Simulating a 2-Joint Robot Arm Using Equations of Motion

Adam Kehl, Angelica Cabusi, Jacob Kim, and Jose Banuelos

Professor Johnny Jingze Li

Abstract—We model a simplified two-link robotic arm using inverse kinematics and inverse dynamics with the goal of following a given motion trajectory. We utilize a simplified inverse kinematics solver and derive all necessary equations starting from the Lagrangian. Lastly, we simulate the system using forward dynamics.

Keywords—*n-link, inverse dynamics, kinematics, torque, simulation*

1. Introduction / Question

Can we calculate the torques necessary to drive a robotic manipulator so that its end-effector follows a predetermined motion path? Then, can we simulate the physical system after generating the torques to verify it's stability?

Understanding robotic motion and control systems is crucial for engineers working with all manner of robotic systems in manufacturing, bio-tech, military, etc. We chose to explain and model the equations of motion/force applied at the end of a robotic manipulator for a simple 2D, 2-link arm.

First, we want to understand and derive basic kinematics and utilize standard energy laws in our calculations. To understand how these interplay, Northwestern Robotics offers a series on YouTube that provides a walk-through of modern control systems and their relevant mathematics/physics derivations:

https://www.youtube.com/playlist?list=PLggLP4f-rq00TQamz2pXjzPWpuxhVN_Vy

By having a strong understanding of the underlying physics, kinematics, peak loads, and control systems, engineers can create more advanced machinery and validate their designs before manufacturing.

2. Methodology / Model

In this project, we create a simulation of a robotic arm and program it to follow an arbitrary motion path.

2.1. Description / Assumptions

We assume the masses are known, friction in the joints is negligible, air drag is negligible, the joints are perfect motors (no torque curves), and that the linkage mass is evenly distributed.

2.2. Development

1. First we **solve for forward kinematics** using homogeneous transforms matrices: given a configuration of joint angles (θ_i) and lengths (l_i), we find the coordinate of the end-effector.
2. Then we **derive inverse kinematics** from the forward kinematics: given a target coordinate, find what joint angle configuration places the end-effector there.
3. We then **derive the potential and kinetic energy** of the system for use in the Lagrangian equation of motion.
4. We **populate the Euler-Lagrange equation** of work/force at the end-effector, using the partial derivatives of the Lagrangian.
5. We use IK and create a **time-step estimation** of desired velocities and acceleration and input these into our ID model and generate the torques.
6. Finally, we **simulate using forward dynamics** the behavior of the robotic arm given only the torques over time and compare the end-effector to the desired trajectory.

2.3. Joint-Frame Translation Matrices

2.3.1. Forward Kinematics

Forward kinematics is used to take the robot parameters (angles) and produce a position and orientation of the end-effector. In our case, we simplify the equation a bit since we consider the end effector as a point instead of an end-effector with complex geometry.

To go from the fixed frame F at the origin to the final frame M_3 which represents the end-effector, we want to calculate

$$F \to M_1 \to M_2 \to M_3$$

Thus, we want to solve for the following homogeneous transforms:

$$\begin{aligned} H &= F \to M_1 \to M_2 \to M_3 \\ &= (F \to M_1)(M_1 \to M_2)(M_2 \to M_3) \\ &= H_1 H_2 H_3 \end{aligned}$$

Thus, $H = F \to M_3$.

In the top left of the 3×3 homogeneous transform matrix, we place the 2×2 rotation matrix and in the last column, we include the displacement (within that frame):

$$\begin{bmatrix} \cos\theta & -\sin\theta & disp_x \\ \sin\theta & \cos\theta & disp_y \\ 0 & 0 & 1 \end{bmatrix}$$

2.3.2. Forward Kinematics Example

Let us calculate the forward kinematics for the example shown:

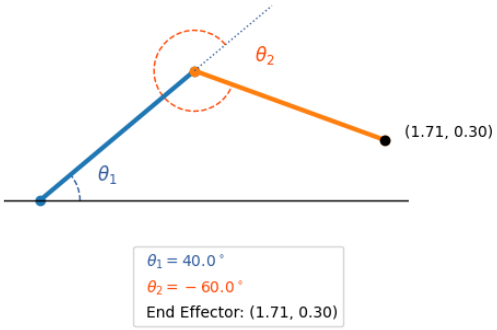

Figure 1. Given the angles θ_1, θ_2, we find the position of the end-effector.

$$\begin{aligned} H &= H_1 H_2 H_3 \\ &= \begin{bmatrix} \cos(40°) & -\sin(40°) & 0 \\ \sin(40°) & \cos(40°) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(-60°) & -\sin(-60°) & 1 \\ \sin(-60°) & \cos(-60°) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 0.7660 & -0.6428 & 0 \\ 0.6428 & 0.7660 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.5 & 0.8660 & 1 \\ -0.8660 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 0.9397 & 0.342 & 1.7057 \\ -0.342 & 0.9397 & 0.3008 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

The top left 2×2 matrix is the rotation of the end-effector:

$$\begin{bmatrix} 0.9397 & 0.342 & 1.7057 \\ -0.342 & 0.9397 & 0.3008 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & disp_x \\ \sin\theta & \cos\theta & disp_y \\ 0 & 0 & 1 \end{bmatrix}$$

Since $\cos\theta = 0.9397$ and $\sin\theta = -0.342$, then:

$$\begin{aligned}\theta &= \tan^{-1}\left(\frac{-0.342}{0.9397}\right)\\ &= \tan^{-1}(-0.364)\\ &\approx -20°\end{aligned}$$

