Analysis of a 2D, 2-arm Robot Manipulator Final Report

Team Members: Kyler Limata, QinFu Hu

Saint Martin's University

Department of Mechanical Engineering

MME 568, Modeling and Simulation

Prepared for Dr. Shawn Duan on November 26th, 2023

Table of Contents

Table of Figures

Table of Tables

Introduction and Objectives

In the evolving landscape of robotics, the design and analysis of robot manipulators stand as a cornerstone in industrial and research applications. This report presents a comprehensive analysis of a 2D two-arm robot manipulator, focusing on its kinematic characteristics. The project, undertaken as part of MME 568 Modeling and Simulation at Saint Martin's University, aims to dissect the complexities of a simplified 2D model of a robot arm, typically a more complex structure in real-world applications.

The project narrows down to a 2D design, encompassing two segments and a gripper, excluding the rotation of the gripper for simplification. This approach allows for a focused analysis on the kinematic behavior of the arm, which is pivotal in understanding its operational dynamics. The model is conceptualized as two linked bars connected at a joint to the base, with the gripper represented by a specific point, emphasizing the motion aspects we intend to study.

Our methodology is grounded in rigorous kinematic analysis, employing tools such as Autolev for simulation and validation. The study involves a detailed examination of linear velocities and accelerations, partial velocities, and the kinematic relationships between various components of the robot arm. Through this analysis, we aim to gain insights into the arm's movement capabilities and limitations, which are crucial for practical applications in automated systems.

System Overview

Robot gripper arms such as those used in real-world industrial environments consist of multiple segments with joints that can each have one or more degrees of freedom, as well as a wrist joint with a gripper or other tool attached mounted on a base that can possibly rotate. Shown below is one such example of an industrial robot arm.

Figure 1 An Industrial Arm Robot with a wielding attachment [1]

These robots of course, move their end – gripper or otherwise – through three-dimensional space by rotating their joints. However, this project will focus on a simpler design that only moves through 2D space. The specific design covered consists of two segments and a gripper on the end, though rotation of the gripper will not be considered for this project as it typically

would rotate about the axis through the final segment, which of course it cannot do in 2D space. A rough sketch of what this robot looks like in 2D is shown below.

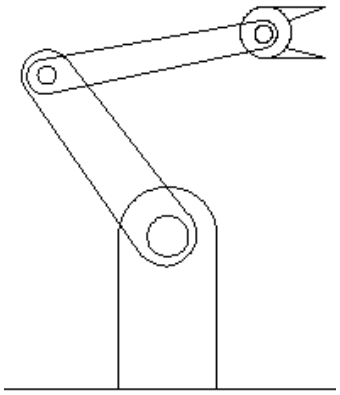

Figure 2 Rough Sketch of 2D robot arm

Significance of Project

Robotics as a field is rapidly becoming more relevant to the manufacturing industry and are arguably already a mainstay. Among the many types of robots used in manufacturing are of course arm-based robots like the one in this project, either with grippers on the end or other tools. Arguably, they are one of the most useful since they can work on a variety of products without being super specialized and several of them can be deployed to work on cooperatively.

Understanding how such robots work, then, is significant to the manufacturing industry, especially since their movement can be very complicated due to the high number of rotational degrees of freedom. Since translation of the arm's gripper or other tool needs to happen by rotating two joints – or more for some models – it is not straightforward to predict how the robot needs to move to get its end from one point to another in space. That of course is not even considering that there are multiple series of rotations that can take that end from point A to point B, but many of them may not be optimal or even possible due to constraints imposed by the geometry of the workspace. While doing such a fully-fledged analysis for an optimal solution to a robot arm's movement through space is not necessarily within the scope of this project, doing a simple, 2D analysis should provide a good basis for understanding just how these sorts of robots work.

Reasons for Selecting Project

The selection of this project stems from a desire to better understand the world of robotics. As discussed in the prior section, they are both very complex and very relevant in today's world. While there are many different types of robots out there, robot arms are among the simpler to analyze since they are fixed in space while still having a dynamic range of motion driven by autonomous algorithms. Through development of the motion analysis model for the 2D robot arm, it is possible to begin understanding how all robots work and develop the general knowledge necessary to model more complex robots.

Literature Review

One paper that delves into the simulation of robotic arms is “Model and simulation of arm robot with 5 degrees of freedom using MATLAB” by M Sabri et al. This paper looks at the simulation of motion for a 3D robot arm which is like our 2D robot arm in that it is made up of two segments with a gripper. M Sabri et al begin by discussing how to break down the robot arm into the generalized coordinates described by the 5 degrees of freedom, with a comparison to the human arm; the robot arm for this project will of course not be as complex, but this still provides useful insight. After that, the paper then goes into detail on modeling the motion of the robot arm; they determine a set path for the gripper to move a ball along and set up the equations of motion using inverse kinematics, including the position and orientation constraints (they do not account for the limitation of join motion range). Since each joint of the robot moves independently, M Sabri et al make use of the Newton-Raphson to find the best configuration of all five degrees-of-freedom, discussing their computations in detail and presenting their results in one graph for each [2]. While the scope of this project only covers a 2D robot arm, their methods for calculating the movement of each degree-of-freedom may still be helpful in this project.

Yizhong Luan et al also cover the simulation of an industrial robot arm in their paper “Structural Dynamics Simulation Analysis of Industrial Robot Arm Based on Kane Method”, with the goal of improving the design process for industrial robots. The paper opens by discussing how the traditional method design method results in robot arms with poor stability and how it will present an improved design method using Kane’s method. A robot arm with specific parameters is then presented, which Luan et al then apply Kane’s principle to create a transformation matrix. In the following section, they use the characteristics of the specific robot to lay out the equations for a topology optimization model of the robot’s motion, which goes into the dynamics side of motion analysis, and wrap up by presenting a process for optimization. Comparing their design method to the traditional method, Luan et al shows that using the Kane method achieves a significant improvement in stability of the arm [3]. While this paper covers more of the dynamics side than the kinematic side, it should nevertheless be helpful to inform the methodology used in this project.

(Mahil & Al-Durra, 2016) presented at the IEEE 59th International Midwest Symposium on Circuits and Systems focused on the modeling, analysis, and simulation of a two-degree-of-freedom (2-DOF) robotic manipulator. This work is pivotal in the field of robotics, especially for applications requiring high precision, such as in automotive, aerospace, and medical fields. The team commenced with a detailed mathematical model of a robotic arm, utilizing the Denavit-Hartenberg convention and the Langrangian equation of motion to thoroughly understand its dynamics and kinematics. The study then advanced into a comprehensive qualitative analysis using MATLAB and Simulink, assessing the system's stability, controllability, and observability. Ultimately, the researchers developed and implemented sophisticated control systems,

including a full state feedback controller, a linear quadratic regulator (LQR), and a full-order observer, significantly enhancing the manipulator's accuracy and precision [4].

(Teng Fong et al., 2015) conducted a comprehensive study on the design and analysis of a Linear Quadratic Regulator (LQR) for a nonlinear rotary inverted pendulum system, underscoring its importance in robust controller design for nonlinear systems. This research is particularly relevant in areas like robotics, aerospace, and marine engineering. It involved the dynamic system modeling of an inverted pendulum, where the mathematical model was linearized at upright equilibrium points. This model was then validated using a nonlinear least square frequency domain identification approach. Focusing on the LQR controller, the study analyzed the effect of the weighting parameter Q on the system's response and optimized the state-feedback control vector K. The research demonstrated that the LQR controller, recognized for its simplicity and robust performance, was more effective than full state feedback control in maintaining the pendulum's upright position, even in the presence of external disturbances [5].

(He et al., 2017) in their paper published in IEEE Transactions on Industrial Informatics, developed a neural network (NN) controller for vibration suppression in a flexible robotic manipulator system with input dead zone. The NN controller approximates the unknown dynamics of the robotic manipulator and mitigates the effects of input dead zone in the actuators. The flexible manipulator model is based on the lumping spring-mass method, and two types of NN controllers are proposed: full state feedback and output feedback with a high-gain observer. These controllers are designed to compensate for the dead zone effect and accurately track the desired trajectory. The effectiveness of the NN controllers is validated through simulations and experiments, highlighting their superiority over the proportional derivative (PD) controller in managing the system's vibrations [6].

(Zhang & Liu, 2012) in their paper "Observer-based partial differential equation boundary control for a flexible two-link manipulator in task space," published in IET Control Theory and Applications, address the trajectory control of a flexible two-link manipulator using a partial differential equation (PDE) dynamic model. The study makes a significant contribution by proposing a novel non-linear PDE observer to estimate distributed positions and velocities along flexible links, a task not possible with typical ordinary differential equation observers. Additionally, they use the singular perturbation approach to decompose the rigidity-flexibility coupling dynamics, aiding the control design process. The researchers developed a boundary control scheme to regulate the end effector's trajectory in task space while suppressing vibration. Theoretical analysis and simulation results confirm the asymptotic stability of both the observer and the control algorithm, underscoring the effectiveness of their approach [7].

Theory and Procedure

Performing a kinematic analysis of the gripper arm required simplification of the model. The arm was simplified into two linked bars, connected to each other at point J – representing the

joint - and to the base at point O. The gripper is represented by point E as its motion is not being considered for this project.

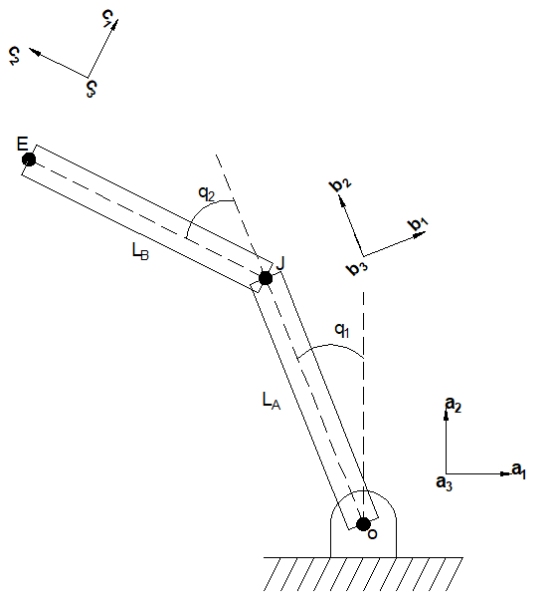

Figure 3 Kinematic Model of Robot Arm

The kinematic model contains three reference frames A (Newtonian), B, and C, which are tied to points O, J, and E, respectively. The generalized coordinates q_1 and q_2 represent the rotation of link B with length L_A and link C with length L_B with respect to points O and J, respectively. Since this is a 2D system, all three reference frames have their third axis pointing out of the page, and therefore in the same direction.

Kinematic Analysis

At this stage, only the linear velocities and accelerations of the middle joint and the end effector with respect to Newtonian reference frame A are desired. This section will go over the manual derivation of both.

Setup

Figure 3 below demonstrates the trigonometric relationship between frames A and B, which can be extrapolated to frames B and C.

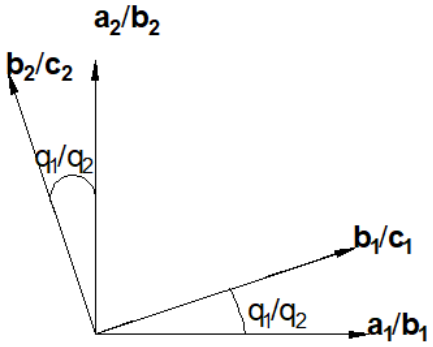

Figure 4 Relationship between frames A and B, B and C

Since the velocities and accelerations of points J and E need to be derived with respect to frame A, this can be used to create the transform matrices in **Tables 1 and 2** can be constructed so that everything can be rewritten in terms of A.

