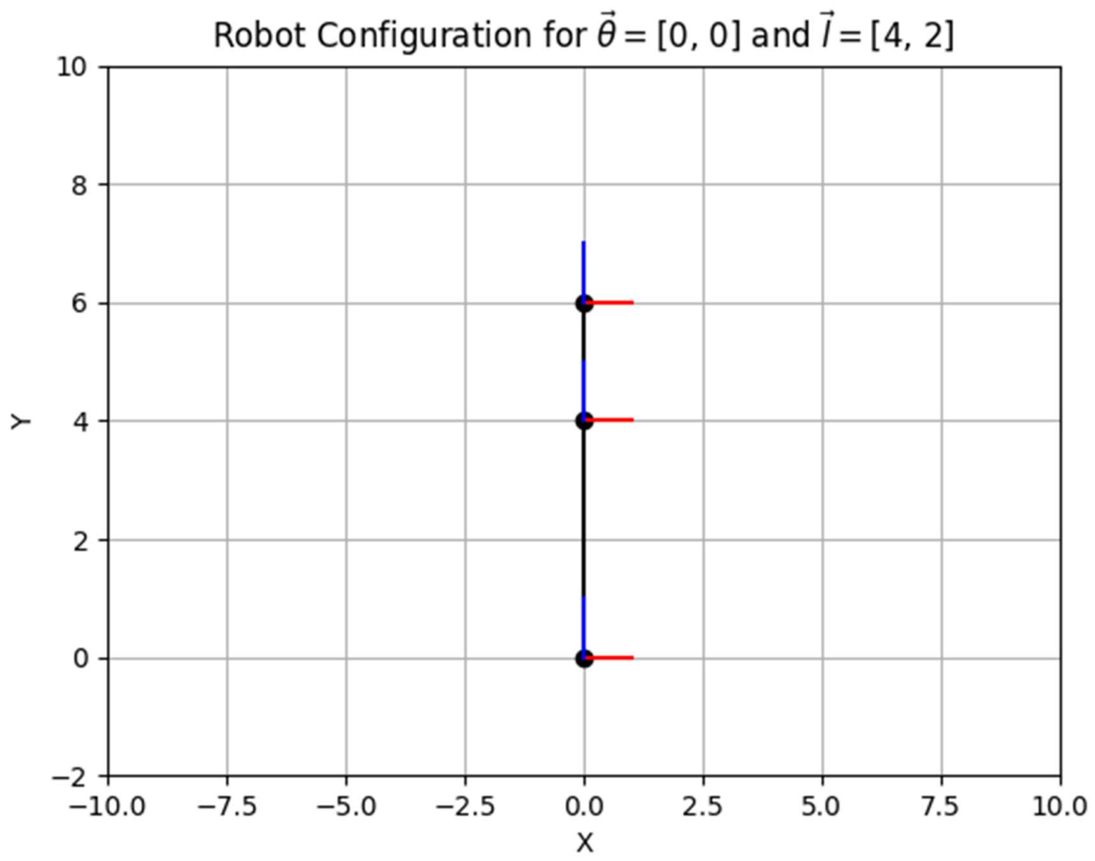
```python
import numpy as np
import matplotlib.pyplot as plt

def get_axes(p, R):
    x_axis = p + (R @ np.array([[1], [0]]).flatten().tolist())
    y_axis = p + (R @ np.array([[0], [1]]).flatten().tolist())
    return x_axis, y_axis

def R(theta):
    return np.array([[np.cos(theta), -np.sin(theta)],
                     [np.sin(theta), np.cos(theta)]])

def T(R, d):
    T = np.eye(3)
    T[0:2, 0:2] = R
    T[0:2, 2] = d.flatten()
    return T

def visualize_robot(joint_angles, link_lengths):

    # Get all frame rotation matrices
    Rlist = [R(angle) for angle in joint_angles]
    Rlist.append(R(0));

    # Get all frame displacement vectors
    dlist = list(link_lengths)
    dlist.insert(0, 0)

    # Compute the transformation matrices for each frame
    Tlist = []
    for i in range(len(Rlist)):
        d = np.array([[0], [dlist[i]]])
        Tlist.append(T(Rlist[i], d))

    # Iteratively compute the homogeneous coordinates of each frame and extract
the position and rotation matrices for each frame
    frame_pos = []
    frame_rot = []

    T_curr = np.eye(3)
    for Transformation in Tlist:
        T_curr = T_curr @ Transformation
        frame_pos.append(T_curr[0:2, 2].flatten().tolist())
        frame_rot.append(T_curr[0:2, 0:2])
    # Plot the robot arm as a series of frame locations
    plt.figure()
    plt.plot([coord[0] for coord in frame_pos], [coord[1] for coord in
frame_pos], 'ko-')

    # Plot the coordinate axes of each individual frames
    p = []
    for i in range(len(Rlist)):
        p = frame_pos[i]
        Rm = frame_rot[i]
        x_axis, y_axis = get_axes(p, Rm)
        plt.plot([p[0], x_axis[0]], [p[1], x_axis[1]], 'r-')
        plt.plot([p[0], y_axis[0]], [p[1], y_axis[1]], 'b-')

    # Plot formatting
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.title(r'Robot Configuration for $\vec{\theta} = $' + str([round(angle, 2)
for angle in joint_angles]) + r' and $\vec{l} = $' + str(link_lengths))
    plt.grid(True)
    plt.xlim([-10, 10])
    plt.ylim([-2, 10])

    return plt, p, np.arctan2(frame_rot[-1][1, 0], frame_rot[-1][0, 0])

# Example usage
joint_angles = [0, 0]
plt, end_effector_position, end_effector_theta = visualize_robot(joint_angles,
[4, 2])
plt.savefig('image1.png')

joint_angles = [-np.pi/4, -np.pi/2]
plt, end_effector_position, end_effector_theta = visualize_robot(joint_angles,
[4, 2])
plt.savefig('image2.png')

joint_angles = [np.pi/8, -2*np.pi/3]
plt, end_effector_position, end_effector_theta = visualize_robot(joint_angles,
[2, 3])
plt.savefig('image3.png')

joint_angles = [np.pi/3, float(-3) * np.pi/4]
plt, end_effector_position, end_effector_theta = visualize_robot(joint_angles,
[3, 3])
plt.savefig('image4.png')
```