The last column is the total displacement from the origin frame to the end-effector and thus gives the end-effector's position and rotation is:

$$(1.71, 0.30, -20°)$$

2.4. Inverse Kinematics (IK)

Calculating inverse kinematics is the process of calculating the intermediate homogeneous transform matrices given H:

$$\begin{aligned}H &= H_1H_2H_3\\ &= \begin{bmatrix}\cos(\theta_1) & -\sin(\theta_1) & 0\\ \sin(\theta_1) & \cos(\theta_1) & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\cos(\theta_2) & -\sin(\theta_2) & 1\\ \sin(\theta_2) & \cos(\theta_2) & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 1\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}\\ &= \begin{bmatrix}\cos(\theta_1) & -\sin(\theta_1) & 0\\ \sin(\theta_1) & \cos(\theta_1) & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\cos(\theta_2) & -\sin(\theta_2) & \cos(\theta_2)+1\\ \sin(\theta_2) & \cos(\theta_2) & \sin(\theta_2)\\ 0 & 0 & 1\end{bmatrix}\\ &= \begin{bmatrix}\cos\theta_1\cos\theta_2 - \sin\theta_1\sin\theta_2 & -(\cos\theta_1\sin\theta_2 + \sin\theta_1\cos\theta_2) & \cos\theta_1(\cos\theta_2+1) - \sin\theta_1\sin\theta_2\\ \sin\theta_1\cos\theta_2 + \cos\theta_1\sin\theta_2 & \cos\theta_1\cos\theta_2 - \sin\theta_1\sin\theta_2 & \sin\theta_1(\cos\theta_2+1) + \cos\theta_1\sin\theta_2\\ 0 & 0 & 1\end{bmatrix}\\ &= \begin{bmatrix}\cos(\theta_1+\theta_2) & -\sin(\theta_1+\theta_2) & \cos(\theta_1+\theta_2)+\cos\theta_1\\ \sin(\theta_1+\theta_2) & \cos(\theta_1+\theta_2) & \sin(\theta_1+\theta_2)+\sin\theta_1\\ 0 & 0 & 1\end{bmatrix}\end{aligned}$$

Let our desired end-effector pose be represented as the matrix :

$$H = \begin{bmatrix}\cos(\theta_{end}) & -\sin(\theta_{end}) & x_{end}\\ \sin(\theta_{end}) & \cos(\theta_{end}) & y_{end}\\ 0 & 0 & 1\end{bmatrix}$$

We can then create a system of equations and solve for the angles of the robotic arm. However, we're going to simplify our pose by disregarding orientation of end-effector and simply solving for the displacement. Thus yielding the following system:

$$\begin{aligned}x_{end} &= \cos(\theta_1) + \cos(\theta_1+\theta_2),\\ y_{end} &= \sin(\theta_1) + \sin(\theta_1+\theta_2)\end{aligned}$$

First, to simplify further algebra and decouple the variables, we introduce the distance from the origin to the end-effector:

$$\begin{aligned}r^2 &:= (x_{end})^2 + (y_{end})^2\\ &= (\cos(\theta_1)+\cos(\theta_1+\theta_2))^2 + (\sin(\theta_1)+\sin(\theta_1+\theta_2))^2\\ &= \cos^2(\theta_1) + \cos^2(\theta_1+\theta_2) + 2\cos(\theta_1)\cos(\theta_1+\theta_2)\\ &\quad + \sin^2(\theta_1) + \sin^2(\theta_1+\theta_2) + 2\sin(\theta_1)\sin(\theta_1+\theta_2)\\ &= \left[\cos^2(\theta_1)+\sin^2(\theta_1)\right] + \left[\cos^2(\theta_1+\theta_2)+\sin^2(\theta_1+\theta_2)\right]\\ &\quad + 2\cos(\theta_1)\cos(\theta_1+\theta_2) + 2\sin(\theta_1)\sin(\theta_1+\theta_2)\\ &= 1 + 1 + 2\left[\cos(\theta_1)\cos(\theta_1+\theta_2) + \sin(\theta_1)\sin(\theta_1+\theta_2)\right]\\ &= 2 + 2\cos((\theta_1+\theta_2)-\theta_1)(\cos(A-B) = \cos A\cos B + \sin A\sin B)\\ r^2 &= 2 + 2\cos(\theta_2)\end{aligned}$$

To solve for θ_2, we can now do:

$$\begin{aligned}r^2 &= 2 + 2\cos(\theta_2)\\ \cos(\theta_2) &= \frac{r^2-2}{2}\\ \theta_2 &= \pm\arccos\left(\frac{r^2-2}{2}\right)\\ \theta_2 &= \pm\arccos\left(\frac{(x_{end})^2+(y_{end})^2-2}{2}\right)\end{aligned}$$

where the $\pm$ represents elbow up or elbow down configurations.

To solve for θ_1, we use the sum to product formula to derive:

$$\begin{aligned}x_{end} &= \cos(\theta_1) + \cos(\theta_1+\theta_2)\\ &= 2\cdot\cos(\theta_1+\tfrac{\theta_2}{2})\cdot\cos(\tfrac{\theta_2}{2})\\ y_{end} &= \sin(\theta_1) + \sin(\theta_1+\theta_2)\\ &= 2\cdot\sin(\theta_1+\tfrac{\theta_2}{2})\cdot\cos(\tfrac{\theta_2}{2})\end{aligned}$$

We can then form the following since the $\cos(\frac{\theta_2}{2})$ and 2 cancel:

$$\begin{aligned}\tan(\theta_1+\tfrac{\theta_2}{2}) &= \frac{y_{end}}{x_{end}}\\ \theta_1+\tfrac{\theta_2}{2} &= \text{atan2}(y_{end}, x_{end})\\ \theta_1 &= \text{atan2}(y_{end}, x_{end}) - \tfrac{\theta_2}{2}\end{aligned}$$

2.4.1. Inverse Kinematics Example

Recall forward kinematics example end-effector coordinate:

$$(1.71, 0.30)$$

Let us calculate the joint configurations for the end-effector to reach this coordinate (disregarding end-effector angle for simplicity):

$$\begin{aligned}\theta_2 &= \pm\arccos\left(\frac{(x_{end})^2+(y_{end})^2-2}{2}\right)\\ &= \pm\arccos\left(\frac{(1.71)^2+(0.3)^2-2}{2}\right)\\ &= \pm\arccos(0.50705)\\ &\approx \{-59.533, 59.533\}\end{aligned}$$

Let's use the positive angle first:

$$\begin{aligned}\theta_1 &= \text{atan2}(y_{end}, x_{end}) - \tfrac{\theta_2}{2}\\ &= \text{atan2}(0.3, 1.71) - \tfrac{1.03904686}{2}\\ &\approx 0.174 - 0.51952343\\ &\approx= -0.346 = -19.82°\end{aligned}$$

then solve using the negative angle:

$$\begin{aligned}\theta_1 &= \text{atan2}(y_{end}, x_{end}) - \tfrac{\theta_2}{2}\\ &= \text{atan2}(0.3, 1.71) - \tfrac{-1.03904686}{2}\\ &\approx 0.174 + 0.51952343\\ &\approx= 0.69352343 = 39.74°\end{aligned}$$

Thus, we obtained two valid configuration solutions, one that represents "elbow up" and the other for "elbow down":

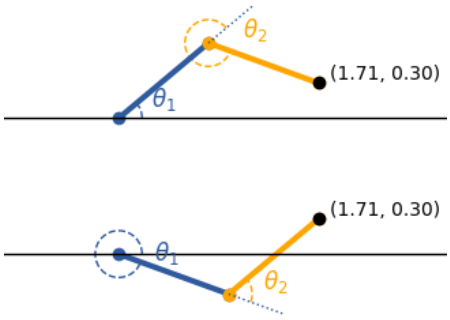

Figure 2. Two solutions to the inverse kinematics problem.

