ME 5250: Robot Mechanics and Control
Spring: 2024

Project-2: 3R Open-chain free fall Simulation

Report By

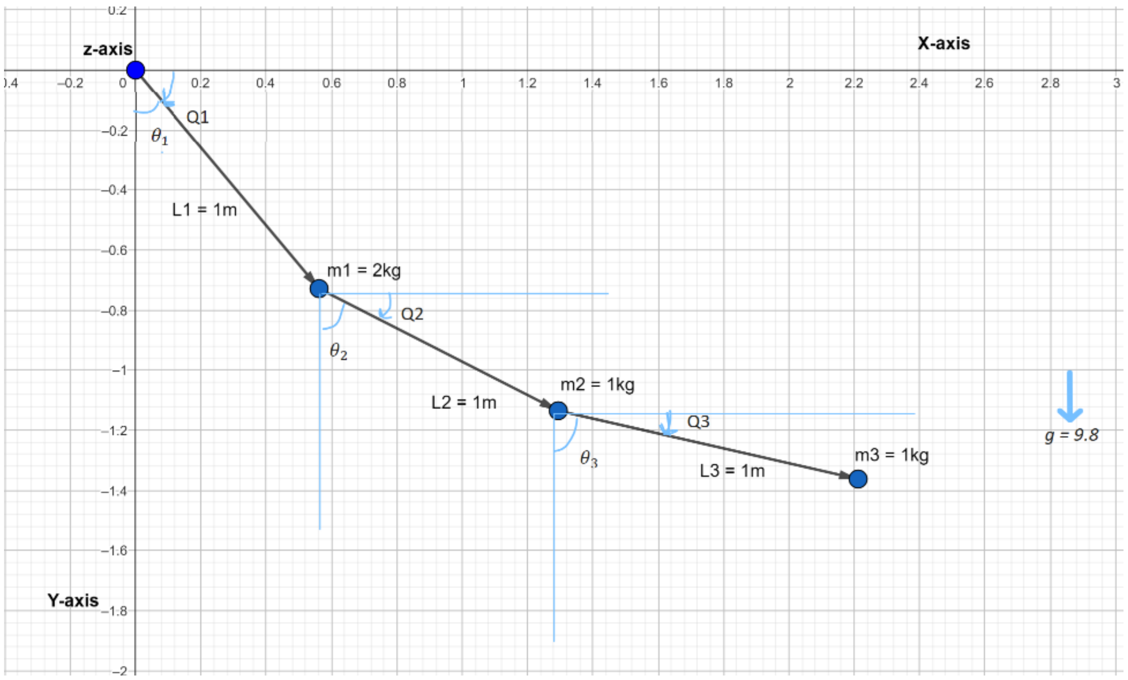
Tirth Snehal Patel – (NUID 002251943)

Problem Statement

Section 8.5 in the textbook discusses solving the forward dynamics problem numerically. Consider a planar 3R arm, initially with all joint angles zero (arm stretched out to the right). Simulate the arm falling under gravity in the case there is no joint friction. Each link has a concentrated mass at its Bp. You should pick the values for the link lengths and masses. Submit your code listing as an appendix (gradescope can run similarity checks between assignments to help detect copied code). Keep any functions or subroutines all in the same single file. If we have concerns that the code may not function, please be prepared to send your source code file upon request, but it need not be submitted initially to gradescope.

Explanation

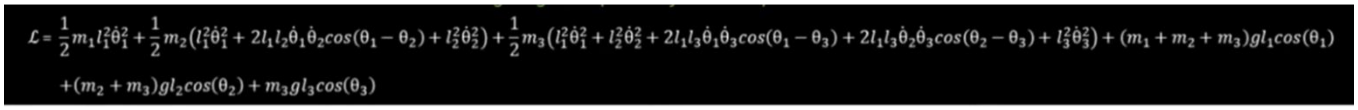
The below image best describes the 3R planar robot that we are planning to manipulate.