Table 1 Transform Matrix between frames B and A

	$\hat{a}_1$	$\hat{a}_2$	$\hat{a}_3$
$\hat{b}_1$	$\cos(q_1)$	$\sin(q_1)$	0
$\hat{b}_2$	$-\sin(q_1)$	$\cos(q_1)$	0
$\hat{b}_3$	0	0	1

Table 2 Transform Matrix between frames C and B

	$\hat{b}_1$	$\hat{b}_2$	$\hat{b}_3$
$\hat{c}_1$	$\cos(q_2)$	$\sin(q_2)$	0
$\hat{c}_2$	$-\sin(q_2)$	$\cos(q_2)$	0
$\hat{c}_3$	0	0	1

Aside from that, the generalized velocities can be established, along with the angular velocities, angular accelerations, and the distance vectors between the points as shown in **Tables 3 and 4.**

Table 3 Generalized velocities and accelerations.

$\dot{q}_2$	$\dot{q}_2$	$\ddot{q}_1$	$\ddot{q}_2$
u_1	u_2	$\dot{u_1}$	$\dot{u_2}$

Table 4 Angular velocities and distances

${}^{A}\vec{\omega}^{B}$	${}^{A}\vec{\omega}^{C}$	${}^{A}\vec{\alpha}^{B}$	${}^{A}\vec{\alpha}^{C}$	$\vec{r}^{OJ}$	$\vec{r}^{JE}$
$u_1\hat{b}_3$	$(u_1 + u_2)\hat{c}_3$	$\dot{u_1}\hat{b}_3$	$(\dot{u_1} + \dot{u_2})\hat{c}_3$	$L_A\hat{b}_2$	$L_B\hat{c}_2$

Where ${}^{A}\vec{\omega}^{A}$ is the angular velocity of link A, ${}^{A}\vec{\omega}^{B}$ is the angular velocity of link B, ${}^{A}\vec{\alpha}^{A}$ is the linear acceleration of link A, ${}^{A}\vec{\alpha}^{B}$ is the linear acceleration of link B, $\vec{r}^{OJ}$ is the distance vector from point O to point J, and $\vec{r}^{JE}$ is the distance vector from point J to point E.

The unit vectors $\hat{b}_1$, $\hat{b}_2$, $\hat{c}_1$, and $\hat{c}_2$ can be redefined in terms of $\hat{a}_1$ and $\hat{a}_2$ to avoid repeating these calculations several times while deriving. The latter two will require two transformations, one to frame B and then to frame A, thus requiring the following trigonometric identities.

$$\sin\theta\cos\phi \pm \cos\theta\sin\phi = \sin(\theta \pm \phi) \tag{1}$$

$$\cos\theta\cos\phi \mp \sin\theta\sin\phi = \cos(\theta \pm \phi) \tag{2}$$

Using **Tables 1 and 2** and **Equations 1 and 2**, the unit vectors can then be rewritten in terms of frame A as follows.

$$\hat{b}_1 = \cos(q_1)\hat{a}_1 + \sin(q_1)\,\hat{a}_2 \tag{3}$$

$$\hat{b}_2 = -\sin(q_1)\hat{a}_1 + \cos(q_1)\,\hat{a}_2 \tag{4}$$

$$\hat{c}_1 = \cos(q_2)\hat{b}_1 + \sin(q_2)\,\hat{b}_2$$

$$\hat{c}_1 = \cos(q_2)\,[\cos(q_1)\hat{a}_1 + \sin(q_1)\,\hat{a}_2] + \sin(q_2)\,[-\sin(q_1)\hat{a}_1 + \cos(q_1)\,\hat{a}_2]$$

$$\hat{c}_1 = (\cos(q_1)\cos(q_2) - \sin(q_1)\sin(q_2))\hat{a}_1 + (\sin(q_1)\cos(q_2) + \cos(q_1)\sin(q_2))\hat{a}_2$$

$$\hat{c}_1 = \cos(q_1 + q_2)\,\hat{a}_1 + \sin(q_1 + q_2)\,\hat{a}_2 \tag{5}$$

$$\hat{c}_2 = -\sin(q_2)\hat{b}_1 + \cos(q_2)\,\hat{b}_2$$

$$\hat{c}_2 = -\sin(q_2)[\cos(q_1)\hat{a}_1 + \sin(q_1)\,\hat{a}_2] + \cos(q_2)\,[-\sin(q_1)\hat{a}_1 + \cos(q_1)\,\hat{a}_2]$$

$$\hat{c}_2 = -(\sin(q_1)\cos(q_2) + \cos(q_1)\sin(q_2))\hat{a}_1 + (\cos(q_1)\cos(q_2) - \sin(q_1)\sin(q_2))\hat{a}_2$$

$$\hat{c}_2 = -\sin(q_1 + q_2)\,\hat{a}_1 + \cos(q_1 + q_2)\,\hat{a}_2 \tag{6}$$

With all that out of the way, the velocities and accelerations can then be derived.

Linear Velocities

For velocity, the following kinematic equation for two points fixed on a rigid body can be used to find the velocities of both P and Q with respect to frame A.

$${}^{N}\vec{v}^{2} = {}^{N}\vec{v}^{1} + {}^{N}\vec{\omega}^{B} \times \vec{r}^{12} \tag{7}$$

Where ${}^{N}\vec{v}^{1}$ and ${}^{N}\vec{v}^{2}$ are the velocities of points 1 and 2 with respect to frame N, ${}^{N}\vec{\omega}^{B}$ is the angular velocity of the body B point 1 with respect to frame N, and $\vec{r}^{12}$ is the vector from point

1 to point 2. Noting that there is no velocity at O, the velocity at J ${}^{A}\vec{v}^{J}$ is derived as follows using **Equations 3, 4, 7**, and **Tables 3 and 4**.

$$ {}^{A}\vec{v}^{J} = {}^{A}\vec{v}^{O} + {}^{A}\vec{\omega}^{A} \times \vec{r}^{OJ} $$

$$ {}^{A}\vec{v}^{J} = 0 + u_1\hat{b}_3 \times \left(L_A\hat{b}_2\right) $$

$$ {}^{A}\vec{v}^{J} = -u_1 L_A\hat{b}_1 $$

$$ {}^{A}\vec{v}^{J} = -u_1 L_A(\cos(q_1)\hat{a}_1 + \sin(q_1)\,\hat{a}_2) $$

$$ {}^{A}\vec{v}^{J} = -u_1 L_A\cos(q_1)\hat{a}_1 - u_1 L_A\sin(q_1)\,\hat{a}_2 \tag{8} $$

Now that ${}^{A}\vec{v}^{J}$ has been derived, **Equations 5 to 8** and **Tables 3 and 4** can be used to derive the velocity at P ${}^{A}\vec{v}^{E}$.

$$ {}^{A}\vec{v}^{E} = {}^{A}\vec{v}^{J} + {}^{A}\vec{\omega}^{B} \times \vec{r}^{JE} $$

$$ {}^{A}\vec{v}^{E} = {}^{A}\vec{v}^{J} + (u_1 + u_2)\hat{c}_3 \times (L_B\hat{c}_2) $$

$$ {}^{A}\vec{v}^{E} = {}^{A}\vec{v}^{Q} - (u_1 + u_2)L_C\hat{c}_1 $$

$$ {}^{A}\vec{v}^{E} = -u_1 L_A(\cos(q_1)\hat{a}_1 + \sin(q_1)\,\hat{a}_2) - (u_1 + u_2)L_B(\cos(q_1 + q_2)\,\hat{a}_1 + \sin(q_1 + q_2)\,\hat{a}_2) $$

$$ \begin{aligned} {}^{A}\vec{v}^{E} = &-(u_1 L_A\cos(q_1) + (u_1 + u_2)L_B\cos(q_1 + q_2))\hat{a}_1 \\ &-(\dot{q}_1 L_A\sin(q_1) + (u_1 + u_2)L_B\sin(q_1 + q_2))\hat{a}_2 \end{aligned} \tag{9} $$

Linear Accelerations

The following kinematic equation can be used for acceleration at two points on a fixed rigid body can be used to find the acceleration at P and Q.

$$ {}^{N}\vec{a}^{2} = {}^{N}\vec{a}^{1} + {}^{N}\vec{\omega}^{B} \times ({}^{N}\vec{\omega}^{B} \times \vec{r}^{12}) + {}^{N}\vec{\alpha}^{B} \times \vec{r}^{12} \tag{10} $$

Where ${}^{N}\vec{a}^{1}$ and ${}^{N}\vec{a}^{2}$ are the velocities of points 1 and 2 with respect to frame N, and ${}^{N}\vec{\alpha}^{B}$ is the angular acceleration of body B with respect to frame N. Noting that there is no acceleration at O, the acceleration at J ,${}^{A}\vec{a}^{J}$ is derived as follows using **Tables 3 and 4** and **Equations 3, 4, and 10**.

$$ {}^{A}\vec{a}^{J} = {}^{A}\vec{a}^{O} + {}^{A}\vec{\omega}^{A} \times ({}^{A}\vec{\omega}^{A} \times \vec{r}^{OJ}) + {}^{A}\vec{\alpha}^{A} \times \vec{r}^{OJ} $$

$$ {}^{A}\vec{a}^{J} = 0 + u_1\hat{b}_3 \times \left(u_1\hat{b}_3 \times (-L_A)\hat{b}_2\right) + \dot{u}_1\hat{b}_3 \times \left(-L_A\hat{b}_2\right) $$

$$ {}^{A}\vec{a}^{J} = u_1\hat{b}_3 \times \left(-\dot{q}_1 L_A\hat{b}_1\right) - \dot{u}_1 L_A\hat{b}_1 $$

$$ {}^{A}\vec{a}^{J} = -u_1^2 L_A\hat{b}_2 - \dot{u}_1 L_A\hat{b}_1 $$

$$ {}^{A}\vec{a}^{J} = -u_1^2 L_A(-\sin(q_1)\hat{a}_1 + \cos(q_1)\,\hat{a}_2) - \dot{u}_1 L_A(\cos(q_1)\hat{a}_1 + \sin(q_1)\,\hat{a}_2) $$

$$ {}^{A}\vec{a}^{J} = (u_1^2 L_A\sin(q_1) - \dot{u}_1 L_A\cos(q_1))\hat{a}_1 - (u_1^2 L_A\cos(q_1) + \dot{u}_1 L_A\sin(q_1))\hat{a}_2 \tag{11} $$

Finally, ${}^A\vec{a}^E$ can be derived using **Equations 5, 6, 9, and 10** and **Tables 3 and 4** as follows.

$$ {}^A\vec{a}^E = {}^A\vec{a}^J + {}^A\vec{\omega}^B \times ({}^A\vec{\omega}^B \times \vec{r}^{QP}) + {}^A\vec{\alpha}^B \times \vec{r}^{JE} $$

$$ {}^A\vec{a}^E = {}^A\vec{a}^J + (u_1 + u_2)\hat{c}_3 \times \big((u_1 + u_2)\hat{c}_3 \times L_B\hat{c}_2\big) + (\dot{u}_1 + \dot{u}_2)\hat{c}_3 \times (-L_B\hat{c}_2) $$

$$ {}^A\vec{a}^E = {}^A\vec{a}^J + (u_1 + u_2)\hat{c}_3 \times (-(u_1 + u_2)L_B\hat{c}_1) - (\dot{u}_1 + \dot{u}_2)L_B\hat{c}_1 $$

$$ {}^A\vec{a}^E = {}^A\vec{a}^Q - (u_1 + u_2)^2 L_B\hat{c}_2 - (\dot{u}_1 + \dot{u}_2)L_B\hat{c}_1 $$

$$ \begin{aligned} {}^A\vec{a}^E = {} & -u_1^2 L_A(-\sin(q_1)\hat{a}_1 + \cos(q_1)\,\hat{a}_2) \\ & -\dot{u}_1 L_A(\cos(q_1)\hat{a}_1 + \sin(q_1)\,\hat{a}_2) \\ & -(\dot{u}_1 + \dot{u}_2)L_B(\cos(q_1 + q_2)\,\hat{a}_1 + \sin(q_1 + q_2)\,\hat{a}_2) \\ & -(u_1 + u_2)^2 L_B(-\sin(q_1 + q_2)\,\hat{a}_1 + \cos(q_1 + q_2)\,\hat{a}_2) \end{aligned} $$

$$ \begin{aligned} {}^A\vec{a}^E = {} & (u_1^2 L_A \sin(q_1) - \dot{u}_1 L_A \cos(q_1) - (\dot{u}_1 + \dot{u}_2)L_B \cos(q_1 + q_2) + (u_1 + u_2)^2 L_B \sin(q_1 + q_2))\hat{a}_1 \\ & -(u_1^2 L_A \cos(q_1) + \dot{u}_1 L_A \sin(q_1) + (\dot{u}_1 + \dot{u}_2)L_B \sin(q_1 + q_2) + (u_1 + u_2)^2 L_B \cos(q_1 + q_2))\hat{a}_2 \end{aligned} \quad (12) $$

Kane's Method

Now that the equations of velocity and acceleration at each point have been set up, Kane's method can be used to find the forces on each body.