2.4.2. Inverse Kinematics Solvers

Putting IK to use requires us to avoid several caveats such as singularities or switching IK "branches". For example, we can induce a singularity by traversing a shape like:

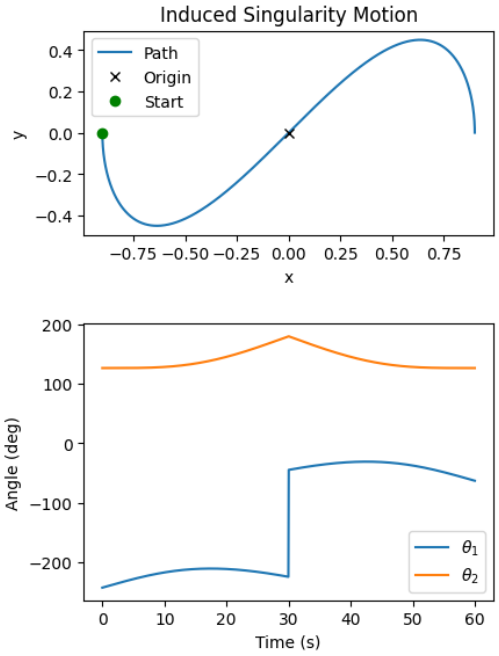

Figure 3. Induced singularity

We can avoid these, if in our inverse kinematics function we incorporate a simple "angle-distance" check to our last position, allowing for circular overflow. See the Code section for our IK solver.

2.5. Inverse Dynamics

2.5.1. Lagrangian

At a high-level, Newtonian mechanics breaks mechanical problems down into forces while Lagrangian methods focuses on energy as its fundamental unit of measure.

The definition of the Lagrangian for a dynamic system is as follows:

$$L = K - P$$
$$L = (\text{kinetic energy}) - (\text{potential energy})$$

Let us represent the angles, angular velocities, and angular accelerations as vector functions:

$$\theta(t) = \begin{bmatrix}\theta_1\\ \theta_2\end{bmatrix}(t), \quad \dot{\theta}(t) = \begin{bmatrix}\dot{\theta_1}\\ \dot{\theta_2}\end{bmatrix}(t), \quad \ddot{\theta}(t) = \begin{bmatrix}\ddot{\theta_1}\\ \ddot{\theta_2}\end{bmatrix}(t)$$

Thus, the kinetic energy depends on the pose and the joints' velocities while the potential energy (due to gravity) only depends on the pose. And therefore, the Lagrangian is simply a function of the pose and angles' velocities as well:

$$L(\theta, \dot{\theta}) = K(\theta, \dot{\theta}) - P(\theta)$$

To solve for the Lagrangian, we start filling out the equation with known quantities.

2.5.2. Kinetic Energy

To solve for the kinetic energy in the system, we sum the translational and rotational energy for each of the linkage masses (assuming the joints have negligible mass).

2.5.3. Kinematics–Translational Kinetic Energy

Let us find the center of mass (COM) as a function of the angles, then take the derivative to get the velocity of each linkage in Cartesian space as a function of the angles and angles' velocities.
Let $(x_i,\ y_i)$ represent the Cartesian location of the center of mass of the i-th linkage. Using trigonometry/geometry we find that the centers of mass are:

$$x_1 = \frac{l_1}{2}\cos(\theta_1), \quad y_1 = \frac{l_1}{2}\sin(\theta_1)$$

and

$$x_2 = l_1\cos(\theta_1) + \frac{l_2}{2}\cos(\theta_1+\theta_2), \quad y_2 = l_1\sin(\theta_1) + \frac{l_2}{2}\sin(\theta_1+\theta_2)$$

for links 1 and 2 respectively.

Now that we have the positions of each linkages' center of mass as a function of the angles, we can differentiate with respect to time to get the velocities of each joint.

Since we want to differentiate a function of the form $f(\theta, \theta_2)$ with respect to time, we use the chain rule to expand it:

$$\frac{d}{dt}f(\theta,\theta_2) = \frac{\partial f}{\partial\theta_1}\dot{\theta}_1 + \frac{\partial f}{\partial\theta_2}\dot{\theta}_2$$

Thus, the velocities of each center of mass become:

$$\dot{x_1} = -\frac{l_1}{2}\sin(\theta_1)\cdot\dot{\theta}_1$$
$$\dot{y_1} = \frac{l_1}{2}\cos(\theta_1)\cdot\dot{\theta}_1$$

and

$$\dot{x_2} = -l_1\sin(\theta_1)\cdot\dot{\theta}_1 - \frac{l_2}{2}\sin(\theta_1+\theta_2)\cdot(\dot{\theta}_1+\dot{\theta}_2)$$
$$\dot{y_2} = l_1\cos(\theta_1)\cdot\dot{\theta}_1 + \frac{l_2}{2}\cos(\theta_1+\theta_2)\cdot(\dot{\theta}_1+\dot{\theta}_2)$$

Recall the equation for the kinetic energy of a moving object:

$$KE = \frac{1}{2}mv^2$$

To obtain the velocity-squared component for the first link (v_1) we use the Pythagorean Theorem:

$$\begin{aligned} v_1^2 &= \dot{x_1}^2 + \dot{y_1}^2 \\ &= \left(-\frac{l_1}{2}\sin(\theta_1)\cdot\dot{\theta}_1\right)^2 + \left(\frac{l_1}{2}\cos(\theta_1)\cdot\dot{\theta}_1\right)^2 \\ &= \frac{l_1^2}{4}\left(\sin^2(\theta_1)+\cos^2(\theta_1)\right)\cdot\dot{\theta_1}^2 \\ &= \frac{l_1^2}{4}\cdot\dot{\theta_1}^2 \end{aligned}$$