As shown in the figure our link lengths are of 1m and the point mass at the end of the links are Mass #1 = 2kg , Mass #2 = 1kg, and Mass #3 = 1kg respectively. In the initial state, the arm is resting in a stretched-out +x-direction. Our goal is to simulate it under the influence of gravity given that there is no joint friction. To simulate this arm the way we want, we will need the joint accelerations of each joint. To calculate that we will start with calculating the Lagrangian formulation of this 3R planar robot.

The formula of the lagrangian as we know from the textbook is: $\mathcal{L}(q, \dot{q}) = \mathcal{K}(q, \dot{q}) - \mathcal{P}(q).$

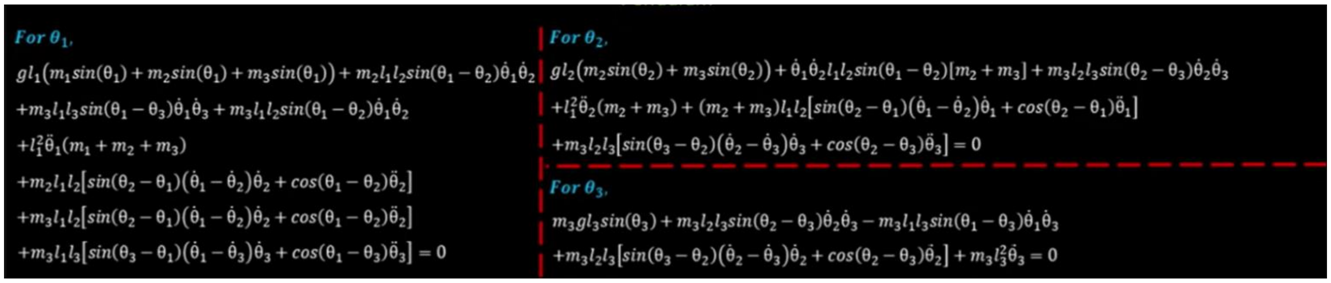
After hand calculations of the kinetic and potential energies of the system at each joints and rotation, we



$$\mathcal{L} = \frac{1}{2}m_1 l_1^2\dot\theta_1^2 + \frac{1}{2}m_2\left(l_1^2\dot\theta_1^2 + 2l_1l_2\dot\theta_1\dot\theta_2 cos(\theta_1-\theta_2) + l_2^2\dot\theta_2^2\right) + \frac{1}{2}m_3\left(l_1^2\dot\theta_1^2 + l_2^2\dot\theta_2^2 + 2l_1l_3\dot\theta_1\dot\theta_3 cos(\theta_1-\theta_3) + 2l_1l_3\dot\theta_2\dot\theta_3 cos(\theta_2-\theta_3) + l_3^2\dot\theta_3^2\right) + (m_1+m_2+m_3)gl_1cos(\theta_1)$$
$$+(m_2+m_3)gl_2cos(\theta_2) + m_3gl_3cos(\theta_3)$$

get the following Lagrangian formulation*. (Writing in latex would be taxing, hence attaching a screenshot. All hand calculations are attached in the appendix)*

From the above obtained lagrangian we can calculate joint torques for each thetas by using the formula of euler-lagrange's equation of motion. (*Again, below screenshot from my notion shows the joint torques*)