Partial Velocities

Using Kane's method requires using the partial velocities with respect to the generalized velocity of each degree of freedom, which are u_1 and u_2 in our case. Velocities ${}^A\vec{v}^J$ and ${}^A\vec{v}^E$ can found from **Equations 8 and 9**. These partial velocities are as follows.

Table 5. Table of Partial Velocities

r	$\vec{v}_r^J$	$\vec{v}_r^E$
1	$\dfrac{\partial {}^A\vec{v}^J}{\partial u_1} = -L_A\cos(q_1)\hat{a}_1 - L_A\sin(q_1)\,\hat{a}_2$	$\dfrac{\partial {}^A\vec{v}^E}{\partial u_1} = -(L_A\cos(q_1) + L_B\cos(q_1 + q_2))\hat{a}_1 - (L_A\sin(q_1) + L_B\sin(q_1 + q_2))\hat{a}_2$
2	$\dfrac{\partial {}^A\vec{v}^Q}{\partial u_2} = 0$	$\dfrac{\partial {}^A\vec{v}^P}{\partial u_2} = -L_B\cos(q_1 + q_2)\,\hat{a}_1 - L_B\sin(q_1 + q_2)\,\hat{a}_2$

Kane's Method also requires using the partial angular velocities, again with respect to the degree-of-freedom velocities. Angular velocities ${}^A\vec{\omega}^B$ and ${}^A\vec{\omega}^C$ can be found from **Table 4** and their partial derivatives are as follows.

Table 6 Table of Partial Angular Velocities

r	${}^{A}\vec{\omega}^{B}$	${}^{A}\vec{\omega}^{C}$
1	$\frac{\partial\, {}^{A}\vec{\omega}^{B}}{\partial u_1} = \hat{a}_3$	$\frac{\partial\, {}^{A}\vec{\omega}^{C}}{\partial u_1} = \hat{a}_3$
2	$\frac{\partial\, {}^{A}\vec{\omega}^{B}}{\partial u_2} = 0$	$\frac{\partial\, {}^{A}\vec{\omega}^{C}}{\partial u_2} = \hat{a}_3$

Moment of inertia

To simplify the overall calculations, it was decided to model the two links as thin rods, which means the only relevant dimension would be their lengths. Assuming that the mass distribution is uniform across each link and that the moment of inertia is about one end they would have the following 3D inertia tensors and corresponding dyadics:

$$I_B = \frac{1}{3} m_B L_A^2 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \vec{\vec{I}}_B = \frac{1}{3} m_B L_A^2 \left(\hat{b}_1^2 + \hat{b}_3^2\right) \tag{13}$$

$$I_C = \frac{1}{3} m_C L_B^2 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \vec{\vec{I}}_C = \frac{1}{3} m_C L_B^2 (\hat{c}_1^2 + \hat{c}_3^2) \tag{14}$$

Both dyadics can then be expressed in terms of Newtonian Frame A using **Equation 1** – where $\theta = \phi$ – and **Equations 3 to 6**.

$$\vec{\vec{I}}_B = \frac{1}{3} m_B L_A^2 \left[(\cos(q_1)\hat{a}_1 + \sin(q_1)\,\hat{a}_2)^2 + {\hat{a}_3}^2\right]$$

$$\vec{\vec{I}}_B = \frac{1}{3} m_B L_A^2 \left[((\cos(q_1)\hat{a}_1)^2 + 2\cos(q_1)\sin(q_1)\,\hat{a}_1\hat{a}_2 + (\sin(q_1)\,\hat{a}_2)^2) + {\hat{a}_3}^2\right]$$

$$\vec{\vec{I}}_B = \frac{1}{3} m_B L_A^2 \left[((\cos(q_1)\hat{a}_1)^2 + 2\sin(2q_1)\,\hat{a}_1\hat{a}_2 + (\sin(q_1)\,\hat{a}_2)^2) + {\hat{a}_3}^2\right] \tag{15}$$

$$\vec{\vec{I}}_C = \frac{1}{3} m_B L_A^2 \left[(\cos(q_1 + q_2)\,\hat{a}_1 + \sin(q_1 + q_2)\,\hat{a}_2)^2 + {\hat{a}_3}^2\right]$$

$$\vec{\vec{I}}_C = \frac{1}{3} m_B L_A^2 \left[((\cos(q_1 + q_2)\hat{a}_1)^2 + 2\cos(q_1 + q_2)\sin(q_1 + q_2)\,\hat{a}_1\hat{a}_2 + (\sin(q_1 + q_2)\,\hat{a}_2)^2) + {\hat{a}_3}^2\right]$$

$$\vec{\vec{I}}_C = \frac{1}{3} m_C L_B^2 \left[\left((\cos(q_1 + q_2)\hat{a}_1)^2 + 2\sin\left(2(q_1 + q_2)\right)\hat{a}_1\hat{a}_2 + (\sin(q_1 + q_2)\,\hat{a}_2)^2\right) + {\hat{a}_3}^2\right] \tag{16}$$

Solving for Kane's Equations

Kane's equation is as follows, where r is the degree-of-freedom of the system, F_r are the generalized active forces with respect to r, and F_r^* are the generalized inertial forces.

$$F_r + F_r^* = 0 \tag{17}$$

These two terms can then be defined as follows, where $\vec{\omega}_r^{B_k}$ is the partial angular velocity of body k with respect to r, $\vec{v}_r^{B_k}$ is the partial linear velocity of body k with respect to r, $\vec{R}^{B_k}$ is the resultant force acting on body k, and $\vec{T}^{B_k}$ is the resultant moment acting on body k – where * denotes the inertial resultant.

$$F_r = \sum_{k=1}^{N} \left(\vec{\omega}_r^{B_k} \cdot \vec{T}^{B_k} + \vec{v}_r^{B_k} \cdot \vec{R}^{B_k}\right) \tag{18}$$

$$F_r^* = \sum_{k=1}^{N} \left(\vec{\omega}_r^{B_k} \cdot \vec{T}^{B_k^*} + \vec{v}_r^{B_k} \cdot \vec{R}^{B_k^*}\right) \tag{19}$$

The inertial resultant fore and moments are also defined here, where m^{B_k} is the mass of body k, and $\vec{a}^{B_k}$ is the linear acceleration of body k.

$$\vec{R}^{B_k^*} = -m^{B_k}\vec{a}^{B_k} \tag{20}$$

$$\vec{T}^{B_k^*} = -\vec{\bar{I}}^{B_k} \cdot \vec{\alpha}^{B_k} - \vec{\omega}^{B_k} \times \left(\vec{\bar{I}}^{B_k} \cdot \vec{\omega}^{B_k}\right) \tag{21}$$

In this instance, body 1 is link B and body 2 is link C. The active resultant force acting on both bodies is the force of gravity, and the active resultant torque is simulated motor torques at points O and J, which are denoted by $\vec{T}^B$ and $\vec{T}^C$, respectively. Taking this all into account the equations can then be rewritten as follows.

$$F_r = \vec{v}_r^B \cdot (-m^B g \hat{a}_2) + \vec{v}_r^C \cdot (-m^C g \hat{a}_2) + \vec{\omega}_r^B \cdot \vec{T}^B + \vec{\omega}_r^C \cdot \vec{T}^C \tag{22}$$

$$F_r^* = \vec{v}_r^C \cdot (-m^C \vec{a}^C) + \vec{v}_r^C (-m^C \vec{a}^P) + \vec{\omega}_r^B \cdot \vec{T}^{B^*} + \vec{\omega}_r^C \cdot \vec{T}^{C^*} \tag{23}$$

Substituting the $\vec{\bar{I}}_B$ *and* $\vec{\bar{I}}_C$is used to obtain $\vec{T}^B$ and $\vec{T}^C$ are shown in Eq. (22) and Eq. (26).

$$\vec{T}^{B^*} = -\vec{\bar{I}}^B \cdot \vec{\alpha}^B - \vec{\omega}^B \times \left(\vec{\bar{I}}^B \cdot \vec{\omega}^B\right)$$

$$\vec{T}^{B^*} = -\frac{1}{3} m_B L_A^2 \left[((\cos(q_1)\hat{a}_1)^2 + 2\sin(2q_1)\,\hat{a}_1\hat{a}_2 + (\sin(q_1)\,\hat{a}_2)^2) + \hat{a}_3{}^2\right] \cdot \dot{u}_1 \hat{b}_3 - u_1 \hat{b}_3 \times \left(\frac{1}{3} m_B L_A^2 \left[((\cos(q_1)\hat{a}_1)^2 + 2\sin(2q_1)\,\hat{a}_1\hat{a}_2 + (\sin(q_1)\,\hat{a}_2)^2) + \hat{a}_3{}^2\right] \cdot u_1 \hat{b}_3\right)$$

$$\vec{T}^{B^*} = -\frac{1}{3}m_B L_A^2\left[((\cos(q_1)\hat{a}_1)^2 + 2\sin(2q_1)\,\hat{a}_1\hat{a}_2 + (\sin(q_1)\,\hat{a}_2)^2) + \hat{a}_3{}^2\right]\cdot u_1\hat{a}_3$$
$$- u_1\hat{a}_3$$
$$\times\left(\frac{1}{3}m_B L_A^2\left[((\cos(q_1)\hat{a}_1)^2 + 2\sin(2q_1)\,\hat{a}_1\hat{a}_2 + (\sin(q_1)\,\hat{a}_2)^2) + \hat{a}_3{}^2\right]\cdot u_1\hat{a}_3\right)$$

$$\vec{T}^{B^*} = -\frac{1}{3}m_B L_A^2 u_1\hat{a}_3 - u_1\hat{a}_3 \times\left(\frac{1}{3}m_B L_A^2 u_1\hat{a}_3\right)$$

$$\vec{T}^{B^*} = -\frac{1}{3}m_B L_A^2 u_1\hat{a}_3 \tag{23}$$

$$\vec{T}^{C^*} = -\vec{\bar{I}}^C\cdot\vec{a}^C - \vec{\omega}^C\times\left(\vec{\bar{I}}^C\cdot\vec{\omega}^C\right)$$