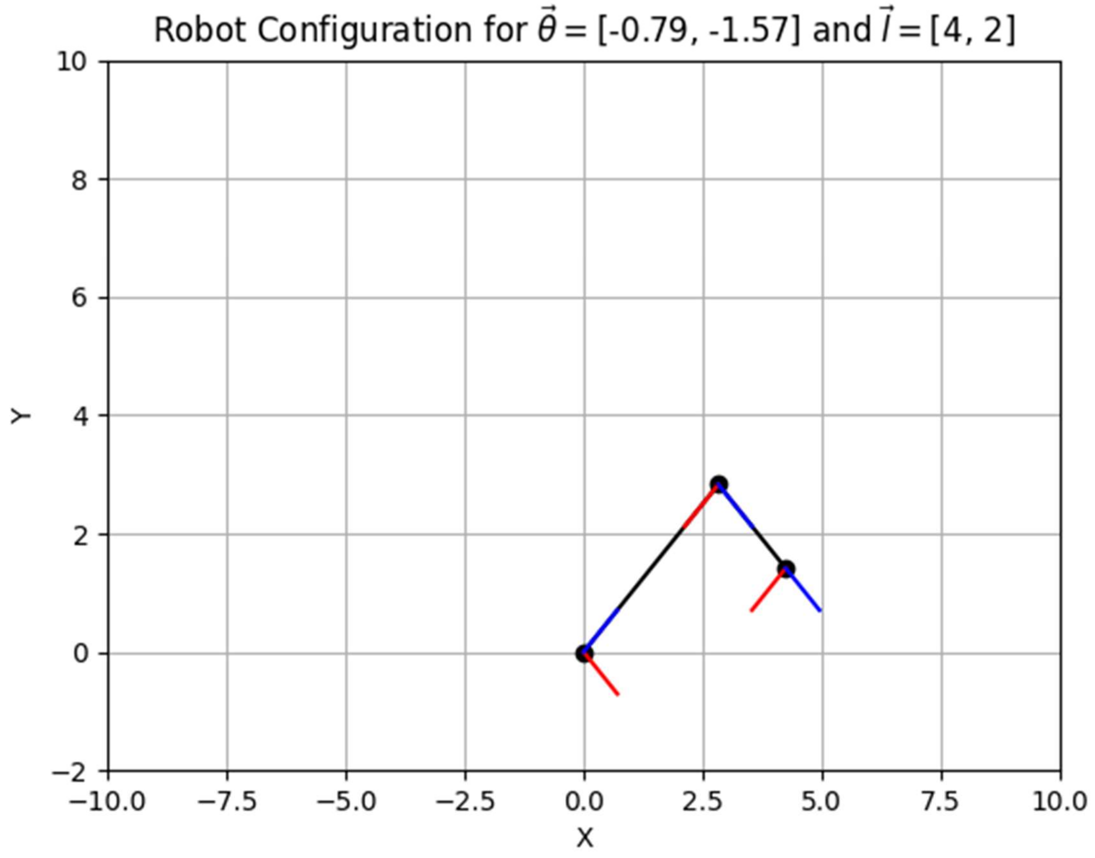
```python
print("End Effector Position:", end_effector_position)
print("End Effector Orientation (radians):", end_effector_theta)
```