Then, for the second link:

$$\begin{aligned} v_2^2 &= \dot{x_2}^2 + \dot{y_2}^2 \\ &= \left(-l_1\sin(\theta_1)\cdot\dot{\theta}_1 - \tfrac{l_2}{2}\sin(\theta_1+\theta_2)\cdot(\dot{\theta}_1+\dot{\theta}_2)\right)^2 \\ &\quad + \left(l_1\cos(\theta_1)\cdot\dot{\theta}_1 + \frac{l_2}{2}\cos(\theta_1+\theta_2)\cdot(\dot{\theta}_1+\dot{\theta}_2)\right)^2 \\ &= l_1^2\dot{\theta_1}^2 + \tfrac{l_2^2}{4}(\dot{\theta}_1+\dot{\theta}_2)^2 \\ &\quad + 2\left(-l_1\sin(\theta_1)\cdot\dot{\theta}_1\right)\cdot\left(-\tfrac{l_2}{2}\sin(\theta_1+\theta_2)\cdot(\dot{\theta}_1+\dot{\theta}_2)\right) \\ &\quad + 2\left(l_1\cos(\theta_1)\cdot\dot{\theta}_1\right)\cdot\left(\tfrac{l_2}{2}\cos(\theta_1+\theta_2)\cdot(\dot{\theta}_1+\dot{\theta}_2)\right) \\ &= l_1^2\dot{\theta_1}^2 + \frac{l_2^2}{4}(\dot{\theta}_1+\dot{\theta}_2)^2 + l_1l_2\dot{\theta}_1(\dot{\theta}_1+\dot{\theta}_2)\cdot\cos(\theta_2) \end{aligned}$$

2.5.4. Inertia–Rotational Kinetic Energy

Now we need the rotational kinetic energy of each linkage. We know the rotational inertia of a uniform rod about it's center of mass is:

$$I_i = \frac{1}{12} m_i {l_i}^2$$

Recall also that the rotational energy is:

$$K_{rot_i} = \frac{1}{2} I_i {\omega_i}^2$$

where the angular speeds are simply: $\omega_1 = \dot{\theta_1}$ and $\omega_2 = \dot{\theta_1} + \dot{\theta_2}$.

Therefore, the kinetic energy of the linkages is the sum of their translational and rotational energies:

$$K(\theta, \dot{\theta}) = \left[\tfrac{1}{2} m_1 {v_1}^2 + \tfrac{1}{2} m_2 {v_2}^2\right] + \left[\tfrac{1}{2} I_1 {\dot{\theta_1}}^2 + \tfrac{1}{2} I_2 (\dot{\theta_1} + \dot{\theta_2})^2\right]$$

2.6. Potential Energy

The potential energy in our system is gravitational potential energy:

$$PE = mgh$$

Thus,

$$\begin{aligned} P(\theta) &= m_1 y_1 g + m_2 y_2 g, \quad g \approx 9.81\ m/s^2 \\ &= m_1 g \frac{l_1}{2} \sin(\theta_1) + m_2 g (l_1 \sin(\theta_1) + \frac{l_2}{2} \sin(\theta_1 + \theta_2)) \end{aligned}$$

2.7. Euler-Lagrangian

The Euler-Lagrangian equation derivation is beyond the scope of this class. It comes from a variational principle, e.g. Hamilton's principle of stationary action or D'Alembert's principle of virtual work. In our case, the generalized force conjugate to θ_i is the torque τ_i:

$$\tau_i = \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_i}\right) - \frac{\partial L}{\partial \theta_i}$$

Let's start by computing the partial derivative of the Lagrangian with respect to the velocity $\dot{\theta}_i$. We observe that the potential energy does not depend on velocity, so we can simplify our calculations to be just:

$$\frac{\partial L}{\partial \dot{\theta}_i} = \frac{\partial K}{\partial \dot{\theta}_i}$$

By utilizing the concept of a manipulator inertia matrix (or "mass matrix"), we can group/rewrite the kinetic energy into:

$$\begin{aligned} K\left(\begin{bmatrix}\theta_1 \\ \theta_2\end{bmatrix}, \begin{bmatrix}\dot{\theta_1} \\ \dot{\theta_2}\end{bmatrix}\right) &= \tfrac{1}{2} \begin{bmatrix}\dot{\theta_1} & \dot{\theta_2}\end{bmatrix} \begin{bmatrix}M_{11} & M_{21} \\ M_{12} & M_{22}\end{bmatrix} \begin{bmatrix}\dot{\theta_1} \\ \dot{\theta_2}\end{bmatrix} \\ &(M_{12} = M_{21}) \text{ due to symmetry} \\ &= \tfrac{1}{2} M_{11} {\dot{\theta_1}}^2 + M_{12} \dot{\theta_1} \dot{\theta_2} + \tfrac{1}{2} M_{22} {\dot{\theta_2}}^2 \end{aligned}$$

Where the manipulator inertia matrix is defined as:

$$M(\theta) = \begin{bmatrix} I_1 + I_2 + \frac{m_1 {l_1}^2}{4} + m_2({l_1}^2 + \frac{{l_2}^2}{4} + l_1 l_2 \cos(\theta_2)) & I_2 + m_2(\frac{{l_2}^2}{4} + \frac{l_1 l_2}{2} \cos(\theta_2)) \\ I_2 + m_2(\frac{{l_2}^2}{4} + \frac{l_1 l_2}{2} \cos(\theta_2)) & I_2 + \frac{m_2 {l_2}^2}{4} \end{bmatrix}$$

Thus, explicitly we have:

$$\frac{\partial K}{\partial \dot{\theta_1}} = M_{11}\dot{\theta_1} + M_{12}(\dot{\theta_1} + \dot{\theta_2}), \quad \frac{\partial K}{\partial \dot{\theta_2}} = M_{12}\dot{\theta_1} + M_{22}(\dot{\theta_1} + \dot{\theta_2})$$

or we can write simply:

$$\frac{\partial K}{\partial \dot{\theta}} = \frac{\partial L}{\partial \dot{\theta}} = M(\begin{bmatrix}\theta_1 \\ \theta_2\end{bmatrix}) \begin{bmatrix}\dot{\theta_1} \\ \dot{\theta_2}\end{bmatrix} = M(\theta)\dot{\theta}$$