$$\vec{T}^{C^*} = -\frac{1}{3}m_C L_B^2\left[\left((\cos(q_1+q_2)\hat{a}_1)^2 + 2\sin\big(2(q_1+q_2)\big)\,\hat{a}_1\hat{a}_2 + (\sin(q_1+q_2)\,\hat{a}_2)^2\right) + \hat{a}_3{}^2\right]$$
$$\cdot(u_1+u_2)\hat{c}_3$$
$$-(u_1+u_2)\hat{c}_3$$
$$\times\left(\frac{1}{3}m_C L_B^2\left[\begin{pmatrix}(\cos(q_1+q_2)\hat{a}_1)^2 + 2\sin\big(2(q_1+q_2)\big)\,\hat{a}_1\hat{a}_2\\ +(\sin(q_1+q_2)\,\hat{a}_2)^2\end{pmatrix} + \hat{a}_3{}^2\right]\right.$$
$$\left.\cdot(u_1+u_2)\hat{c}_3\right)$$

$$\vec{T}^{C^*} = -\frac{1}{3}m_C L_B^2\left[\left((\cos(q_1+q_2)\hat{a}_1)^2 + 2\sin\big(2(q_1+q_2)\big)\,\hat{a}_1\hat{a}_2 + (\sin(q_1+q_2)\,\hat{a}_2)^2\right) + \hat{a}_3{}^2\right]$$
$$\cdot(u_1+u_2)\hat{a}_3$$
$$-(u_1+u_2)\hat{a}_3$$
$$\times\left(\frac{1}{3}m_C L_B^2\left[\begin{pmatrix}(\cos(q_1+q_2)\hat{a}_1)^2 + 2\sin\big(2(q_1+q_2)\big)\,\hat{a}_1\hat{a}_2\\ +(\sin(q_1+q_2)\,\hat{a}_2)^2\end{pmatrix} + \hat{a}_3{}^2\right]\right.$$
$$\left.\cdot(u_1+u_2)\hat{a}_3\right)$$

$$\vec{T}^{C^*} = -\frac{1}{3}m_B L_A^2(u_1+u_2)\hat{a}_3 - (u_1+u_2)\hat{a}_3\times\left(\frac{1}{3}m_B L_A^2(u_1+u_2)\hat{a}_3\right)$$

$$\vec{T}^{C^*} = -\frac{1}{3}m_C L_B^2(u_1+u_2)\hat{a}_3 \tag{25}$$

Now the Kane Equations can be calculated. First, we will calculate F_1 and F_2.

$$F_1 = \vec{v}_1^J\cdot(-m^B g\hat{a}_2) + \vec{v}_1^E\cdot(-m^C g\hat{a}_2) + \vec{\omega}_1^B\cdot\vec{T}^B + \vec{\omega}_1^C\cdot\vec{T}^C$$

$$F_1 = (-L_A\cos(q_1)\hat{a}_1 - L_A\sin(q_1)\,\hat{a}_2)\cdot(-m^B g\hat{a}_2)$$
$$+(-(L_A\cos(q_1) + L_B\cos(q_1+q_2))\hat{a}_1 - (L_A\sin(q_1) + L_B\sin(q_1+q_2))\hat{a}_2)$$
$$\cdot(-m^C g\hat{a}_2) + \hat{a}_3\cdot T^B L_A\hat{a}_3 + \hat{a}_3\cdot T^C L_B\hat{a}_3$$

$$F_1 = L_A \sin(q_1) m^B g + (L_A \sin(q_1) + L_B \sin(q_1 + q_2)) m^C g + T^B L_A + T^C L_B \qquad (26)$$

$$F_2 = \vec{v}_2^J \cdot (-m^B g \hat{a}_2) + \vec{v}_2^E \cdot (-m^C g \hat{a}_2) + \vec{\omega}_2^B \cdot \vec{T}^B + \vec{\omega}_2^C \cdot \vec{T}^C$$

$$F_2 = (-L_B \cos(q_1 + q_2)\, \hat{a}_1 - L_B \sin(q_1 + q_2)\, \hat{a}_2) \cdot (-m^C g \hat{a}_2) + \hat{a}_3 \cdot T^C L_B \hat{a}_3$$

$$F_2 = L_B\, m^C g \sin(q_1 + q_2) + T^C L_B$$

Next, we will calculate F_1^* and F_2^* .

$$F_1^* = \vec{v}_1^J \cdot (-m^B \vec{a}^J) + \vec{v}_1^E (-m^C \vec{a}^E) + \vec{\omega}_1^B \cdot \vec{T}^{B^*} + \vec{\omega}_1^C \cdot \vec{T}^{C^*}$$

$$\begin{aligned} F_1^* = {} & (-L_A \cos(q_1) \hat{a}_1 - L_A \sin(q_1)\, \hat{a}_2) \\ & \cdot -m^B [(u_1^2 L_A \sin(q_1) - \dot{u}_1 L_A \cos(q_1)) \hat{a}_1 - (u_1^2 L_A \cos(q_1) + \dot{u}_1 L_A \sin(q_1)) \hat{a}_2] \\ & + [-(L_A \cos(q_1) + L_B \cos(q_1 + q_2)) \hat{a}_1 - (L_A \sin(q_1) + L_B \sin(q_1 + q_2)) \hat{a}_2] \\ & \cdot -m^C [(u_1^2 L_A \sin(q_1) - n_1 L_A \cos(q_1) - (\dot{u}_1 + \dot{u}_2) L_B \cos(q_1 + q_2) \\ & + (u_1 + u_2)^2 L_B \sin(q_1 + q_2)) \hat{a}_1 \\ & - (u_1^2 L_A \cos(q_1) + n_1 L_A \sin(q_1) + (\dot{u}_1 + \dot{u}_2) L_B \sin(q_1 + q_2) \\ & + (u_1 + u_2)^2 L_B \cos(q_1 + q_2)) \hat{a}_2] + \hat{a}_3 \cdot -\frac{1}{3} m_B L_A^2 \dot{u}_1 \hat{a}_3 + \hat{a}_3 \\ & \cdot -\frac{1}{3} m_C L_B^2 (\dot{u}_1 + \dot{u}_2) \hat{a}_3 \end{aligned}$$

$$\begin{aligned} F_1^* = {} & m^C [(L_B \cos(q_1 + q_2) + L_A \cos(q_1)) \\ & * ((u_1 + u_2)^2 L_B \sin(q_1 + q_2) + u_1^2 L_A \sin(q_1) - (\dot{u}_1 + \dot{u}_2) L_B \cos(q_1 + q_2) \\ & - \dot{u}_1 L_A \cos(q_1))] \\ & + m^C [(L_B \sin(q_1 + q_2) + L_A \sin(q_1)) \\ & * (\dot{u}_1 L_A \sin(q_1) + (u_1 + u_2)^2 L_B \sin(q_1 + q_2) + u_1^2 L_A \cos(q_1) \\ & + (\dot{u}_1 + \dot{u}_2) L_B \sin(q_1 + q_2))] + L_A m^B \cos(q_1)\, (u_1^2 L_A \sin(q_1) - \dot{u}_1 L_A \cos(q_1)) \\ & - L_A m^B \sin(q_1)\, (u_1^2 L_A \cos(q_1) + \dot{u}_1 L_A \sin(q_1)) \\ & - \frac{1}{3} \left(m_B L_A^2 \dot{u}_1 + m_C L_B^2 (\dot{u}_1 + \dot{u}_2) \right) \end{aligned}$$

$$F_2^* = \vec{v}_2^J \cdot (-m^C \vec{a}^C) + \vec{v}_2^E \cdot (-m^C \vec{a}^P) + \vec{\omega}_2^B \cdot \vec{T}^{B^*} + \vec{\omega}_2^C \cdot \vec{T}^{C^*}$$

$$\begin{aligned} F_2^* = {} & (-L_B \cos(q_1 + q_2)\, \hat{a}_1 - L_B \sin(q_1 + q_2)\, \hat{a}_2) \\ & \cdot -m^C [(u_1^2 L_A \sin(q_1) - \dot{u}_1 L_A \cos(q_1) - (\dot{u}_1 + \dot{u}_2) L_B \cos(q_1 + q_2) \\ & + (u_1 + u_2)^2 L_B \sin(q_1 + q_2)) \hat{a}_1 \\ & - (u_1^2 L_A \cos(q_1) + \dot{u}_1 L_A \sin(q_1) + (\dot{u}_1 + \dot{u}_2) L_B \sin(q_1 + q_2) \\ & + (u_1 + u_2)^2 L_B \cos(q_1 + q_2)) \hat{a}_2] + \hat{a}_3 \cdot -\frac{1}{3} m_C L_B^2 (\dot{u}_1 + \dot{u}_2) \hat{a}_3 \end{aligned}$$

$$F_2^* = L_A m^C \cos(q_1 + q_2)\,[(u_1 + u_2)^2 L_B \sin(q_1 + q_2) + u_1^2 L_A \sin(q_1) - (\dot{u}_1 + \dot{u}_2) L_B \cos(q_1 + q_2) - \dot{u}_1 L_A \cos(q_1)] + L_A m^C \sin(q_1 + q_2)\,[\dot{u}_1 L_A \sin(q_1) + (u_1 + u_2)^2 L_B \sin(q_1 + q_2) - u_1^2 L_A \cos(q_1) + (\dot{u}_1 + \dot{u}_2) L_B \sin(q_1 + q_2)] - \frac{1}{3} m_C L_B^2 (\dot{u}_1 + \dot{u}_2)$$

Combining these equations, we get the following equations of motion with respect to each degree of freedom.

$$F_1^* + F_1 = m^C[(L_B \cos(q_1 + q_2) + L_A \cos(q_1)\,) * ((u_1 + u_2)^2 L_B \sin(q_1 + q_2) + u_1^2 L_A \sin(q_1) - (\dot{u}_1 + \dot{u}_2) L_B \cos(q_1 + q_2) - \dot{u}_1 L_A \cos(q_1))] + m^C[(L_B \sin(q_1 + q_2) + L_A \sin(q_1)) * (\dot{u}_1 L_A \sin(q_1) + (u_1 + u_2)^2 L_B \sin(q_1 + q_2) + u_1^2 L_A \cos(q_1) + (\dot{u}_1 + \dot{u}_2) L_B \sin(q_1 + q_2))] + L_A m^B \cos(q_1)\,(u_1^2 L_A \sin(q_1) - \dot{u}_1 L_A \cos(q_1)) - L_A m^B \sin(q_1)\,(u_1^2 L_A \cos(q_1) + \dot{u}_1 L_A \sin(q_1)) - \frac{1}{3}\left(m_B L_A^2 \dot{u}_1 + m_C L_B^2 (\dot{u}_1 + \dot{u}_2)\right) + L_A \sin(q_1) m^B g + (L_A \sin(q_1) + L_B \sin(q_1 + q_2)) m^C g + T^B L_A + T^C L_B$$

$$F_2^* + F_2 = L_A m^C \cos(q_1 + q_2)\,[(u_1 + u_2)^2 L_B \sin(q_1 + q_2) + u_1^2 L_A \sin(q_1) - (\dot{u}_1 + \dot{u}_2) L_B \cos(q_1 + q_2) - \dot{u}_1 L_A \cos(q_1)] + L_A m^C \sin(q_1 + q_2)\,[\dot{u}_1 L_A \sin(q_1) + (u_1 + u_2)^2 L_B \sin(q_1 + q_2) - u_1^2 L_A \cos(q_1) + (\dot{u}_1 + \dot{u}_2) L_B \sin(q_1 + q_2)] - \frac{1}{3} m_C L_B^2 (\dot{u}_1 + \dot{u}_2) + L_B\, m^C g \sin(q_1 + q_2) + T^C L_B$$