Code available at gitrepo

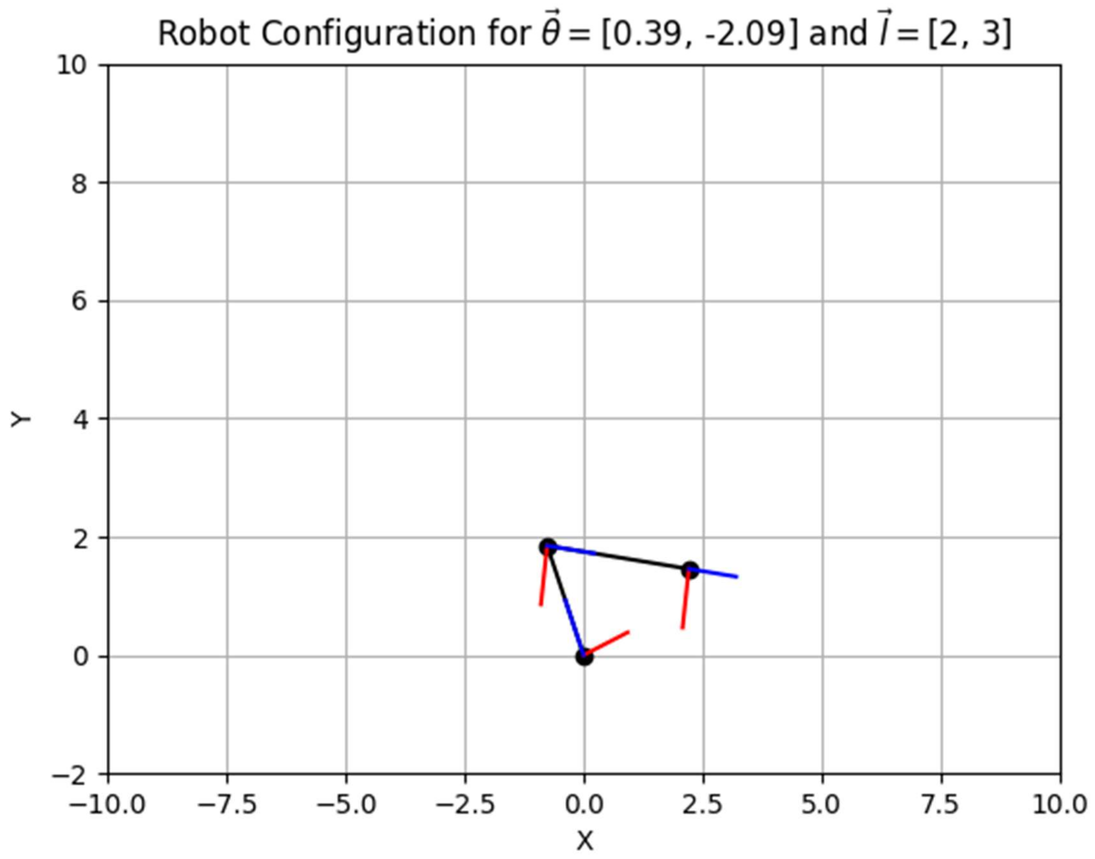
2a)


Robot Configuration for $\vec{\theta} = [0, 0]$ and $\vec{l} = [4, 2]$
Y
X
10
8
6
4
2
0
−2
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

2b)