For θ_1,

$$gl_1(m_1sin(\theta_1) + m_2sin(\theta_1) + m_3sin(\theta_1)) + m_2l_1l_2sin(\theta_1-\theta_2)\dot\theta_1\dot\theta_2$$
$$+m_3l_1l_3sin(\theta_1-\theta_3)\dot\theta_1\dot\theta_3 + m_3l_1l_2sin(\theta_1-\theta_2)\dot\theta_1\dot\theta_2$$
$$+l_1^2\ddot\theta_1(m_1+m_2+m_3)$$
$$+m_2l_1l_2[sin(\theta_2-\theta_1)(\dot\theta_1-\dot\theta_2)\dot\theta_2 + cos(\theta_1-\theta_2)\ddot\theta_2]$$
$$+m_3l_1l_2[sin(\theta_2-\theta_1)(\dot\theta_1-\dot\theta_2)\dot\theta_2 + cos(\theta_1-\theta_2)\ddot\theta_2]$$
$$+m_3l_1l_3[sin(\theta_3-\theta_1)(\dot\theta_1-\dot\theta_3)\dot\theta_3 + cos(\theta_1-\theta_3)\ddot\theta_3] = 0$$

For θ_2,

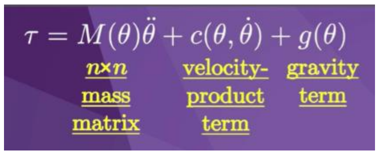
$$gl_2(m_2sin(\theta_2) + m_3sin(\theta_2)) + \dot\theta_1\dot\theta_2l_1l_2sin(\theta_1-\theta_2)[m_2+m_3] + m_3l_2l_3sin(\theta_2-\theta_3)\dot\theta_2\dot\theta_3$$
$$+l_1^2\ddot\theta_2(m_2+m_3) + (m_2+m_3)l_1l_2[sin(\theta_2-\theta_1)(\dot\theta_1-\dot\theta_2)\dot\theta_1 + cos(\theta_2-\theta_1)\ddot\theta_1]$$
$$+m_3l_2l_3[sin(\theta_3-\theta_2)(\dot\theta_2-\dot\theta_3)\dot\theta_3 + cos(\theta_2-\theta_3)\ddot\theta_3] = 0$$

For θ_3,

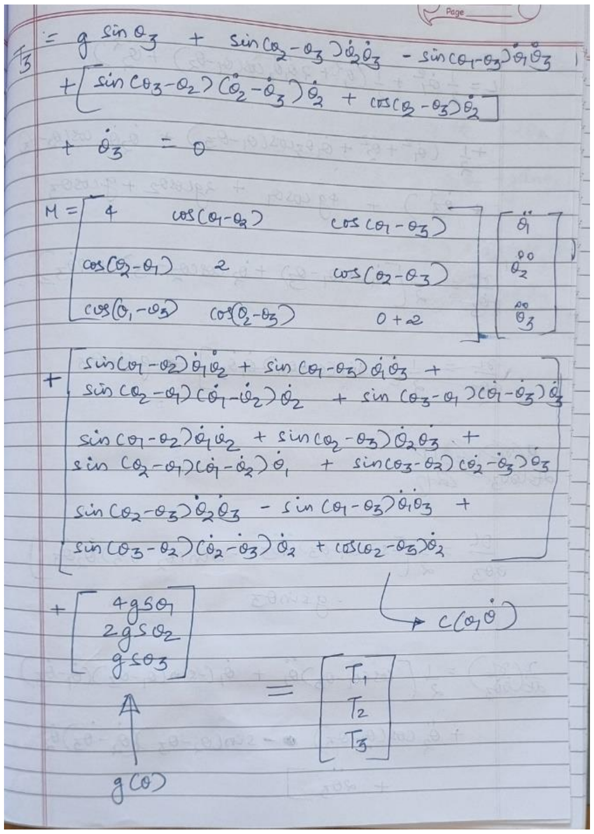
$$m_3gl_3sin(\theta_3) + m_3l_2l_3sin(\theta_2-\theta_3)\dot\theta_2\dot\theta_3 - m_3l_1l_3sin(\theta_1-\theta_3)\dot\theta_1\dot\theta_3$$
$$+m_3l_2l_3[sin(\theta_3-\theta_2)(\dot\theta_2-\dot\theta_3)\dot\theta_2 + cos(\theta_2-\theta_3)\ddot\theta_2] + m_3l_3^2\ddot\theta_3 = 0$$

Here we have taken the rhs as zero because the robot is at rest and no joint torques are acting initially.