Then, to take the derivative with respect to time, we get:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_i}\right) = M(\theta)\ddot{\theta} + C(\theta, \dot{\theta})\dot{\theta}$$

where the term $C(\theta, \dot{\theta}) \cdot \dot{\theta}$ (or Coriolis–a velocity product term) satisfies the following:

$$C(\theta, \dot{\theta}) \cdot \dot{\theta} = \begin{bmatrix} -m_2 l_1 \frac{l_2}{2} \sin(\theta_1) \dot{\theta_2} (\dot{\theta_1} + \dot{\theta_2}) \\ m_2 l_1 \frac{l_2}{2} \sin(\theta_2) {\dot{\theta_1}}^2 \end{bmatrix}$$

Lastly, we compute the partial derivative of the Lagrangian with respect to the angles $\frac{\partial L}{\partial \theta_i}$:

$$\frac{\partial L}{\partial \theta_i} = \frac{\partial K}{\partial \theta_i} - \frac{\partial P}{\partial \theta_i}$$

Since we already solved for the potential energy with:

$$P = m_1 g \frac{l_1}{2} \sin(\theta_1) + m_2 g (l_1 \sin P(\theta_1) + \frac{l_2}{2} \sin(\theta_1 + \theta_2))$$

Taking the partial derivative with respect to each angle, we obtain:

$$\begin{aligned} \frac{\partial P}{\partial \theta_1} &= m_1 g \frac{l_1}{2} \cos(\theta_1) + m_2 g (l_1 \cos(\theta_1) + \frac{l_2}{2} \cos(\theta_1 + \theta_2)), \\ \frac{\partial P}{\partial \theta_2} &= m_2 g \frac{l_2}{2} \cos(\theta_1 + \theta_2) \end{aligned}$$

Then, for the kinetic energy term, we also find the partial derivative with respect to each angle:

$$\frac{\partial K}{\partial \theta_1} = 0$$

Intuitively, this makes sense. The orientation of the first angle won't change the overall shape of the robotic arm in a way that influences the kinetic energy, only the second joint will.

And the second partial derivative is:

$$\frac{\partial K}{\partial \theta_2} = -\frac{m_2 l_1 l_2}{2} \sin(\theta_2) \dot{\theta_1} (\dot{\theta_1} + \dot{\theta_2}) - m_2 \frac{l_2}{2} g \cos(\theta_1 + \theta_2)$$

Thus, combining both of these terms yields:

$$\begin{aligned} \frac{\partial L}{\partial \theta_1} = \frac{\partial K}{\partial \theta_1} - \frac{\partial P}{\partial \theta_1} &= \left(0\right) - \left(m_1 g \frac{l_1}{2} \cos(\theta_1) + m_2 g (l_1 \cos(\theta_1) + \frac{l_2}{2} \cos(\theta_1 + \theta_2))\right) \\ &= -(m_1 \frac{l_1}{2} + m_2 l_1) g \cos(\theta_1) - m_2 \frac{l_2}{2} g (\cos(\theta_1 + \theta_2), \\ \frac{\partial L}{\partial \theta_2} = \frac{\partial K}{\partial \theta_2} - \frac{\partial P}{\partial \theta_2} &= \left(-\frac{m_2 l_1 l_2}{2} \sin(\theta_2) \dot{\theta_1} (\dot{\theta_1} + \dot{\theta_2})\right) - \left(m_2 g \frac{l_2}{2} \cos(\theta_1 + \theta_2)\right) \end{aligned}$$

From this step, we separate out the components with a gravity term and define it as $G(\theta)$:

$$G(\theta) = \begin{bmatrix} (m_1 \frac{l_1}{2} + m_2 l_1) g \cos(\theta_1) + m_2 \frac{l_2}{2} g (\cos(\theta_1 + \theta_2) \\ m_2 \frac{l_2}{2} g \cos(\theta_1 + \theta_2) \end{bmatrix}$$

The "missing" component from the previous term is factored into the Coriolis matrix.

Combine all the pieces and we are left with the following:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_i}\right) - \frac{\partial L}{\partial \theta_i} = M(\theta)\ddot{\theta} + C(\theta, \dot{\theta})\dot{\theta} + G(\theta) = \tau$$

$M(\theta)\ddot{\theta}$: The torque required to accelerate the joints given the pose.

$C(\theta, \dot{\theta})\dot{\theta}$: Extra torque needed due to the object already moving.

$G(\theta)$: Torque to hold the arm up against gravity in the current pose.

3. Simulation / Interpretation

Ultimately we wanted to create a model of a robotic arm that would be able to follow an arbitrary motion path. In our example, we take a looping path and break it up into even time steps (but our method would work for any desired velocity/acceleration along the path).

We defined a circular/looping motion path and split it up evenly over an amount of time. Our goal was to move the end-effector to each point at the calculated time. Thus, we calculated the velocity needed between every two points over their difference in time.

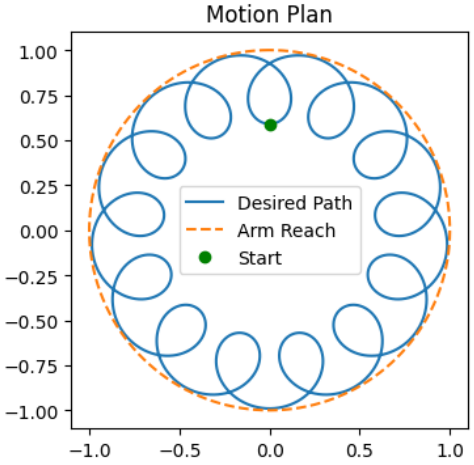

Figure 4. Our desired motion path.

We inputted the desired locations, time, and velocities to reach each subsequent point into our inverse dynamics model, yielding us a torque value between each segment of points.

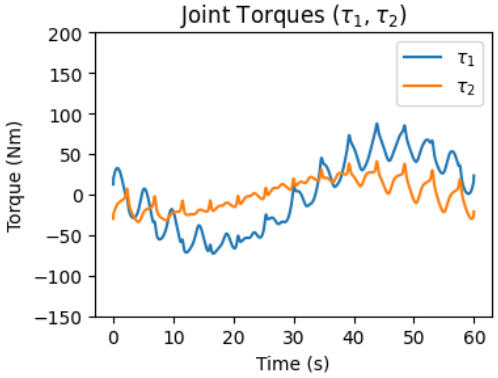

Figure 5. The calculated torques necessary to drive the robotic arm.