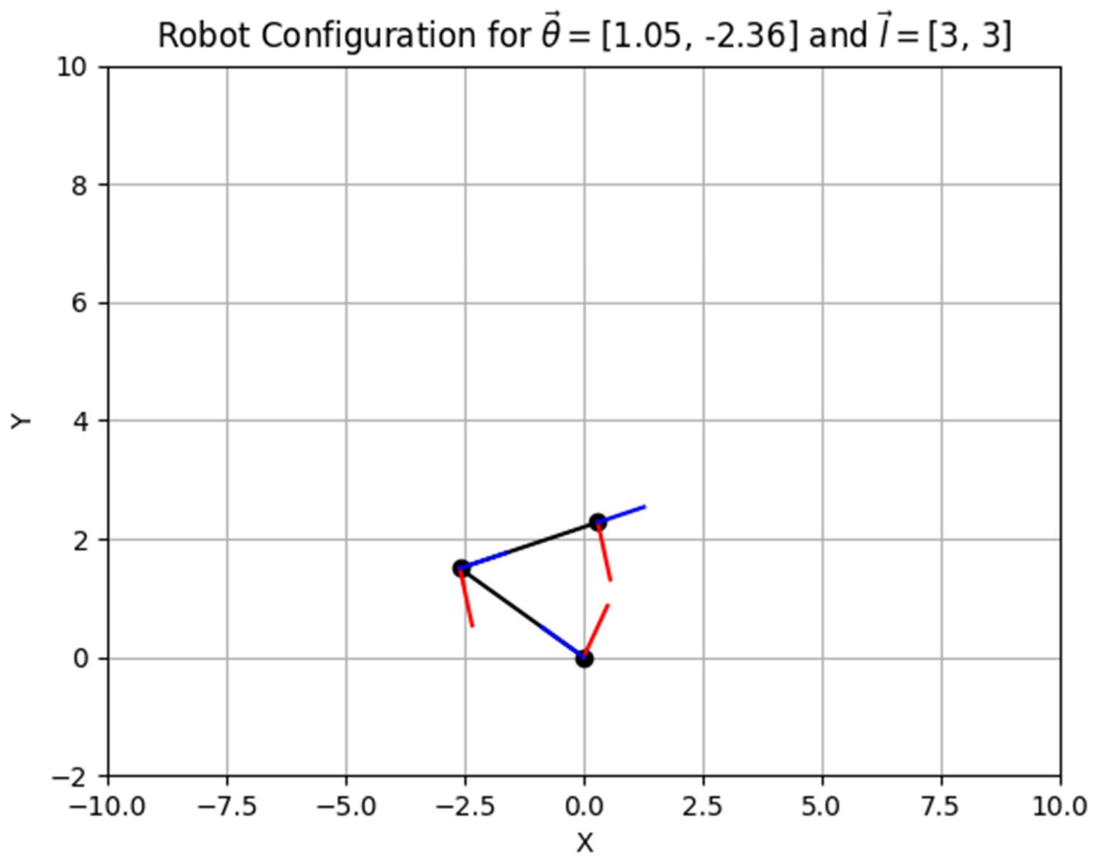

Robot Configuration for $\vec{\theta}$ = [-0.79, -1.57] and $\vec{l}$ = [4, 2]
Y
X
10
8
6
4
2
0
−2
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

2c)



Robot Configuration for $\vec{\theta} = [0.39, -2.09]$ and $\vec{l} = [2, 3]$
10
8
6
4
2
0
−2
Y
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
X

3)



Robot Configuration for $\vec{\theta} = [1.05, -2.36]$ and $\vec{l} = [3, 3]$
10
8
6
4
2
0
−2
Y
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
X

4)

Code

Code process:

1. Calibrate
2. Initialize the bus (establish communication with all servos, ensure they have calibration loaded)
3. Read and store the starting position
4. Move to the new pose (desired_position, which is the zero position of the robot as determined by the stored calibration)
5. Hold
6. Then return to the stored starting position.
7. Disable servo torque so that the robot can be positioned by hand

Setup was as before, robot arm clamped to desk. When the script was ran, the robot joints were more jerky than expected, and oscillations persisted for a significant period of time following an abrupt stop at desired position. Working theory is that the PID gains set by the utils script are well suited for the standard servos, but I purchased and installed the higher torque model. As a result, I messed around with the PID values until it seemed slightly better. No formal process was used, so its not likely to be optimal. Future use might require some HIL PID tuning.
Video of robot arm movement