From the given larger equations we can calculate the mass matrix, Coriolis centrifugal forces, and gravity vector. As we know from the formula. By hand calculation below notebook image is self-explanatory.



$$\tau = \underbrace{M(\theta)}_{\text{n×n mass matrix}}\ddot\theta + \underbrace{c(\theta,\dot\theta)}_{\text{velocity-product term}} + \underbrace{g(\theta)}_{\text{gravity term}}$$



$$\tau_3 = g\sin\theta_3 + \sin(\theta_2-\theta_3)\dot\theta_2\dot\theta_3 - \sin(\theta_1-\theta_3)\dot\theta_1\dot\theta_3$$
$$+\left[\sin(\theta_3-\theta_2)(\dot\theta_2-\dot\theta_3)\dot\theta_2 + \cos(\theta_2-\theta_3)\dot\theta_2\right]$$
$$+\dot\theta_3 = 0$$

$$M = \begin{bmatrix} 4 & \cos(\theta_1-\theta_2) & \cos(\theta_1-\theta_3) \\ \cos(\theta_2-\theta_1) & 2 & \cos(\theta_2-\theta_3) \\ \cos(\theta_1-\theta_3) & \cos(\theta_2-\theta_3) & 0+2 \end{bmatrix}\begin{bmatrix}\ddot\theta_1 \\ \ddot\theta_2 \\ \ddot\theta_3\end{bmatrix}$$

$$+\begin{bmatrix} \sin(\theta_1-\theta_2)\dot\theta_1\dot\theta_2 + \sin(\theta_1-\theta_3)\dot\theta_1\dot\theta_3 + \sin(\theta_2-\theta_1)(\dot\theta_1-\dot\theta_2)\dot\theta_2 + \sin(\theta_3-\theta_1)(\dot\theta_1-\dot\theta_3)\dot\theta_3 \\ \sin(\theta_1-\theta_2)\dot\theta_1\dot\theta_2 + \sin(\theta_2-\theta_3)\dot\theta_2\theta_3 + \sin(\theta_2-\theta_1)(\dot\theta_1-\dot\theta_2)\dot\theta_1 + \sin(\theta_3-\theta_2)(\dot\theta_2-\dot\theta_3)\theta_3 \\ \sin(\theta_2-\theta_3)\dot\theta_2\dot\theta_3 - \sin(\theta_1-\theta_3)\dot\theta_1\theta_3 + \sin(\theta_3-\theta_2)(\dot\theta_2-\dot\theta_3)\dot\theta_2 + \cos(\theta_2-\theta_3)\dot\theta_2 \end{bmatrix} \to c(\theta,\dot\theta)$$

$$+\underbrace{\begin{bmatrix} 4gs\theta_1 \\ 2gs\theta_2 \\ gs\theta_3 \end{bmatrix}}_{g(\theta)} = \begin{bmatrix} T_1 \\ T_2 \\ T_3 \end{bmatrix}$$

Given the joint angles, velocities, gravity and input torques we can compute the joint accelerations as:

Acceleration = inv(Mass Matrix) * (Torque – Coriolis Force + Gravity)