As a bonus, we fed the torques back into the forward dynamics model to simulate the robot:

$$\ddot{\theta} = M(\theta)^{-1}(\tau - C(\theta, \dot{\theta})\dot{\theta} - G(\theta))$$

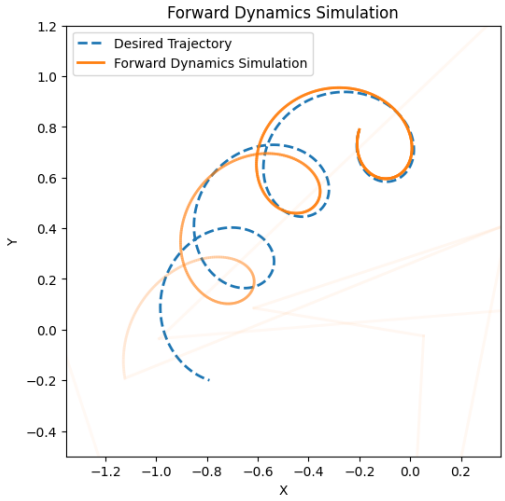

Figure 6. Quickly diverges due to numerical approximation errors.

However, the system diverges very quickly and easily.

3.0.1. Limitations

Our forward dynamics simulation diverges due to a large number of reasons including: numerical approximations, estimations we made over the time step between points, starting velocity discrepancies, etc. In the real world, we introduce an error term in the inverse dynamics model to account for drag, friction of the motor, the motor's dynamics changing due to heat/wear, the measurements not being precisely accurate, etc.

The next logical step would be to (in all modern control systems) is to incorporate proportional/integral/derivative controls to influence damping and rigidity of the motion and correct for small disturbances. All to say, we have only begun to explore the complexities of robotic control systems.

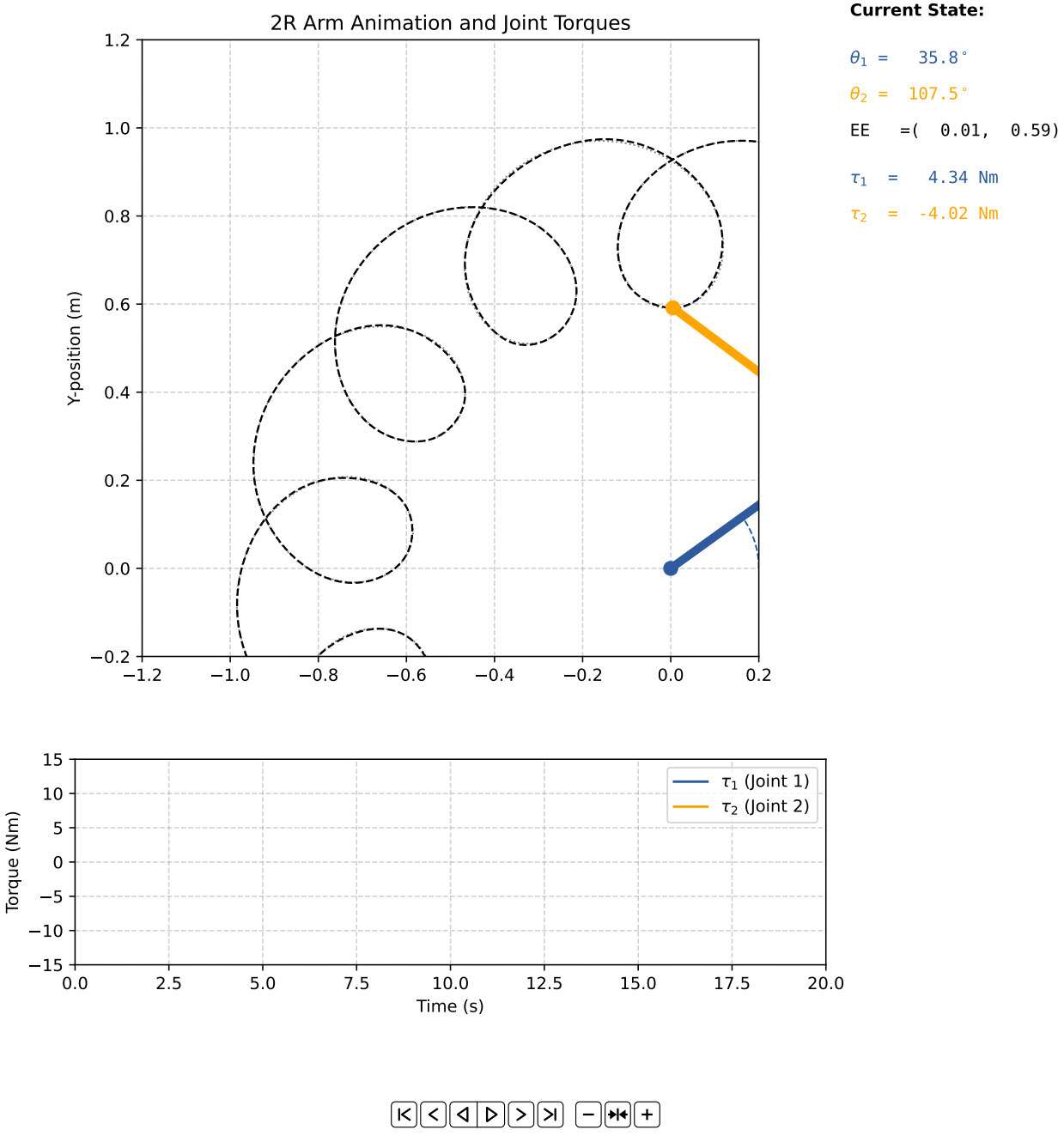

Figure 7. (Animation only works in Firefox or Adobe Acrobat Reader)

