ID6040: Introduction to Robotics

Ayush Mukund Jamdar EE20B018

March 26, 2023

1 Assignment 3

The assignment aims to let the learner implement an inverse kinematics solution to find joint angles of a two link serial manipulator trying to reach three desired points in the workspace. The method used is the Analytical approach using trigonometric equations.

2 The Assignment

2.1 The Problem

The module `inv_kin.py` implements the function `inv_kin_fn()`. The function takes the desired coordinated in the 2D space and returns the two joint angles required to get there. See how it does this in the following section.

2.2 Inverse Kinematics

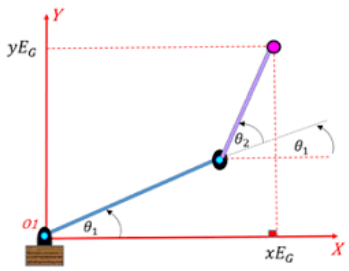

Figure 1: A graphical representation of the system.

The given system is represented in Figure 1. Let l_1 be the length of the arm from the origin and l_2 the length of the next one. It is easy to see that the x and y coordinates of the endpoint are given by -

$$x = l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2)$$
$$y = l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2)$$

Well, this looks like the forward kinematics problem where θ_1 and θ_2 are used to estimate end effector coordinates. For the current problem, we need to find θ_1 and θ_2 from the desired coordinates x_d and y_d.

Let's solve the trigonometric equations. Squaring the equations,

$$x_d^2 = l_1^2 \cos^2(\theta_1) + l_2^2 \cos^2(\theta_1 + \theta_2) + 2l_1 l_2 \cos\theta_1 \cos(\theta_1 + \theta_2)$$
$$y_d^2 = l_1^2 \sin^2(\theta_1) + l_2^2 \sin^2(\theta_1 + \theta_2) + 2l_1 l_2 \sin\theta_1 \sin(\theta_1 + \theta_2)$$

On adding the equations, it is easy to get $\cos\theta_2$

$$cos\theta_2 = \frac{x_d^2 + y_d^2 - l_1^2 - l_2^2}{2l_1 l_2}$$

One could simply take the arccos and get done with θ_2 but cosine inverse has a problem. Elbow-up and elbow-down configurations are the two solutions for any point in the dexterous workspace. arccos has a sign ambiguity for the two solutions since its principal angle range is 0 to π. So we take two more steps.

$$\sin\theta_2 = \pm\sqrt{1 - \cos^2\theta_2}$$

$$\theta_2 = atan2(\frac{\sin\theta_2}{\cos\theta_2})$$

Note the disposal of $atan2$. Unlike the usual arctan, $atan2$ in Python takes arguments as (x, y) instead of (x/y). Hence the information of four sign combinations is retained and $atan2$ can precisely point out which quadrant the angle belongs to. (arctan returns values in the range of its principle angle, $-\pi/2$ to $\pi/2$.)

Now, proceed to find θ_1. In the two equations we started with, expanding the sinusoid functions, there remain two equations in $\sin\theta_1$ and $\cos\theta_1$.