Autolev Derivation

To ensure that the derivation of the prior section was correct, the equations of motion were rederived using AutoLev; the code for that is shown below.

```
   (1) % PROJECT2.AL
   (2) NEWTONIAN A
   (3) Bodies B,C
   (4) CONSTANTS g, LA, LB,TB,TC
   (5) POINTS O, J, E
   (6) VARIABLES q1',q2'
   (7) MOTIONVARIABLES' u1',u2'
   (8) q1' = u1; q2' = u2
-> (9) q1' = u1
-> (10) q2' = u2

   (11) SIMPROT(B,A,3,q1)
-> (12) B_A = [COS(q1), -SIN(q1), 0; SIN(q1), COS(q1), 0; 0, 0, 1]

   (13) SIMPROT(C,B,3,q2)
-> (14) C_B = [COS(q2), -SIN(q2), 0; SIN(q2), COS(q2), 0; 0, 0, 1]
```

```
   (15) w_A> = u1*b3>
-> (16) w_A> = u1*B3>

   (17) w_B> = (u1+u2)*c3>
-> (18) w_B> = (u1+u2)*C3>

   (19) Alpha_A> = u1'*b3>
-> (20) Alpha_A> = u1'*B3>

   (21) Alpha_B> = (u1'+u2')*c3>
-> (22) Alpha_B> = (u1'+u2')*C3>

   (23) R_OJ> = LA*B2>
-> (24) R_OJ> = LA*B2>

   (25) R_JE> = LB*C2>
-> (26) R_JE> = LB*C2>

   (27) V_J_A>= CROSS(w_A>,R_OJ>)
-> (28) V_J_A> = -LA*u1*B1>

   (29) V_E_A> = V_J_A> + CROSS(w_B>,R_JE>)
-> (30) V_E_A> = -LA*u1*B1> - LB*(u1+u2)*C1>

   (31) EXPRESS(V_J_A>,A)
-> (32) V_J_A> = -LA*COS(q1)*u1*A1> + LA*SIN(q1)*u1*A2>

   (33) EXPRESS(V_E_A>,A)
-> (34) V_E_A> = (-LA*COS(q1)*u1-LB*COS(q1+q2)*(u1+u2))*A1> +
(LA*SIN(q1)*u1+
        LB*SIN(q1+q2)*(u1+u2))*A2>

   (35) A_J_A> = CROSS(w_A>,CROSS(w_A>,R_OJ>))+CROSS(Alpha_A>,R_OJ>)
-> (36) A_J_A> = -LA*u1'*B1> - LA*u1^2*B2>

   (37) A_E_A> = A_J_A> +
CROSS(w_B>,CROSS(w_B>,R_JE>))+CROSS(Alpha_B>,R_JE>)
-> (38) A_E_A> = -LA*u1'*B1> - LA*u1^2*B2> - LB*(u1'+u2')*C1> -
LB*(u1+u2)^2*
        C2>

   (39) EXPRESS(A_J_A>,A)
-> (40) A_J_A> = -LA*(SIN(q1)*u1^2+COS(q1)*u1')*A1> - LA*(COS(q1)*u1^2-
SIN(q1)*
        u1')*A2>

   (41) EXPRESS(A_E_A>,A)
-> (42) A_E_A> = (-LA*SIN(q1)*u1^2-LB*SIN(q1+q2)*(u1+u2)^2-LA*COS(q1)*u1'-
LB*
        COS(q1+q2)*(u1'+u2'))*A1> +
(LA*SIN(q1)*u1'+LB*SIN(q1+q2)*(u1'+u2')-LA*
        COS(q1)*u1^2-LB*COS(q1+q2)*(u1+u2)^2)*A2>
```

```
   (43) Mass B=mlB
   (44) Mass C=mlC
   (45) force(J,-mlB*g*a2>)
-> (46) FORCE_J> = -g*mlB*A2>

   (47) force(E,-mlC*g*a2>)
-> (48) FORCE_E> = -g*mlC*A2>

   (49) %Define Inertia Tensor
   (50) Inertia B, (1/3)*mlB*LA^2, 0, (1/3)*mlB*LA^2
-> (51) I_B_BO>> = 0.3333333*mlB*LA^2*B1>*B1> + 0.3333333*mlB*LA^2*B3>*B3>

   (52) Inertia C, (1/3)*mlB*LB^2, 0, (1/3)*mlB*LB^2
-> (53) I_C_CO>> = 0.3333333*mlB*LB^2*C1>*C1> + 0.3333333*mlB*LB^2*C3>*C3>

   (54) %Define Torque
   (55) Torque(B,b3>*TB)
-> (56) TORQUE_B> = TB*B3>

   (57) Torque(C,c3>*TC)
-> (58) TORQUE_C> = TC*C3>

   (59) V_O_A> = 0>
-> (60) V_O_A> = 0>

   (61) P_O_BO> = 1/2*LA*B2>
-> (62) P_O_BO> = 0.5*LA*B2>

   (63) V_J_B> = EXPRESS(V_J_A>,B)
-> (64) V_J_B> = -LA*u1*B1>

   (65) P_J_CO> = 1/2*LB*C2>
-> (66) P_J_CO> = 0.5*LB*C2>

   (67) v2pts(A,B,O,BO)
-> (68) V_BO_A> = 0.5*LA*u1*B1>

   (69) v2pts(B,C,J,CO)
-> (70) V_CO_B> = -LA*u1*B1> + 0.5*LB*u2*C1>

   (71) V_CO_A> = EXPRESS(V_CO_B>,A)
-> (72) V_CO_A> = (0.5*LB*COS(q1+q2)*u2-LA*COS(q1)*u1)*A1> +
(LA*SIN(q1)*u1-0.5
        *LB*SIN(q1+q2)*u2)*A2>

   (73) % Accelerations
   (74) A_O_A> = 0>
-> (75) A_O_A> = 0>

   (76) A_J_B> = EXPRESS(A_J_A>,B)
-> (77) A_J_B> = -LA*u1'*B1> - LA*u1^2*B2>

   (78) a2pts(A,B,O,BO)
-> (79) A_BO_A> = 0.5*LA*u1'*B1> - 0.5*LA*u1^2*B2>
```

```
   (80) a2pts(B,C,J,CO)
-> (81) A_CO_B> = -LA*u1'*B1> - LA*u1^2*B2> + 0.5*LB*u2'*C1> -
0.5*LB*u2^2*C2>

   (82) A_CO_A> = EXPRESS(A_CO_B>,A)
-> (83) A_CO_A> = (0.5*LB*COS(q1+q2)*u2'-LA*SIN(q1)*u1^2-
0.5*LB*SIN(q1+q2)*u2^2
        -LA*COS(q1)*u1')*A1> + (LA*SIN(q1)*u1'-LA*COS(q1)*u1^2-
0.5*LB*COS(q1+
        q2)*u2^2-0.5*LB*SIN(q1+q2)*u2')*A2>

   (84) zero =fr()+frstar()
-> (85) zero[1] = -TB - TC - g*LA*mlB*SIN(q1) -
g*mlC*(LA*SIN(q1)+LB*SIN(q1+q2)
        ) - 0.5*LA*LB*mlC*SIN(q2)*u2^2 - 0.3333333*LB*(LB*mlB-
1.5*LA*mlC*COS(
        q2))*u2' - 0.3333333*(mlB*LB^2+1.75*mlB*LA^2+3*mlC*LA^2)*u1'

-> (86) zero[2] = -TC - g*LB*mlC*SIN(q1+q2) -
0.25*LB*(2*LA*mlC*SIN(q2)*u1^2+
        LB*(mlC+1.333333*mlB)*u2'+1.333333*(LB*mlB-
1.5*LA*mlC*COS(q2))*u1')

   (87) kane()
-> (88) ZERO[1] = -TB - TC - g*LA*mlB*SIN(q1) -
g*mlC*(LA*SIN(q1)+LB*SIN(q1+q2)
        ) - 0.5*LA*LB*mlC*SIN(q2)*u2^2 - 0.3333333*LB*(LB*mlB-
1.5*LA*mlC*COS(
        q2))*u2' - 0.3333333*(mlB*LB^2+1.75*mlB*LA^2+3*mlC*LA^2)*u1'

-> (89) ZERO[2] = -TC - g*LB*mlC*SIN(q1+q2) - 0.5*LA*LB*mlC*SIN(q2)*u1^2 -
0.25
        *LB^2*(mlC+1.333333*mlB)*u2' - 0.3333333*LB*(LB*mlB-
1.5*LA*mlC*COS(q2))
        *u1'

   (90) R_OE> = R_OJ> + R_JE>
-> (91) R_OE> = LA*B2> + LB*C2>

   (92) EXPRESS(R_OE>,A)
-> (93) R_OE> = (LA*SIN(q1)+LB*SIN(q1+q2))*A1> +
(LA*COS(q1)+LB*COS(q1+q2))*A2>

   (94) mag = mag(R_OE>)
-> (95) mag =
((LA*SIN(q1)+LB*SIN(q1+q2))^2+(LA*COS(q1)+LB*COS(q1+q2))^2)^0.5

   (96) R_OE_x = cos(q1)*mag(R_OE>)
-> (97) R_OE_x =
COS(q1)*((LA*SIN(q1)+LB*SIN(q1+q2))^2+(LA*COS(q1)+LB*COS(q1+
        q2))^2)^0.5

   (98) R_OE_y = sin(q1)*mag(R_OE>)
```

```
-> (99) R_OE_y =
SIN(q1)*((LA*SIN(q1)+LB*SIN(q1+q2))^2+(LA*COS(q1)+LB*COS(q1+
        q2))^2)^0.5

   (100) % Set units of each component
   (101) units g=m/sec^2,la=m, lb=m, mlB=kg, mlC=kg,
t=sec,q1=deg,q2=deg,q1=rad/sec,u2=rad/sec, TB = N*m, TC = N*m
   (102) % Set the values of the constants
   (103) input la=1, lb=2, g=9.81, mlB=10, mlC=20
   (104) % Set the starting value
   (105) input q1=10, q2=10,u1=0,u2=0,TB=1,TC = 1
   (106) %set the general simulation factors
   (107) input tfinal=16, integstp=0.1, absErr=1.0E-07, relErr=1.0E-07
   (108) %output the value
   (109) output t, q1,q2,u1,u2,R_OE_x, R_OE_y
   (110) % save the code as matlab code
   (111) code dynamics() project2.m,subs
```

Simulation Results and Analysis

Due to time constraints, only a limited simulation of the robot arm could be performed. This consisted of applying differing torques about points O and J; the generalized velocities and accelerations were recorded along with the motion of both the end effector – represented by point E – and the joint of point J. The simulation itself was conducted over a relatively short period of time – 0.5 seconds – since an actual robot arm would likely not take a long time to complete a single motion.

Parameters

As shown in the Autolev code in *Appendix A*, the simulation used the following values for the constants and initial values for the motion variables. The two values listed for the torque variables are switched between for each simulation.

Table 7 Simulation Parameters

Constant	Value	Motion Variable	Initial Value
L_A	$1\ m$	q_1	0°
L_B	$1\ m$	q_2	0°
g	$9.81\ m/s^2$	u_1	$0\ rad/s$
m^B	$1\ kg$	u_2	$0\ rad/s$
m^C	$1\ kg$		
T^B, T^C	$-10\ Nm, -20\ Nm$		

Results

The graphs shown below represent the results of the four different simulations; for each simulation, they are ordered by the generalized coordinates, generalized velocities, and the movement of the end effector and joint.

