Ryan Huang
4/22/2024

Project 2 - Prompt 2: Simulating the Forward Dynamics of a Planar 3R Arm

Background

Based on the prompt, the simulation of a falling 3R arm is analogous to the simulation of a triple pendulum due to each link having a concentrated mass at the tip. The other assumptions include zero joint friction and the arm being fully stretched out to the right at the start. Additionally, the masses and link lengths can be freely chosen. For simplicity, the lengths and masses were treated as one.

Based on this information, the following diagram was used to derive the equations of motion used to model the forward dynamics of the system.

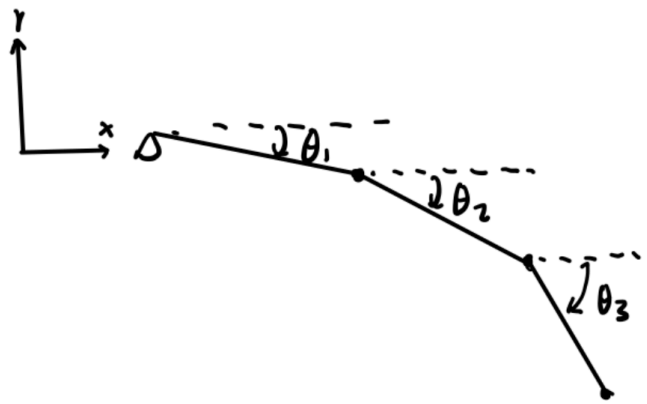

Figure 1. Diagram of the 3R Arm along with positive coordinate convention for position and angle

From Figure 1, the positions and velocities of each mass can be solved below.

$$x_1 = \cos\theta_1$$
$$y_1 = -\sin\theta_1$$
$$x_2 = \cos\theta_1 + \cos\theta_2$$
$$y_2 = -\sin\theta_1 - \sin\theta_2$$
$$x_3 = \cos\theta_1 + \cos\theta_2 + \cos\theta_3$$
$$y_3 = -\sin\theta_1 - \sin\theta_2 - \sin\theta_3$$

$$\dot{x}_1 = -\dot{\theta}_1 \sin\theta_1$$
$$\dot{y}_1 = -\dot{\theta}_1 \cos\theta_1$$
$$\dot{x}_2 = -\dot{\theta}_1 \sin\theta_1 - \dot{\theta}_2 \sin\theta_2$$
$$\dot{y}_2 = -\dot{\theta}_1 \cos\theta_1 - \dot{\theta}_2 \cos\theta_2$$
$$\dot{x}_3 = -\dot{\theta}_1 \sin\theta_1 - \dot{\theta}_2 \sin\theta_2 - \dot{\theta}_3 \sin\theta_3$$
$$\dot{y}_3 = -\dot{\theta}_1 \cos\theta_1 - \dot{\theta}_2 \cos\theta_2 - \dot{\theta}_3 \cos\theta_3$$

Figure 2. Positions and velocities of each mass defined by their angle and angular velocities

Equations of Motion

As a falling 3R arm is rather complex to solve with Newton's laws, a lagrangian approach was used to solve for the equations of motion. Using the standard equations for kinetic energy and potential energy along with the coordinates defined in Figure 2, the lagrangian was derived by solving for the total kinetic energy and subtracting the total potential energy.

$$\mathcal{L} = \frac{3}{2}\dot{\theta}_1^2 + \dot{\theta}_2^2 + 2\dot{\theta}_1\dot{\theta}_2\cos(\theta_1-\theta_2) + \dot{\theta}_1\dot{\theta}_3\cos(\theta_1-\theta_3) + \dot{\theta}_2\dot{\theta}_3\cos(\theta_2-\theta_3) + \frac{1}{2}\dot{\theta}_3^2 + 3g\sin\theta_1 + 2g\sin\theta_2 + g\sin\theta_3$$

With this, the equations of motion can be derived by treating the joint angles as the generalized coordinates to use in the following equation.

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}}\right) - \frac{\partial \mathcal{L}}{\partial q} = 0$$

This resulted in three coupled, second order ordinary differential equations (ODE) shown below.

