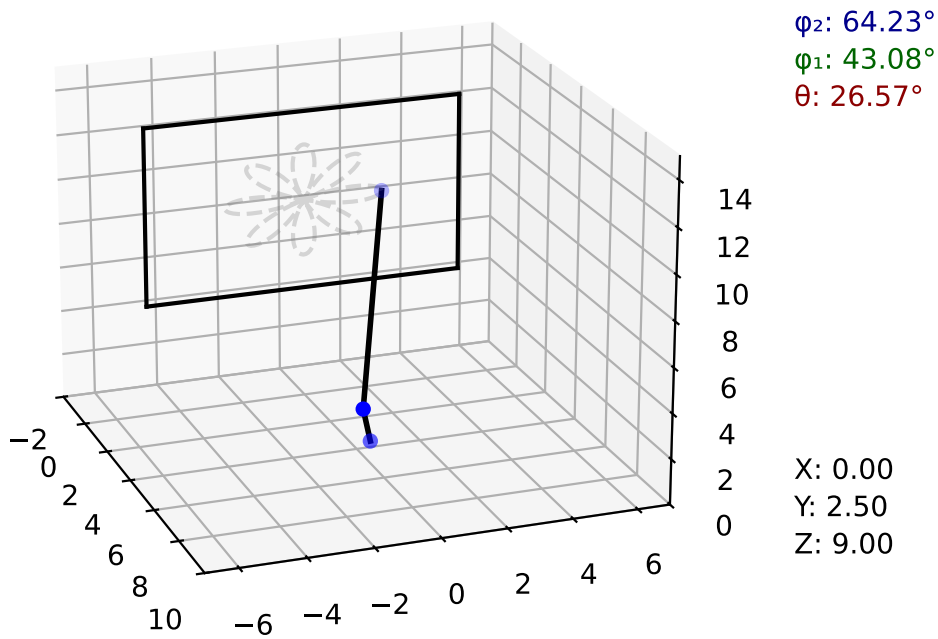
```
    elif 10 <= t < 11:
        s = (0.576 + pi) * (11 - t) - pi
        return transform(-1.52 * np.cos(s) - 2.72, 1.52 * np.sin(
    s) + 3.5)
    elif 11 <= t < 12:
        return transform(-1.2, 4.9434 - 1.4434 * (12 - t))
    elif 12 <= t < 13:
        return transform(-0.6 - 0.6 * (13 - t),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

Animation Program Code

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

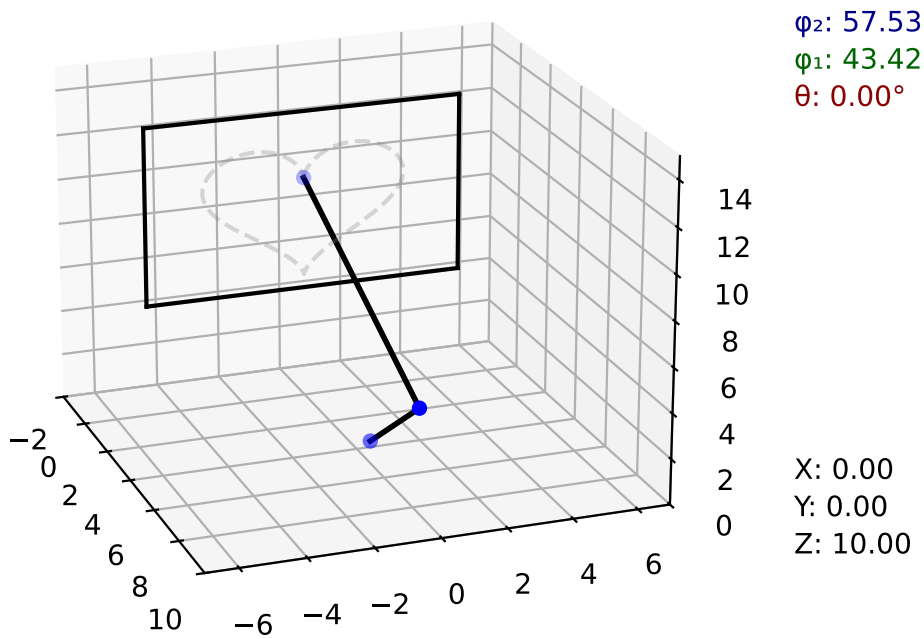