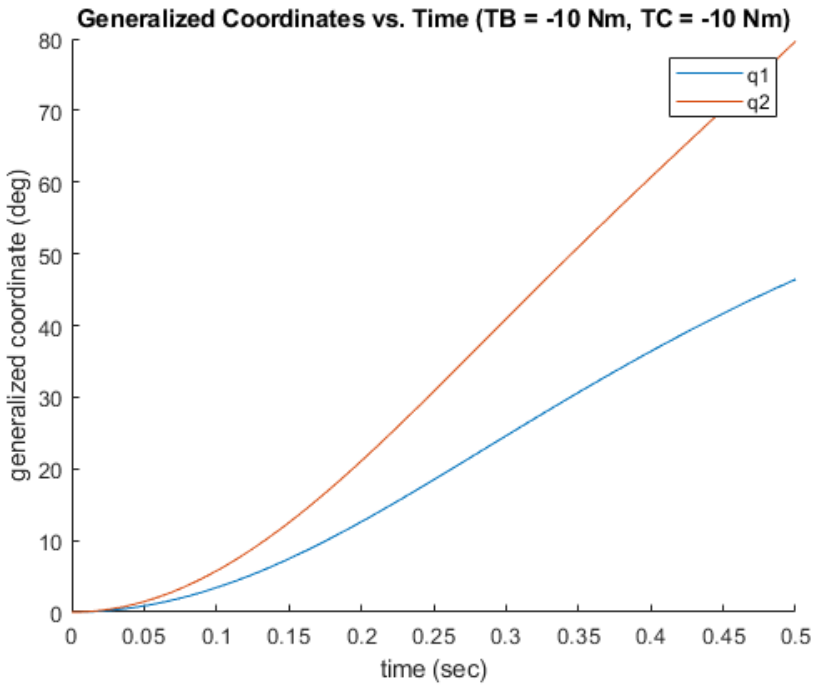

Figure 5. Generalized Coordinates VS. Time for TB = -10 Nm, TC = -10 Nm

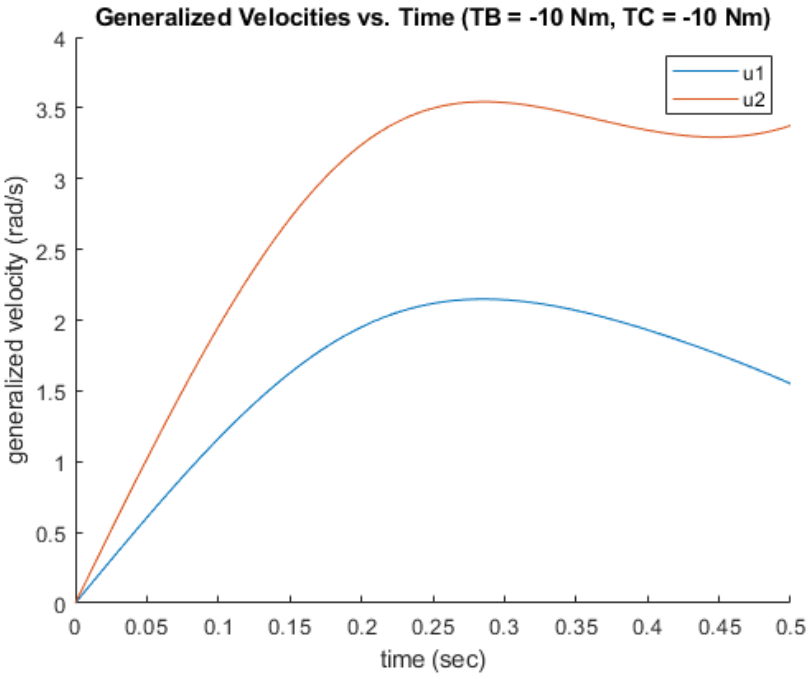

Figure 6. Generalized Velocities VS. Time for TB = -10 Nm, TC = -10 Nm

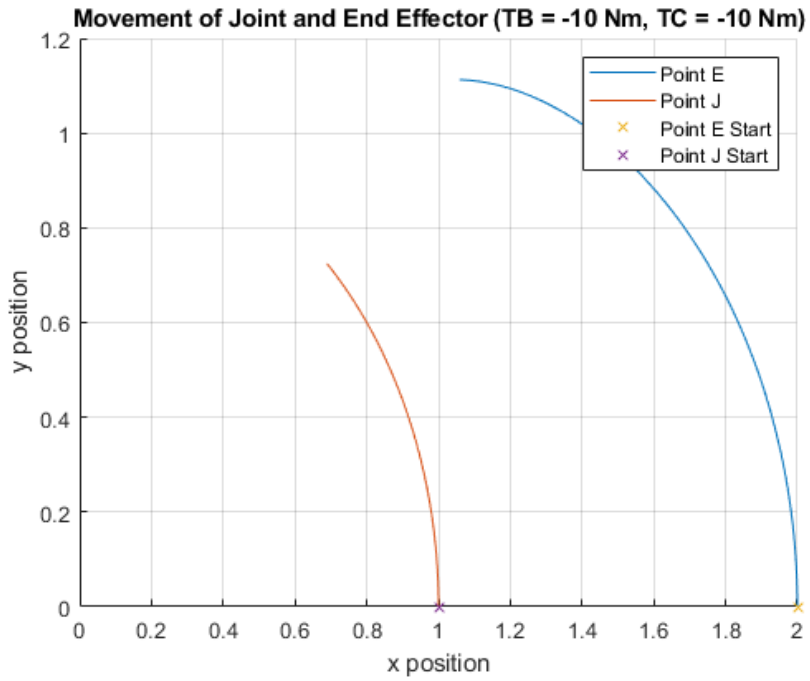

Figure 7. Figure of Joint Movement and End Effector for TB = -10 Nm, TC = -10 Nm

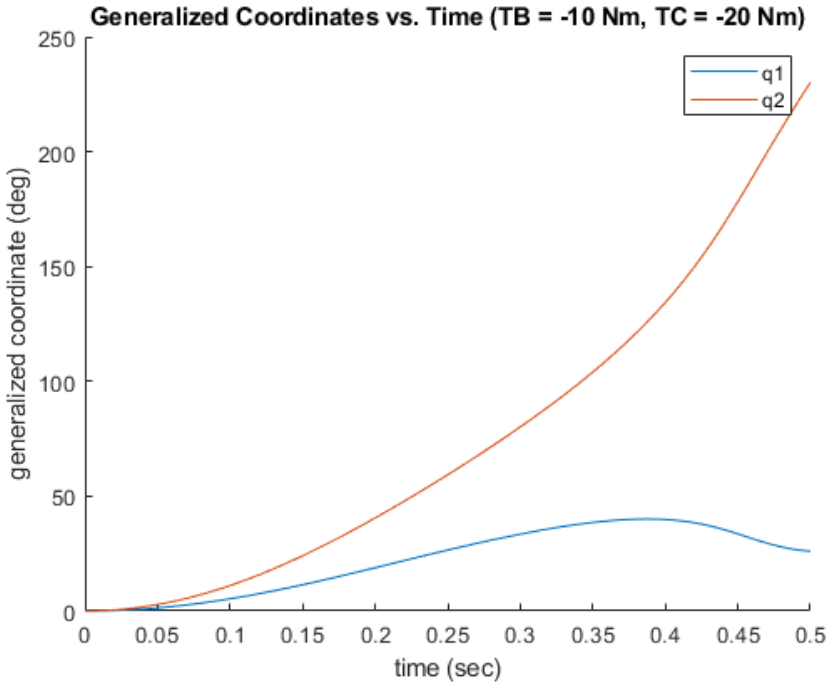

Figure 8 Generalized Coordinates VS. Time for TB = -10 Nm, TC = -20 Nm

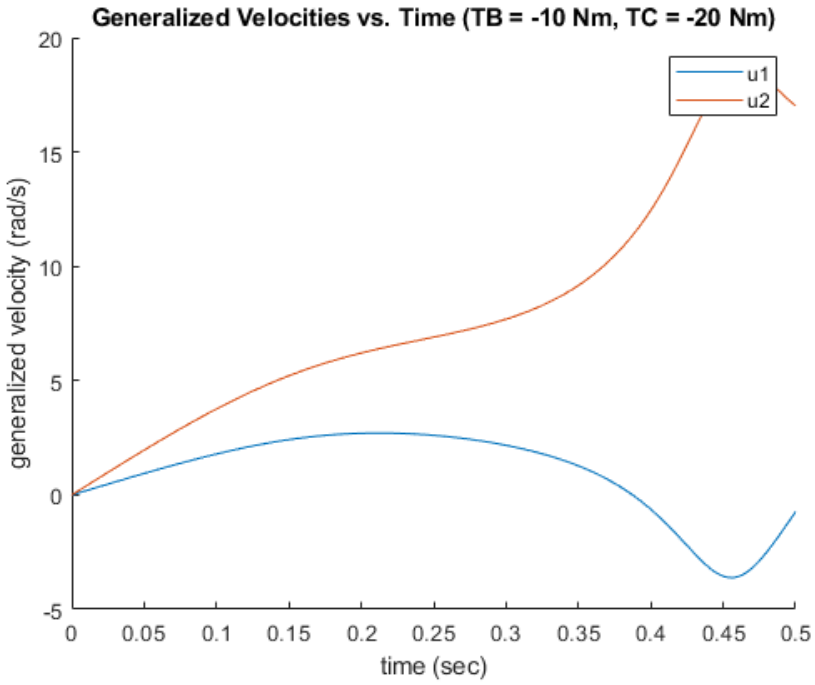

Figure 9 Generalized Velocities VS. Time for TB = -10 Nm, TC = -20 Nm

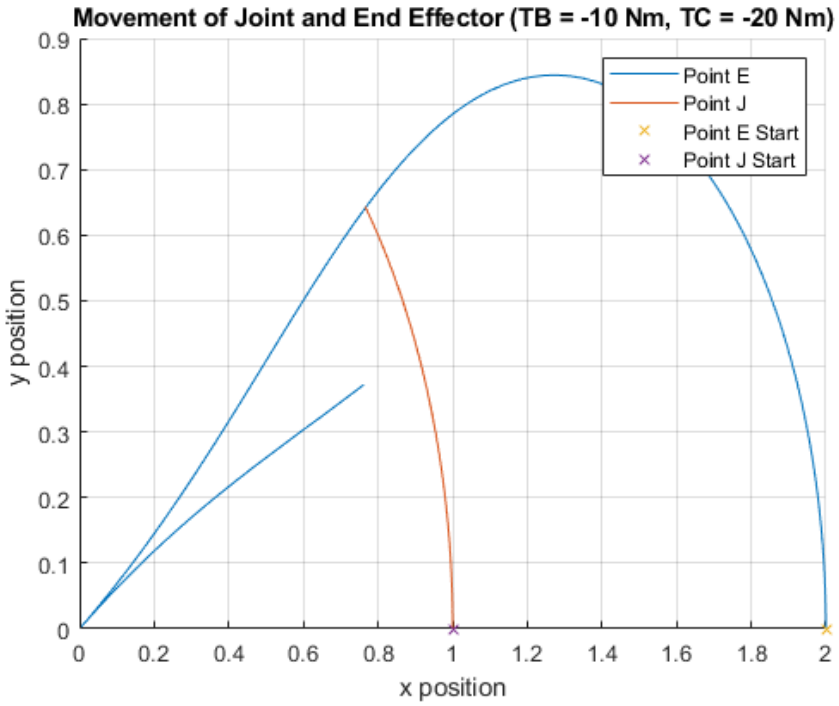

Figure 10 Figure of Joint Movement and End Effector for TB = -10 Nm, TC = -20 Nm

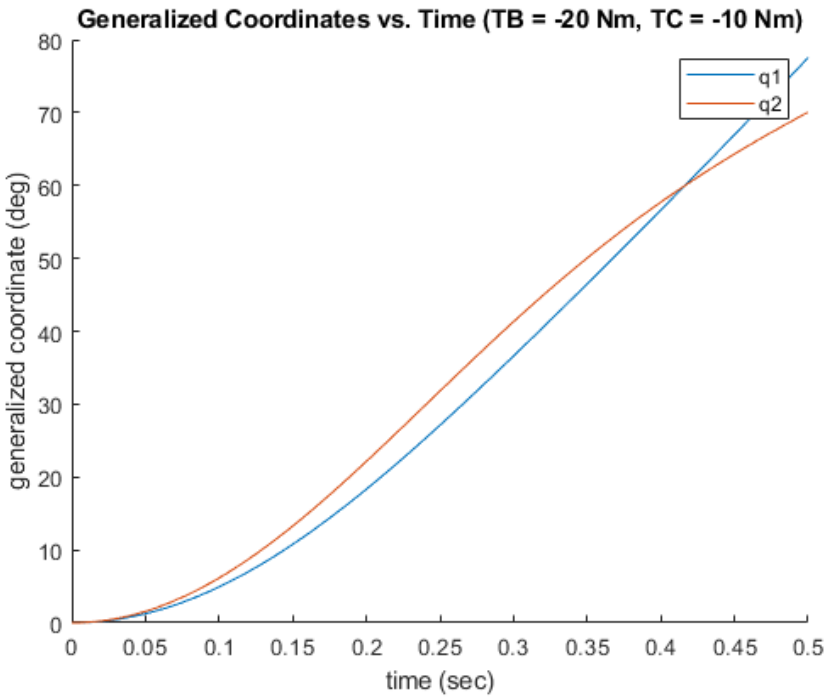

Figure 11 Generalized Coordinates VS. Time for TB = -20 Nm, TC = -10 Nm

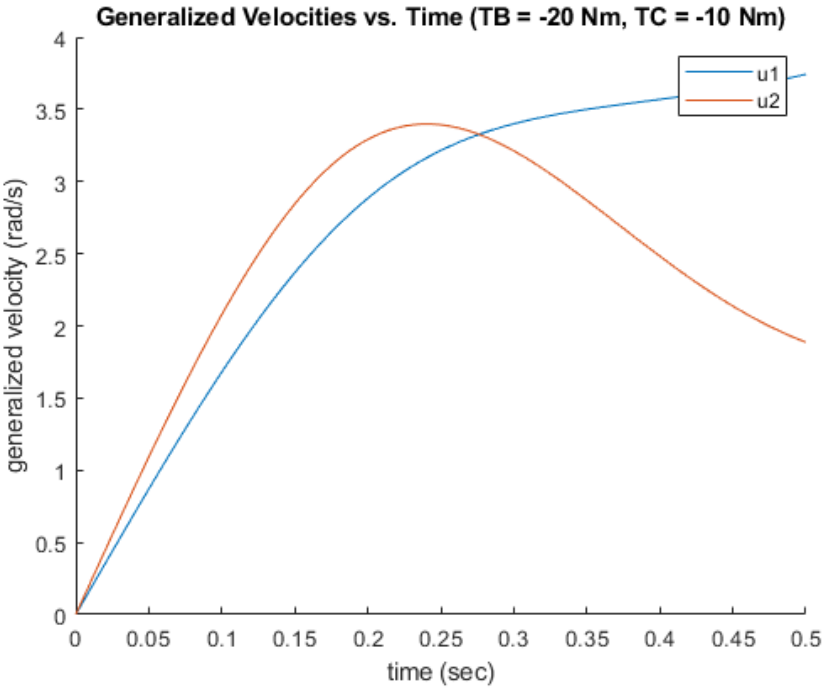

Figure 12 Generalized Velocities VS. Time for TB = -20 Nm, TC = -10 Nm

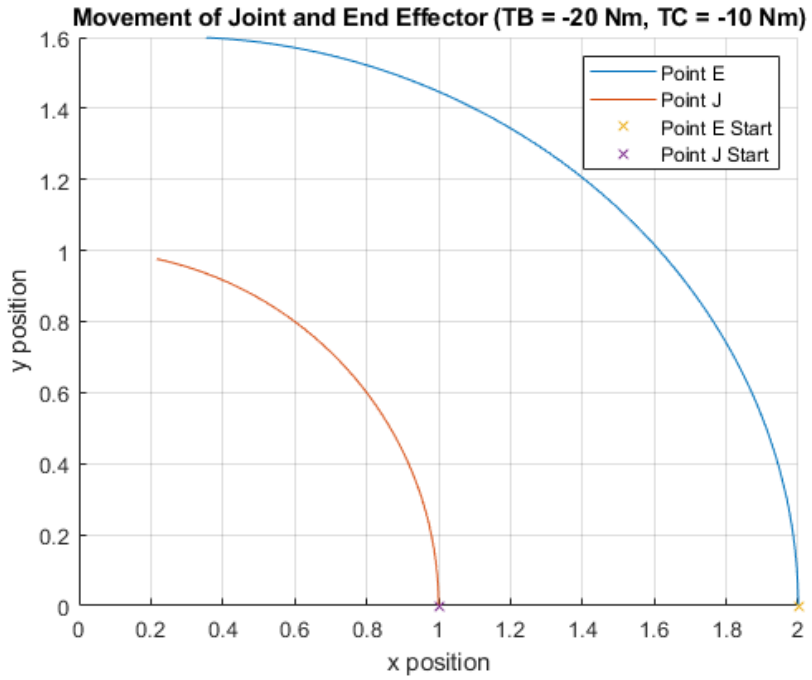