$$M = l_1 + l_2\cos\theta_2$$
$$N = l_2\sin\theta_2$$

After substitution the equations become,

$$M\cos\theta_1 - N\sin\theta_1 = x_d$$

$$N\cos\theta_1 + M\sin\theta_1 = y_d$$

Next, finding θ_1. A polar coordinate substitution for M and N yields

$$r\cos(\gamma + \theta_1) = x_d$$

$$r\sin(\gamma + \theta_1) = y_d$$

where,

$$r = \sqrt{M_2 + N^2}$$
$$\gamma = atan2(N, M)$$

Finally,

$$\theta_1 = atan2(\frac{y_d}{x_d}) - \gamma$$

2.3 Simulation

The above equations are implemented in the Python module `inv_kin.py` as follows. Note that an exception is raised if the desired point is outside the reachable workspace. For points within the dexterous workspace, two solutions are possible but since only the end effector position is desired, any of the two solution sets works. I arbitrarily return the set with positive sine.

```
def inv_kin_fn(goal_position):
    # Compute joint angles [in degrees] to reach desired position [in meters]
    x_desired = goal_position[0]
    y_desired = goal_position[1]

    l1 = robot_params.link_1_length
    l2 = robot_params.link_2_length

    # Algebraic solution
```

```
cos_theta_2 = (x_desired**2 + y_desired**2 - l1**2 - l2**2) / (2 * l1 * l2)

if np.abs(cos_theta_2) > 1:
    # no solution
    raise ValueError("Desired point out of workspace: No Solution")
else:
    # solutions possible
    sin_theta_2 = [
        np.sqrt(1 - cos_theta_2**2),
        -np.sqrt(1 - cos_theta_2**2),
    ]  # two solutions

    theta_2 = np.arctan2(
        sin_theta_2, cos_theta_2
    )  # arctan2 is a 4 quadrant inverse
    # in radian

    M = l1 + l2 * np.cos(theta_2)
    N = l2 * np.sin(theta_2)

    theta_1 = np.arctan2(y_desired, x_desired) - np.arctan2(N, M)

    theta_1 = [t * 180 / math.pi for t in theta_1]
    theta_2 = [t * 180 / math.pi for t in theta_2]

# arbitrarily take the first solution set since only the end effector
# position is required
print("Desired joint angles", [theta_1[0], theta_2[0]])
return [theta_1[0], theta_2[0]]
```

It was observed that the default Simulation Time $dt = 0.01ms$ was not enough for the first two goals to be reached in time. Hence, non-default settings had to be used without any compromise on the desired accuracy of 0.01. In a few trials it was observed that a Simulation time of at least 80 ms was required. (Reference: CoppeliaSim Simulation Dialog)

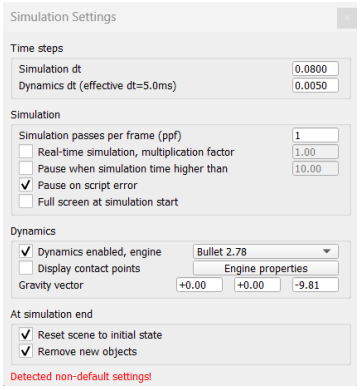

Figure 2: Setting Simulation Time

2.4 Simulation Results

The below screenshot shows the successful run of the simulation with correct end effector position for the three goals.

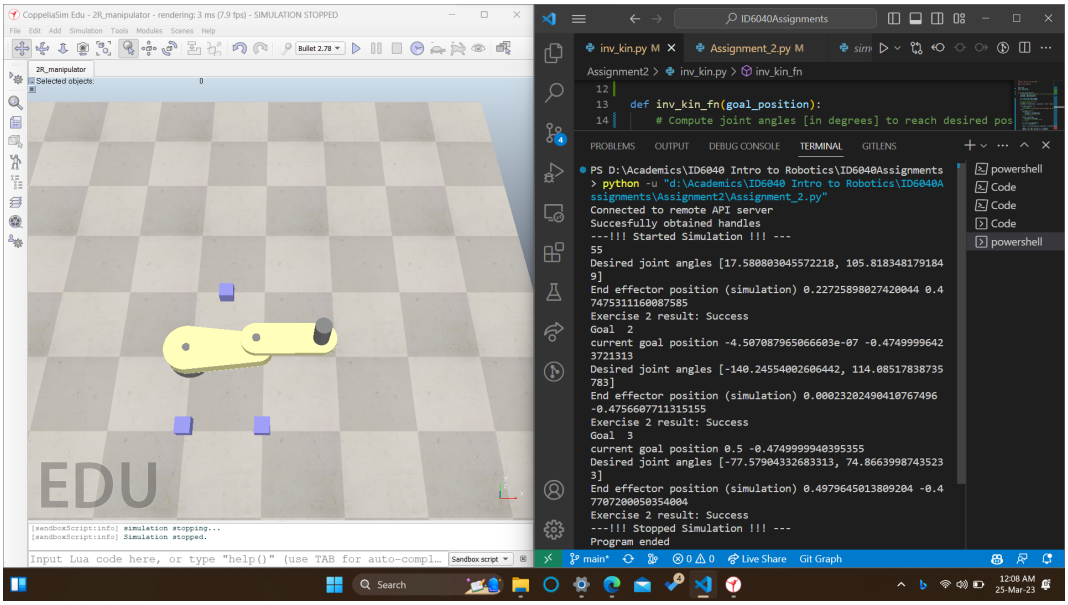

Figure 3: Results

2.5 Conclusion

An analytical approach towards finding joint angles of a two-link manipulator is demonstrated in CoppeliaSim. A few more details over the last assignment over simulation errors and settings are observed.