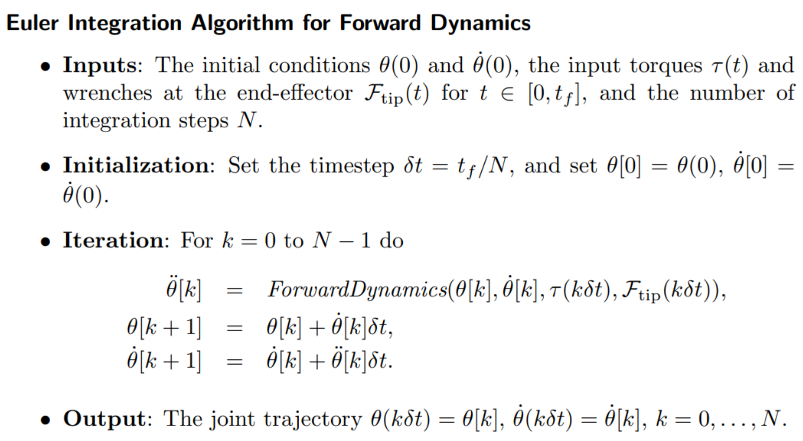
System Design

Now that we have obtained our joint accelerations, we need to put those values into a loop where the robot joints would use those values and act accordingly.

As suggest by Professor Whitney I have created my own function called ThreeLinkDynamics that takes joint angles, joint velocities and joint torques as arguments, calculates the mass matrix, Coriolis forces and gravity vector and then it returns the joint accelerations as variable ddQ.

Once we get out vector ddQ, initiating a continuous loop in the main code, we use the forward euler dynamics method as shown in the formula below

Once we get these continuous values of joint acceleration and joint angles, we calculate the positions of each joint using sine and cosine of the angles and hence finally plotting the graph of the same using appropriate markers and linewidths. Also we have added the feature of deleting the plots generated previously hence forming a continuous animation and not a continuous subplot. We have added a pause as well at the end of the loop to make it look smoother.



Euler Integration Algorithm for Forward Dynamics

- **Inputs**: The initial conditions $\theta(0)$ and $\dot{\theta}(0)$, the input torques $\tau(t)$ and wrenches at the end-effector $\mathcal{F}_{\text{tip}}(t)$ for $t \in [0, t_f]$, and the number of integration steps N.
- **Initialization**: Set the timestep $\delta t = t_f/N$, and set $\theta[0] = \theta(0)$, $\dot{\theta}[0] = \dot{\theta}(0)$.
- **Iteration**: For $k = 0$ to $N-1$ do

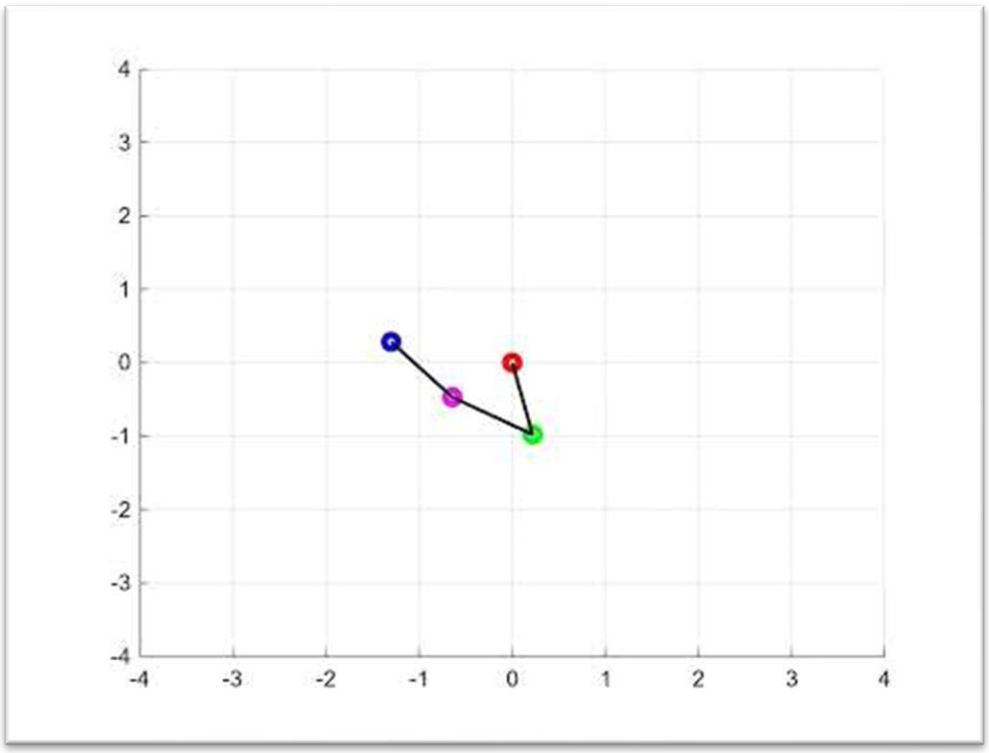
$$\begin{aligned} \ddot{\theta}[k] &= \mathit{ForwardDynamics}(\theta[k], \dot{\theta}[k], \tau(k\delta t), \mathcal{F}_{\text{tip}}(k\delta t)), \\ \theta[k+1] &= \theta[k] + \dot{\theta}[k]\delta t, \\ \dot{\theta}[k+1] &= \dot{\theta}[k] + \ddot{\theta}[k]\delta t. \end{aligned}$$