Figure 13 Figure of Joint Movement and End Effector for TB = -20 Nm, TC = -10 Nm

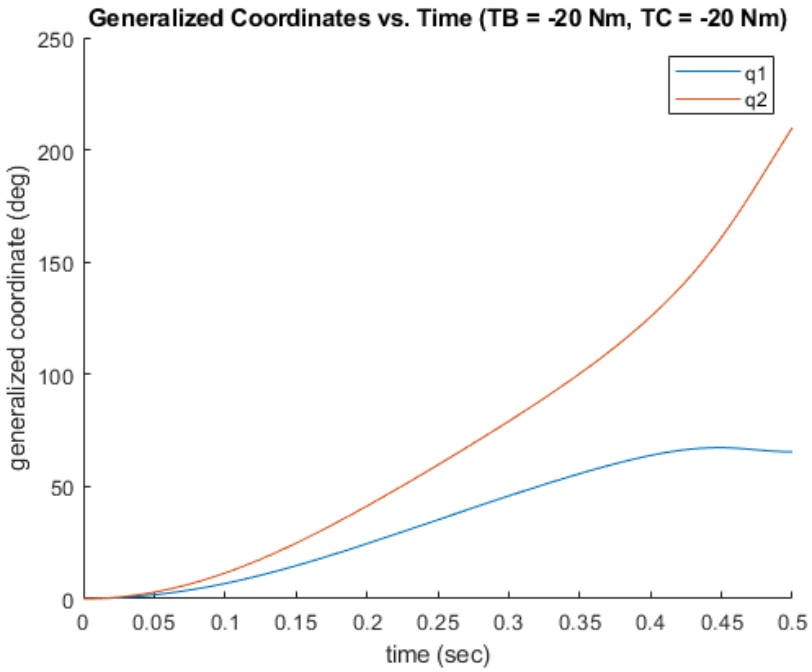

Figure 14 Generalized Coordinates VS. Time for TB = -20 Nm, TC = -20 Nm

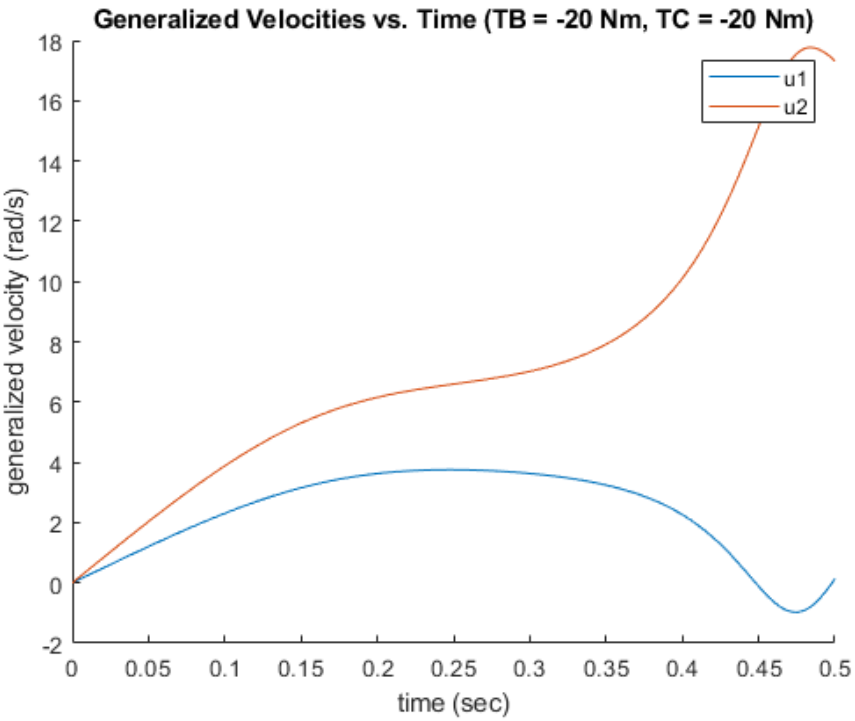

Figure 15 Generalized Velocities VS. Time for TB = -20 Nm, TC = -20 Nm

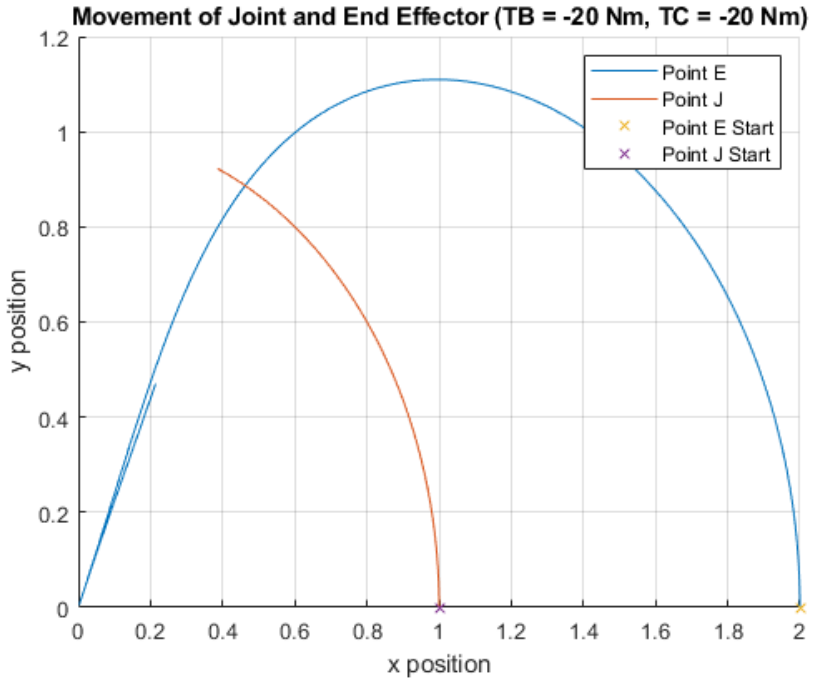

Figure 16 Figure of Joint Movement and End Effector for TB = -20 Nm, TC = -20 Nm

Discussion

One noticeable trend in the results is how the curves for the generalized coordinates and velocities change as the torques increase. In all cases, the coordinate u2 and velocity q2 generally increase over time, indicating an acceleration, which would be caused by the applied torques. Interestingly, u_2's increase isn't as steady and shows deceleration at the end of the simulation period. This deceleration comes a lot sooner when $T_B < T_C$. Also notable is when $T_C = -20\ \mathrm{Nm}$, u_1 drops below 0 near the end of the simulation before demonstrating acceleration.