$$3\ddot{\theta}_1 + 2\ddot{\theta}_2\cos(\theta_1-\theta_2) - 2\dot{\theta}_2\sin(\theta_1-\theta_2)(\dot{\theta}_1-\dot{\theta}_2) + \ddot{\theta}_3\cos(\theta_1-\theta_3) - \dot{\theta}_3\sin(\theta_1-\theta_3)(\dot{\theta}_1-\dot{\theta}_3) + 2\dot{\theta}_1\dot{\theta}_2\sin(\theta_1-\theta_2) + \dot{\theta}_1\dot{\theta}_3\sin(\theta_1-\theta_3) - 3g\cos\theta_1 = 0$$

$$2\ddot{\theta}_2 + 2\ddot{\theta}_1\cos(\theta_1-\theta_2) - 2\dot{\theta}_1\sin(\theta_1-\theta_2)(\dot{\theta}_1-\dot{\theta}_2) + \ddot{\theta}_3\cos(\theta_2-\theta_3) + \dot{\theta}_3\sin(\theta_2-\theta_3)(\dot{\theta}_2-\dot{\theta}_3) - 2\dot{\theta}_1\dot{\theta}_2\sin(\theta_1-\theta_2) + \dot{\theta}_2\dot{\theta}_3\sin(\theta_2-\theta_3) - 2g\cos\theta_2 = 0$$

$$\ddot{\theta}_3 + \ddot{\theta}_1\cos(\theta_1-\theta_3) - \dot{\theta}_1\sin(\theta_1-\theta_3)(\dot{\theta}_1-\dot{\theta}_3) + \ddot{\theta}_2\cos(\theta_2-\theta_3) - \dot{\theta}_2\sin(\theta_2-\theta_3)(\dot{\theta}_2-\dot{\theta}_3) - \dot{\theta}_1\dot{\theta}_3\sin(\theta_1-\theta_3) - \dot{\theta}_2\dot{\theta}_3\sin(\theta_2-\theta_3) - g\cos\theta_3 = 0$$

Python Implementation

The simulation was done in Python by plotting frames of each position using the matplotlib library. The equations of motion were solved numerically using Euler's method. In the code, several functions were defined for convenience.

- **Global Variables**

  There are several global variables defined for constants and variables that are more convenient to have in the global scope.
  - G: Acceleration of Gravity
  - DT: Time Step
  - N: Number of Iterations
  - pos, vel, acc: Lists of Angular Position, Velocity, and Acceleration for each mass

- **def equations(q1, q2, q3, dq1, dq2, dq3, ddq1, ddq2, ddq3):**

This function contained the three coupled ODE's and used the joint angle position, velocity, and acceleration as parameters to solve for the angular accelerations in the next iteration. It returns a list containing the updated accelerations

- **def update(accel):**

  This function takes in the updated accelerations calculated from the equations function and multiplies it by a constant time step to determine the new velocities. The velocities are then multiplied by the time step again to determine the new position. The positions and velocities are then updated for the next iteration to use in the equations function.

- **def plot():**

  This function contains all of the plotting functionality necessary to plot the 3R arm for each iteration. Every plot was saved as a jpg file and stitched together in iMovie to generate a smooth animation of the pendulum. The plot size, title, axis title, and link positions are all defined here for convenience.

Simulation Results

The results of the simulation were overall successful as the arms movement was relatively close to how it might move in reality. As there is no joint friction, the arm will swing forever with chaotic behavior. The initial conditions assumed a fully stretched out arm, so the angles for each arm were 0 degrees. The simulation ran for 300 iterations as Pycharm and the computer used was rather low on computational power. The time step used was 0.01 seconds.

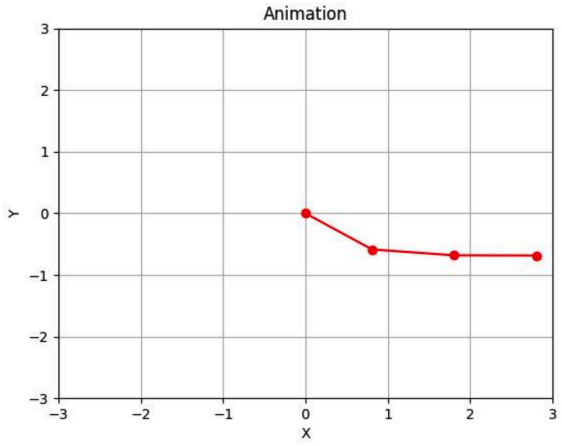

Figure 3. The 3R Arm falling shortly after the simulation began

Figure 3 is a single frame from the simulation shortly after it began. The movement of the arm looks realistic to how a 3R arm might fall in reality. Any inaccuracies in the simulation can be explained by the size of the time step, the implementation of the equations, or the imprecision in

Python's calculations. Additionally, there may be minor errors in the equations of motions themselves that may have been overlooked. However, the simulation itself looks fairly reasonable.

Link to Video + Python File
https://drive.google.com/drive/folders/1AQ4Bkzi6WL5OgTWkFBQroITepgbZD9Q4?usp=drive_link

The video is best played at 2x speed as the video editor I was using (iMovie) was unable to speed up the image duration any faster. The minimum duration for each image is 0.1 seconds.