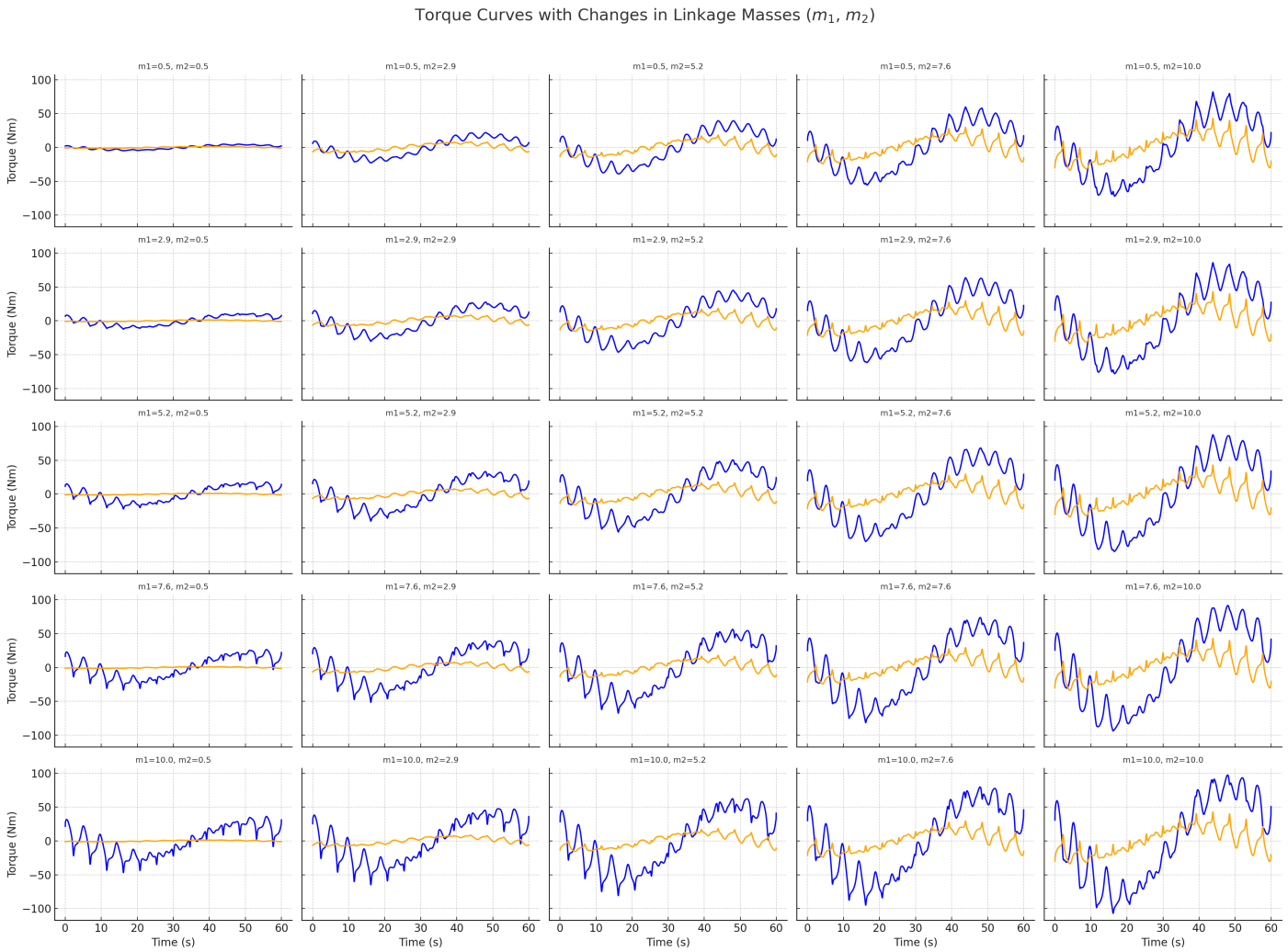

Figure 8. Sweeping over the mass of each linkage and visualizing the change in torque curves over the looping/circular motion path.

4. Code

We include only the code to calculate the inverse dynamics of the robotic arm as well as plot the desired path and output torques.

```python
import numpy as np
import matplotlib.pyplot as plt
scale   = 0.79 # circle radius
radius  = 0.2  # loop radius
freq    = 13   # number of loops around circle
T_total = 60.0 # total motion time (s)
N       = 2000 # number of time steps / path points
t = np.linspace(0, T_total, N) # time vector
# create the looping path along the circle

# generate uniform circular base path
phi = np.linspace(0, 2*np.pi, N) + np.pi/2 # rotate so
    ↪ the starting point is at the top

path_base = scale * np.column_stack((np.cos(phi), np.sin
    ↪ (phi)))

# add loops, evenly sampled along the base path
def tangent_normal_frame(path):
    d = np.gradient(path, axis=0)
    lengths = np.linalg.norm(d, axis=1, keepdims=True)
    lengths[lengths == 0] = 1.0
    Tvec = d / lengths
    Nvec = np.column_stack((-Tvec[:,1], Tvec[:,0]))
    return Tvec, Nvec

def add_loops(path, radius, freq):
    N = len(path)
    u = np.linspace(0, 1, N)
    angles = 2 * np.pi * freq * u + np.pi/2 # fixes loop
    ↪  phase

    Tvec, Nvec = tangent_normal_frame(path)
    offsets = ( radius * np.cos(angles)[:,None] * Tvec +
    ↪  radius * np.sin(angles)[:,None] * Nvec )
    return path + offsets

loopy_path = add_loops(path_base, radius, freq)
```