- **Output**: The joint trajectory $\theta(k\delta t) = \theta[k]$, $\dot{\theta}(k\delta t) = \dot{\theta}[k]$, $k = 0, \ldots, N$.

Below is the **pseudo code** of the whole system:

```
1. Initialize the environment:
   - Clear the command window, close all figures, and clear all variables.
   - Set the default figure window style to 'docked'.
   - Create a new figure for plotting, set the axes limits, and turn on the grid.

2. Define the main simulation loop:
   - Set initial joint angles, joint velocities, external torques, and time variables.
   - Enter an infinite loop to continuously update the robot configuration and plot it.
      a. Compute trigonometric functions of joint angles.
      b. Set external torques (zero for freefall).
      c. Compute joint accelerations using dynamics equations.
      d. Integrate joint velocities to update joint angles.
      e. Plot the robot configuration.
      f. Pause to control animation speed.

3. Define the function to compute joint accelerations (dynamics):
   - Extract joint angles and velocities from input vectors.
   - Compute trigonometric functions of joint angles.
   - Define gravity acceleration.
   - Define mass matrix M, Coriolis and centrifugal matrix C, and gravity vector G.
   - Compute joint accelerations using dynamic equation: M * ddQ + C + G = T.

4. Define the function to plot the robot configuration:
   - Extract joint angles from input vector.
   - Compute the coordinates of robot links based on joint angles.
   - Plot the robot links and joints.

5. End of script.
```

Output:



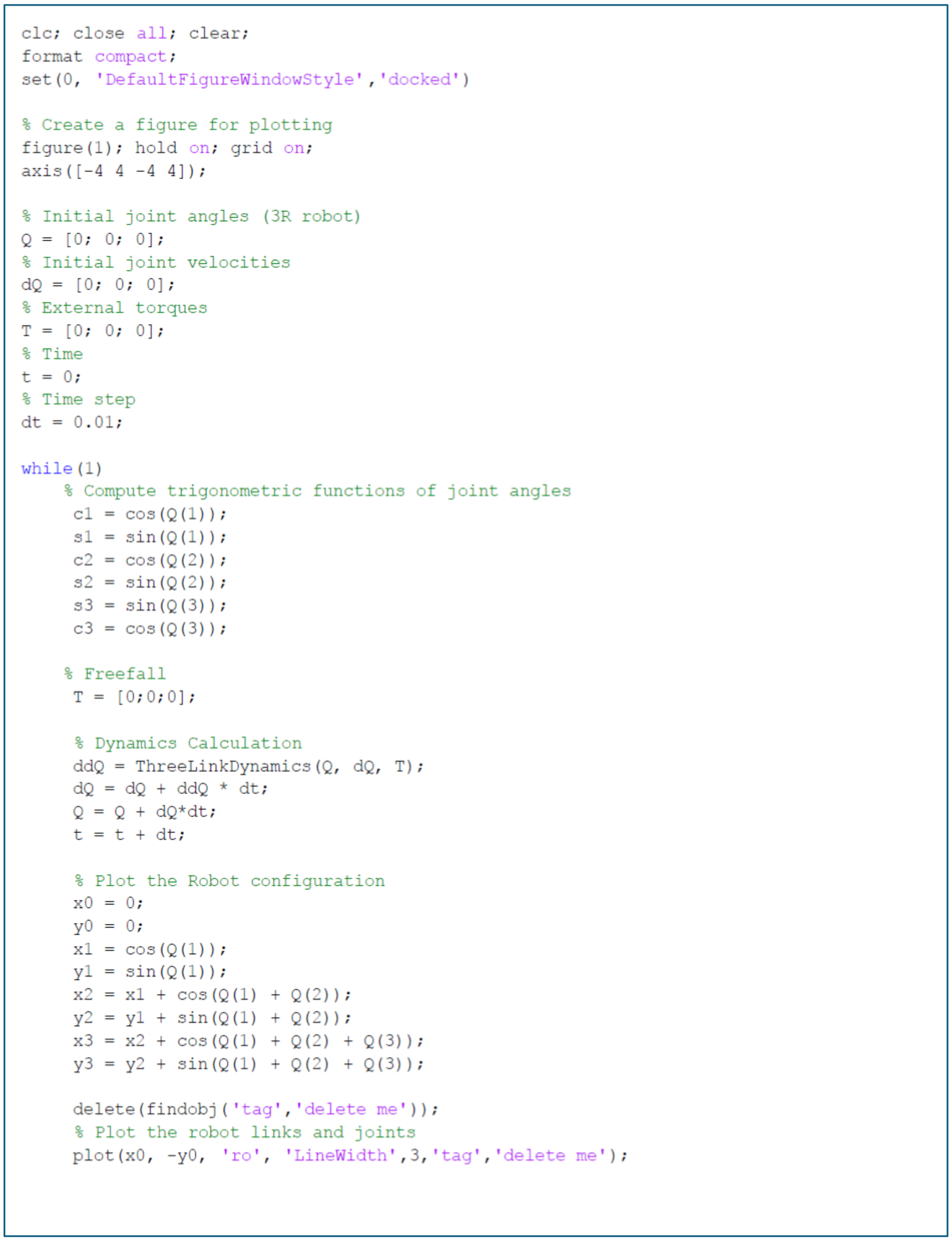
Click the thumbnail to play full video.

Here the open chain is let to free fall and it does act weird after some time given that the joint friction is zero. We can add the joint friction by editing the Torque vector setting it to -dQ/dt.

T = -dQ/dt ← This line will give us joint friction and we can see the output accordingly.

Below is the source code for the same.

