Simulation and Visualization of a 2-Link Robotic Arm Using Forward Kinematics in Python

ABHIJITH S, ABHIJITH PRADEEP, ASHWIN S, ALDRICH JOBY

October 17, 2025

Abstract

This project presents the simulation and visualization of a 2-link robotic arm using forward kinematics implemented in Python. The goal is to calculate the positions of the robotic arm's joints and end-effector in 2D space based on given joint angles and link lengths and to animate its motion to gain intuitive understanding of the arm's kinematics.

1 Introduction

Robotic manipulators are widely used in automation and manufacturing. Understanding the forward kinematics of such arms is fundamental for control and simulation. This project implements forward kinematics for a planar 2-link arm and visualizes its motion using Python's matplotlib library.

2 Background

Forward kinematics involves calculating the position of the end-effector from the known joint parameters. For a 2-link arm, this involves simple trigonometric relationships based on link lengths and joint angles.

Figure 1: 2-link robotic arm

3 Project Objectives

- Implement forward kinematics for a 2-link planar robotic arm.
- Visualize the robotic arm using 2D plotting.
- Animate the arm movement with varying joint angles.
- Structure the project modularly for clarity and extension.

4 Methodology

4.1 Forward Kinematics

For a 2-link robotic arm with link lengths L_1 and L_2 and joint angles θ_1, θ_2, the joint and end-effector positions are:

$$x_1 = L_1 \cos(\theta_1)$$
$$y_1 = L_1 \sin(\theta_1)$$
$$x_2 = x_1 + L_2 \cos(\theta_1 + \theta_2)$$
$$y_2 = y_1 + L_2 \sin(\theta_1 + \theta_2)$$

where (x_1, y_1) is the first joint, and (x_2, y_2) is the end-effector position.

4.2 Visualization and Animation

The project uses matplotlib to plot the arm as connected line segments. Animation is performed by updating joint angles frame-by-frame using `FuncAnimation`.

5 Implementation

The project is divided into four modules. Below is the key module implementing forward kinematics:

Listing 1: Forward Kinematics Function

```
import math
import matplotlib.pyplot as plt

# Forward kinematics function
def forward_kinematics(L1, L2, theta1_deg, theta2_deg):
    # Convert degrees to radians
    theta1 = math.radians(theta1_deg)
    theta2 = math.radians(theta2_deg)

    # First joint position
    x1 = L1 * math.cos(theta1)
    y1 = L1 * math.sin(theta1)

    # End effector position
    x2 = x1 + L2 * math.cos(theta1 + theta2)
    y2 = y1 + L2 * math.sin(theta1 + theta2)

```

```python
    return (0, 0), (x1, y1), (x2, y2)

# Example parameters
L1 = 5  # Length of link 1
L2 = 3  # Length of link 2
theta1 = 45  # Angle of joint 1 in degrees
theta2 = 30  # Angle of joint 2 in degrees

# Get coordinates
base, joint, end_effector = forward_kinematics(L1, L2, theta1, theta2)

# Print the positions
print("Base position:", base)
print("Joint position:", joint)
print("End Effector position:", end_effector)

# Plot the robotic arm
x_values = [base[0], joint[0], end_effector[0]]
y_values = [base[1], joint[1], end_effector[1]]

plt.figure(figsize=(6,6))
plt.plot(x_values, y_values, '-o', linewidth=3, markersize=8)
plt.xlim(- (L1 + L2) - 1, (L1 + L2) + 1)
plt.ylim(- (L1 + L2) - 1, (L1 + L2) + 1)
plt.grid(True)
plt.title("2-Link Robotic Arm (Forward Kinematics)")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.gca().set_aspect('equal', adjustable='box')
plt.show()
```

Other modules handle static plotting, interactive input, and animation using this function.

6 Results

The simulation successfully calculates joint positions and animates the arm movement, providing an intuitive understanding of the relationship between joint angles and the robotic arm's configuration.

7 Conclusion

This project demonstrates forward kinematics for a 2-link robotic arm, with visualization and animation implemented in Python. The modular approach simplifies understanding and allows future extension.

8 Future Work

- Implement inverse kinematics to determine joint angles from desired end-effector positions.
- Extend the simulation to 3D robotic arms.

- Add real-time controls such as sliders for interactive manipulation.
- Integrate dynamic modeling and control algorithms.

9 References

- Craig, J. J. (2005). *Introduction to Robotics: Mechanics and Control*. Pearson.
- Matplotlib Documentation: `https://matplotlib.org/`
- Python Official Documentation: `https://docs.python.org/`