Code 1. Path generation

```python
def inverse_kinematics_both(x, y, l1, l2):
    r2 = x*x + y*y
    cos2 = np.clip((r2 - l1*l1 - l2*l2) / (2*l1*l2), -1,
    ↪  1)
    th2_up = np.arccos(cos2)
    th2_down = -th2_up
    def th1(th2):
        k1 = l1 + l2*np.cos(th2)
        k2 = l2*np.sin(th2)
        return np.arctan2(y, x) - np.arctan2(k2, k1)
    return (th1(th2_up),   th2_up), (th1(th2_down), th2
    ↪ _down)

def compute_joint_trajectory_both(path, l1, l2):
    N = len(path)
    thetas = np.zeros((N,2))
    # seed with our preferred branch (elbow to the right
    ↪ )
    thetas[0] = inverse_kinematics_both(*path[0], l1, l2
    ↪ )[0]

    for i in range(1, N):
        sol_up, sol_down = inverse_kinematics_both(*path
    ↪ [i], l1, l2)
        # pick the one with smaller wrapped distance
        if dist_cfg(sol_up,   thetas[i-1]) <= dist_cfg(
    ↪ sol_down, thetas[i-1]):
            thetas[i] = sol_up
        else:
            thetas[i] = sol_down
    return thetas

# smallest signed difference between angles
def wrap_delta(a, b):
    d = a - b
    # wrap into [\pi, +\pi)
    return (d + np.pi) % (2*np.pi) - np.pi

# euclidean distance in terms of joints (with wrapping)
def dist_cfg(cfg1, cfg2):
    d1 = wrap_delta(cfg1[0], cfg2[0])
    d2 = wrap_delta(cfg1[1], cfg2[1])
    return np.hypot(d1, d2)

def finite_diff_angles(arr, t):
    dt = t[1] - t[0]
    delta = wrap_delta(arr[1:], arr[:-1])
    diffs = delta / dt
    t_mid = (t[:-1] + t[1:]) / 2
    return t_mid, diffs
```

Code 2. Applied kinematics and basic solver

```python
# robot configuration
params = {
    'l1':0.5, 'l2':0.5,
    'm1':1.0, 'm2':10.0, # top link heavier for greater
    ↪ effect on torque plots
    'I1':(1/12)*1.0**2,
    'I2':(1/12)*10.0**2,
    'g': 9.81
}

thetas = compute_joint_trajectory_both(loopy_path,
    ↪ params['l1'], params['l2'])
t_v, vel = finite_diff_angles(thetas, t)
t_a, accel = finite_diff_angles(vel, t_v)

torques = []
for (th1, th2), (d1, d2), (dd1, dd2), ti in zip(
        thetas[1:-1], vel, accel, t_a):
    M = M_matrix(th2,
                 params['l1'], params['l2'],
                 params['m1'], params['m2'],
                 params['I1'], params['I2'])
    C = C_matrix(th2, d1, d2,
                 params['l1'], params['l2'],
                 params['m2'])
    Gv = G_vector(th1, th2,
                 params['l1'], params['l2'],
                 params['m1'], params['m2'],
                 params['g'])
    tau = M.dot([dd1, dd2]) + C.dot([d1, d2]) + Gv
    torques.append((ti, tau[0], tau[1]))

times, tau1, tau2 = map(np.array, zip(*torques))
```

Code 3. Creating inputs for ID to output torques

```python
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(4, 7),
    ↪ gridspec_kw={'height_ratios': [3, 2]})

# top plot is the looping path and the circular reach of
    ↪  the arm
ax1.plot(loopy_path[:,0], loopy_path[:,1], label="
    ↪ Desired Path")
unit_circ = np.column_stack((np.cos(phi), np.sin(phi)))
ax1.plot(unit_circ[:,0], unit_circ[:,1], '--', label="
    ↪ Arm Reach")
ax1.plot(loopy_path[0,0],  loopy_path[0,1],  'go', label
    ↪ ='Start')

ax1.set_aspect('equal', 'box')
ax1.set_title("Motion Plan")
ax1.legend()

# top plot is torques
ax2.plot(times, tau1, label=r"$\tau_1$")
ax2.plot(times, tau2, label=r"$\tau_2$")
ax2.set_xlabel("Time (s)")
ax2.set_ylabel("Torque (Nm)")
```

```
ax2.set_title(r"Joint Torques ($\tau_1, \tau_2$)")
ax2.set_ylim(-150, 200)
ax2.legend()

plt.tight_layout()
plt.show()
```

Code 4. Generating the plots showing desired path and calculated torques

```
def M_matrix(th2, l1, l2, m1, m2, I1, I2):
    M11 = I1 + I2 + (m1*(l1*l1))/4 + m2*((l1*l1) + (l2*l
     ↪ 2)/4 + l1*l2*np.cos(th2))
    M12 = I2 + (m2*(l2*l2))/4 + ((l1*l2)/2)*np.cos(th2))
    M22 = I2 + (m2*(l2*l2))/4
    return np.array([[M11, M12],
                     [M12, M22]])
```

Code 5. Function for the manipulator inertia matrix (mass matrix)

```
def C_matrix(th2, th1_dot, th2_dot, l1, l2, m2):
    h = -m2 * l1 * (l2/2) * np.sin(th2)
    return np.array([[h*th2_dot, h*(th1_dot+th2_dot)],
                     [-h*th1_dot, 0.0]])
```

Code 6. Function for the Coriolis matrix. Note: it's a slightly different variation from the one we derived since this one is more commonly used in computation by toolkits/libraries/etc.

```
def G_vector(th1, th2, l1, l2, m1, m2, g=9.81):
    g1 = (((m1*l1)/2 + m2*l1)*g*np.cos(th1)
          + m2*(l2/2)*g*np.cos(th1+th2))
    g2 = m2*l2/2*g*np.cos(th1+th2)
    return np.array([g1, g2])
```

Code 7. Function for the gravity vector.

References

[1] engrprogrammer, *Two link robotic manipulator modelling and simulation on matlab*. [Online]. Available: https://youtube.com/shorts/RKZ0zjuxSvs?si=t6uC7lh3tqJkBYve.

[2] N. Robotics, *Modern robotics, chapter 11.2.1: Error response*. [Online]. Available: https://www.youtube.com/watch?v=ddkUszFVTUk&list=PLggLP4f-rq02N54sD6xwdDWlDScvb32Pp&index=2.

[3] N. Robotics, *Modern robotics, chapter 8.1: Lagrangian formulation of dynamics*. [Online]. Available: https://www.youtube.com/watch?v=1U6y_68CjeY.

[4] N. Robotics, *Modern robotics, chapters 9.1 and 9.2: Point-to-point trajectories*. [Online]. Available: https://www.youtube.com/watch?v=0ZqeBEa_MWo&list=PLggLP4f-rq00wo1z_Or2rRPAt1pStwmlY&index=2.

[5] M. Simulink, *Derive and apply inverse kinematics to two-link robot arm*. [Online]. Available: https://www.mathworks.com/help/symbolic/derive-and-apply-inverse-kinematics-to-robot-arm.html.

[6] M. Simulink, *Ltv model of two-link robot*. [Online]. Available: https://www.mathworks.com/help/control/ug/linear-time-varying-robot-model.html.