Source code:



```matlab
clc; close all; clear;
format compact;
set(0, 'DefaultFigureWindowStyle','docked')

% Create a figure for plotting
figure(1); hold on; grid on;
axis([-4 4 -4 4]);

% Initial joint angles (3R robot)
Q = [0; 0; 0];
% Initial joint velocities
dQ = [0; 0; 0];
% External torques
T = [0; 0; 0];
% Time
t = 0;
% Time step
dt = 0.01;

while(1)
    % Compute trigonometric functions of joint angles
     c1 = cos(Q(1));
     s1 = sin(Q(1));
     c2 = cos(Q(2));
     s2 = sin(Q(2));
     s3 = sin(Q(3));
     c3 = cos(Q(3));

    % Freefall
     T = [0;0;0];

     % Dynamics Calculation
     ddQ = ThreeLinkDynamics(Q, dQ, T);
     dQ = dQ + ddQ * dt;
     Q = Q + dQ*dt;
     t = t + dt;

     % Plot the Robot configuration
     x0 = 0;
     y0 = 0;
     x1 = cos(Q(1));
     y1 = sin(Q(1));
     x2 = x1 + cos(Q(1) + Q(2));
     y2 = y1 + sin(Q(1) + Q(2));
     x3 = x2 + cos(Q(1) + Q(2) + Q(3));
     y3 = y2 + sin(Q(1) + Q(2) + Q(3));

     delete(findobj('tag','delete me'));
     % Plot the robot links and joints
     plot(x0, -y0, 'ro', 'LineWidth',3,'tag','delete me');
     plot(x1, -y1, 'go', 'LineWidth', 3,'tag','delete me');
     plot(x2, -y2, 'mo', 'LineWidth', 3,'tag','delete me');
     plot(x3, -y3, 'bo', 'LineWidth',3, 'tag','delete me');
     plot([x0, x1, x2, x3], -[y0, y1, y2, y3],  'k-', 'LineWidth',1.5,'tag','delete↙
me');

     % Pause and control animation speed
     pause(0.01);
 end

% Function to calculate joint accelerations using forward dynamics
function ddQ  = ThreeLinkDynamics( Q, dQ, T )
    % Extract joint angles and velocities
    q1 = Q(1);
    q2 = Q(2);
    q3 = Q(3);
    dq1 = dQ(1);
    dq2 = dQ(2);
    dq3 = dQ(3);

    % Trigonometric functions of joint angles
    c1 = cos(q1);
    s1 = sin(q1);
    s2 = sin(q2);
    c2 = cos(q2);
    c3 = cos(q3);
    s3 = sin(q3);

    % Gravity acceleration
    g = 9.8;

    % Mass matrix M
    M = [[4, cos(q1-q2), cos(q1-q3)];  ...
        [cos(q2-q1), 2, cos(q2-q3)];  ...
        [cos(q1-q3), cos(q2-q3), 2]];

    % Coriolis and centrifugal matrix C
    C = [-1; -1; -1]*(sin(q1-q2)*dq1*dq2 + sin(q1-q3)*dq1*dq3 + sin(q2-q1)*(dq1-dq2) ↙
*dq2 + sin(q3-q1)*(dq1-dq3)*dq3);...
        sin(q1-q2)*dq1*dq2 + sin(q2-q3)*dq2*dq3 + sin(q2-q1)*(dq1-dq2)*dq1 + sin(q3-q2) ↙
*(dq2-dq3)*dq3;...
        sin(q2-q3)*dq2*dq3 - sin(q1-q3)*dq1*dq3 + sin(q3-q2)*(dq2-dq3)*dq3 + cos(q2-q3) ↙
*dq2;

    % Gravity vector G
    G = [4*g*c1; 2*g*c2; g*c3];

    % Compute joint accelerations using the dynamic equation: M * ddQ + C + G = T
    ddQ = inv(M) * ( T + C + G );
end
```

Appendix

- Source code can be found here → freefall_3R_project.m

Q	An n-vector of reference joint variables thetad
dq	An n-vector of reference joint velocities theta_dot_d
T	Ti is the generalized force applied at joint i, and T is the list of all joint forces/torques
t	Time
dt	A timestep (e.g., between consecutive rows in a matrix representing a trajectory or force history).
ddQ	An n-vector of reference joint accelerations theta_double_dot_d.
ThreeLinkDynamics	A Function that calculates the forward dynamics of the robot taking Q,dq and T as parameters and returning the joint accelerations calculating Mass matrix, Coriolis forces and G-vector
M	The numerical inertia matrix M(theta) of an n-joint serial chain at the given conguration theta.
C	The vector c(theta; theta_dot) of Coriolis and centripetal terms for a given theta and theta_dot.
G	The joint forces/torques required to balance gravity at theta.

- Video output → https://youtu.be/GQ3aVmLVIvI
- Hand Calculations can be found below
  https://docs.google.com/document/d/1Yo_1AiE7e2LKCLpM8BSGTJLaRnE8JaD7fvNlMPywsCM/edit?usp=sharing