Python Simulation Codes and Graphical Representation

Python Animations



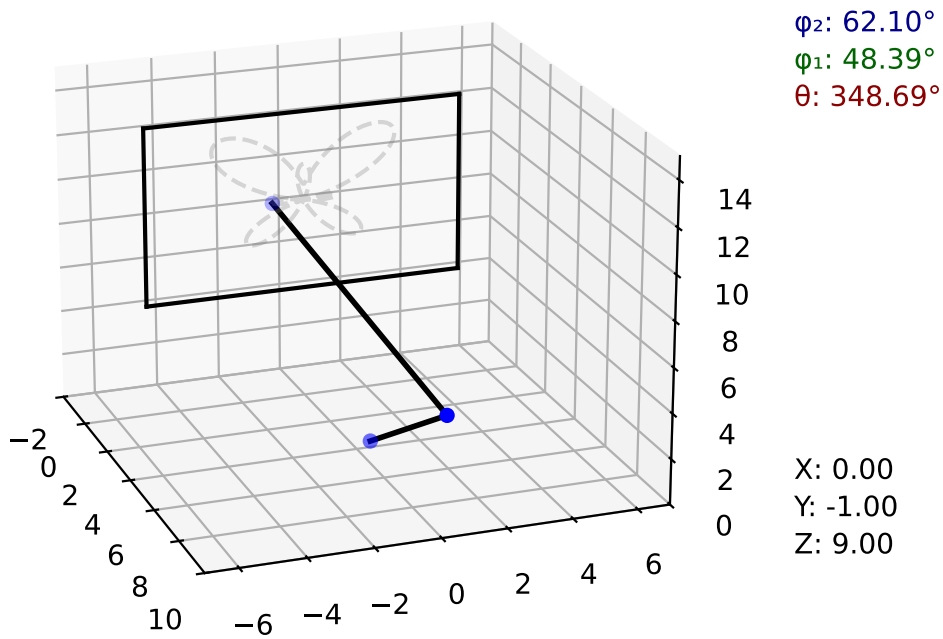
Python Simulation Codes and Graphical Representation

Python Animations



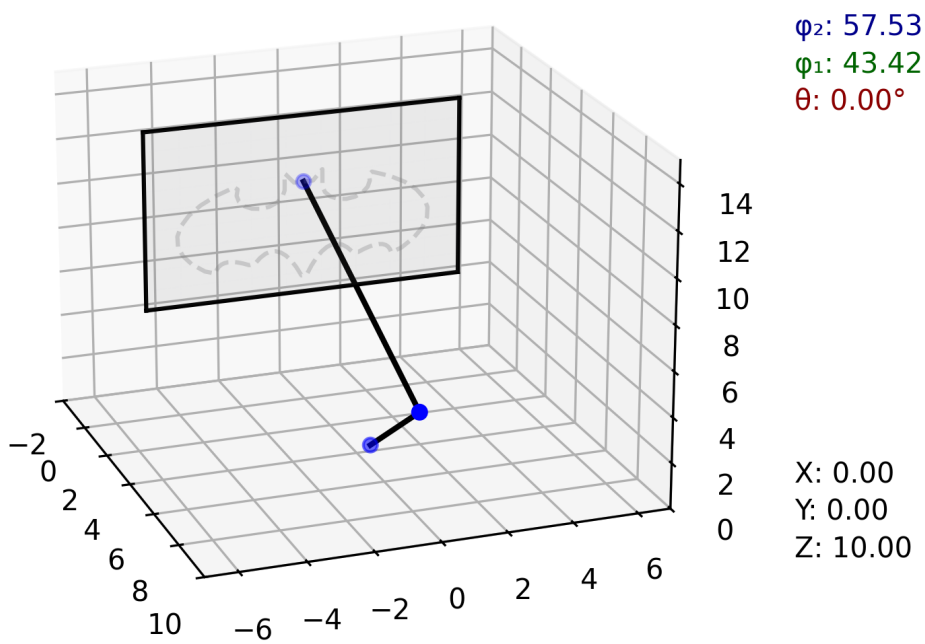
Python Simulation Codes and Graphical Representation

Python Animations

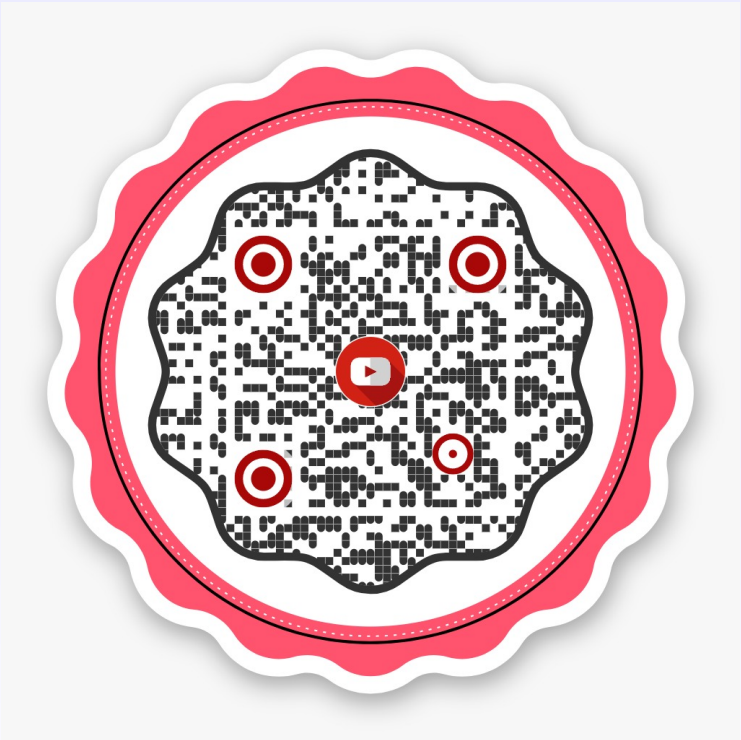


Python Simulation Codes and Graphical Representation

Python Animations



Thank You



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