These phenomena can be explained by looking at **Figures 7, 10, 13, and 16**, which show the movement of the joint and end effector for each of the four trials. In both cases when $T_C = -20\ \mathrm{Nm}$, link B is rotated all the way past 180 degrees, even swinging back up whilst link C does not rotate very far past 90 degrees. The movement of link B in these trials is responsible for that deceleration demonstrated by the curves of u_1. Similar phenomena would likely be present in the trials where $T_C = -10\ \mathrm{Nm}$ had the simulations gone on for a little longer; the deceleration does begin but the trial ends before DOF 2 can begin accelerating again.

Conclusion

In this comprehensive study, the kinematic characteristics of a simplified 2D two-arm robot manipulator were explored. Our focus was on a basic yet insightful model consisting of two linked bars and a non-rotating gripper, aimed at understanding the fundamental operational dynamics of robotic arms.

Through kinematic analysis, the equations of motion were established, and Autolev was used to verify the accuracy of these equations. This approach allowed us to precisely dissect the motion capabilities and limitations of the robot arm, providing valuable insights for its application in automated systems.

These findings demonstrate that even in a simplified model, the complexity of robotic motion is evident. The linear and angular movements within the arm reveal a range of motion possibilities and constraints. This understanding is vital for advancing the design and functionality of more complex robotic systems in industrial and research settings.

Despite the simplifications, this project provides a deeper understanding of the kinematic behavior of robotic arms and lays a foundation for further exploration in the realm of robotics. The insights gained from this study could be instrumental in designing more efficient, accurate, and versatile robotic manipulators for various applications. The simplification of the model does not detract from the relevance of the results; rather, it provides insights useful for future work. Such future work would likely consist of having the simulated arm attempt to follow a pre-defined path as closely as possible; while the insights provided by the simulation were valuable, the simulation itself is not very representative of how robot arms function in the real world.

References

[1] Kelechava, B. (2019, January 4). *Industrial Robots Make Production Safe and Efficient*. The ANSI Blog. https://blog.ansi.org/2019/01/industrial-robots-safety-production-ria/

[2] Sabri, M., Fauzi, R., Fajar, M. S., Geubrina, H. S., & Sabri, F. A. M. (2021). Model and simulation of arm robot with 5 degrees of freedom using MATLAB. IOP Conference Series.Materials Science and Engineering, 1122(1) doi:https://doi.org/10.1088/1757-899X/1122/1/012032

[3] Luan, Y., Guo, J., & Liu, H. (2021). Structural dynamics simulation analysis of industrial robot arm based on kane method. Journal of Physics: Conference Series, 1871(1) doi:https://doi.org/10.1088/1742-6596/1871/1/012155

[4] Mahil, S. M., & Al-Durra, A. (2016). Modeling analysis and simulation of 2-DOF robotic manipulator. *2016 IEEE 59th International Midwest Symposium on Circuits and Systems (MWSCAS)*, 1–4. https://doi.org/10.1109/MWSCAS.2016.7870099

[5] Teng Fong, T., Jamaludin, Z., Bani Hashim, A. Y., & Rahman, M. A. A. (2015). Design and Analysis of Linear Quadratic Regulator for a Non-Linear Positioning System. *Applied Mechanics and Materials*, *761*, 227–232. https://doi.org/10.4028/www.scientific.net/AMM.761.227

[6] He, W., Ouyang, Y., & Hong, J. (2017). Vibration Control of a Flexible Robotic Manipulator in the Presence of Input Deadzone. *IEEE Transactions on Industrial Informatics*, *13*(1), 48–59. https://doi.org/10.1109/TII.2016.2608739.

[7] Zhang, L., & Liu, J. (2012). Observer-based partial differential equation boundary control for a flexible two-link manipulator in task space. *IET Control Theory & Applications*, *6*(13), 2120–2133. https://doi.org/10.1049/iet-cta.2011.0545

Appendix

Appendix of Autolev code

```
% PROJECT2.AL
NEWTONIAN A
Bodies B,C
CONSTANTS g, LA, LB,TB,TC
POINTS O, J, E


VARIABLES q1',q2'
MOTIONVARIABLES' u1',u2'
q1' = u1; q2' = u2


SIMPROT(B,A,3,q1)
```

```
SIMPROT(C,B,3,q2)

w_A> = u1*b3>
w_B> = (u1+u2)*c3>
Alpha_A> = u1'*b3>
Alpha_B> = (u1'+u2')*c3>

R_OJ> = LA*B2>
R_JE> = LB*C2>
V_J_A>= CROSS(w_A>,R_OJ>)
V_E_A> = V_J_A> + CROSS(w_B>,R_JE>)

EXPRESS(V_J_A>,A)
EXPRESS(V_E_A>,A)


A_J_A> = CROSS(w_A>,CROSS(w_A>,R_OJ>))+CROSS(Alpha_A>,R_OJ>)
A_E_A> = A_J_A> + CROSS(w_B>,CROSS(w_B>,R_JE>))+CROSS(Alpha_B>,R_JE>)

EXPRESS(A_J_A>,A)
EXPRESS(A_E_A>,A)

Mass B=mlB
Mass C=mlC

force(J,-mlB*g*a2>)
force(E,-mlC*g*a2>)

%Define Inertia Tensor
Inertia B, (1/3)*mlB*LA^2, 0, (1/3)*mlB*LA^2
Inertia C, (1/3)*mlB*LB^2, 0, (1/3)*mlB*LB^2

%Define Torque
Torque(B,b3>*TB)
Torque(C,c3>*TC)

V_O_A> = 0>
P_O_BO> = 1/2*LA*B2>
V_J_B> = EXPRESS(V_J_A>,B)
P_J_CO> = 1/2*LB*C2>

v2pts(A,B,O,BO)
v2pts(B,C,J,CO)
V_CO_A> = EXPRESS(V_CO_B>,A)

% Accelerations
A_O_A> = 0>
A_J_B> = EXPRESS(A_J_A>,B)
a2pts(A,B,O,BO)
a2pts(B,C,J,CO)
A_CO_A> = EXPRESS(A_CO_B>,A)


zero =fr()+frstar()
```

```
kane()

R_OE> = R_OJ> + R_JE>
EXPRESS(R_OE>,A)

mag = mag(R_OE>)
R_OE_x = cos(q1)*mag(R_OE>)
R_OE_y = sin(q1)*mag(R_OE>)


% Set units of each component
units g=m/sec^2,la=m, lb=m, mlB=kg, mlC=kg,
t=sec,q1=deg,q2=deg,q1=rad/sec,u2=rad/sec, TB = N*m, TC = N*m

% Set the values of the constants
input la=1, lb=2, g=9.81, mlB=10, mlC=20

% Set the starting value
input q1=10, q2=10,u1=0,u2=0,TB=1,TC = 1

%set the general simulation factors
input tfinal=16, integstp=0.1, absErr=1.0E-07, relErr=1.0E-07

%output the value
output t, q1,q2,u1,u2,R_OE_x, R_OE_y

% save the code as matlab code
code dynamics() project2.m,subs
VARIABLES q1',q2'
MOTIONVARIABLES' u1',u2'
q1' = u1; q2' = u2


SIMPROT(B,A,3,q1)
SIMPROT(C,B,3,q2)

w_A> = u1*b3>
w_B> = (u1+u2)*c3>
Alpha_A> = u1'*b3>
Alpha_B> = (u1'+u2')*c3>

R_OJ> = LA*B2>
R_JE> = LB*C2>
V_J_A>= CROSS(w_A>,R_OJ>)
V_E_A> = V_J_A> + CROSS(w_B>,R_JE>)

EXPRESS(V_J_A>,A)
EXPRESS(V_E_A>,A)


A_J_A> = CROSS(w_A>,CROSS(w_A>,R_OJ>))+CROSS(Alpha_A>,R_OJ>)
A_E_A> = A_J_A> + CROSS(w_B>,CROSS(w_B>,R_JE>))+CROSS(Alpha_B>,R_JE>)

EXPRESS(A_J_A>,A)
```

```
EXPRESS(A_E_A>,A)

Mass B=mlB
Mass C=mlC

force(J,-mlB*g*a2>)
force(E,-mlC*g*a2>)

%Define Inertia Tensor
Inertia B, (1/3)*mlB*LA^2, 0, (1/3)*mlB*LA^2
Inertia C, (1/3)*mlB*LB^2, 0, (1/3)*mlB*LB^2

%Define Torque
Torque(B,b3>*TB)
Torque(C,c3>*TC)

V_O_A> = 0>
P_O_BO> = 1/2*LA*B2>
V_J_B> = EXPRESS(V_J_A>,B)
P_J_CO> = 1/2*LB*C2>

v2pts(A,B,O,BO)
v2pts(B,C,J,CO)
V_CO_A> = EXPRESS(V_CO_B>,A)

% Accelerations
A_O_A> = 0>
A_J_B> = EXPRESS(A_J_A>,B)
a2pts(A,B,O,BO)
a2pts(B,C,J,CO)
A_CO_A> = EXPRESS(A_CO_B>,A)


zero =fr()+frstar()
kane()

R_OE> = R_OJ> + R_JE>
EXPRESS(R_OE>,A)

mag = mag(R_OE>)
R_OE_x = cos(q1)*mag(R_OE>)
R_OE_y = sin(q1)*mag(R_OE>)


% Set units of each component
units g=m/sec^2,la=m, lb=m, mlB=kg, mlC=kg,
t=sec,q1=deg,q2=deg,q1=rad/sec,u2=rad/sec, TB = N*m, TC = N*m

% Set the values of the constants
input la=1, lb=2, g=9.81, mlB=10, mlC=20

% Set the starting value
input q1=10, q2=10,u1=0,u2=0,TB=1,TC = 1
```

```
%set the general simulation factors
input tfinal=16, integstp=0.1, absErr=1.0E-07, relErr=1.0E-07

%output the value
output t, q1,q2,u1,u2,R_OE_x, R_OE_y

% save the code as matlab code
code dynamics() project2.m,subs

SIMPROT(B,A,3,q1)
SIMPROT(C,B,3,q2)

w_A> = u1*b3>
w_B> = (u1+u2)*c3>
Alpha_A> = u1'*b3>
Alpha_B> = (u1'+u2')*c3>

R_OJ> = LA*B2>
R_JE> = LB*C2>
V_J_A>= CROSS(w_A>,R_OJ>)
V_E_A> = V_J_A> + CROSS(w_B>,R_JE>)

EXPRESS(V_J_A>,A)
EXPRESS(V_E_A>,A)


A_J_A> = CROSS(w_A>,CROSS(w_A>,R_OJ>))+CROSS(Alpha_A>,R_OJ>)
A_E_A> = A_J_A> + CROSS(w_B>,CROSS(w_B>,R_JE>))+CROSS(Alpha_B>,R_JE>)

EXPRESS(A_J_A>,A)
EXPRESS(A_E_A>,A)

Mass B=mlB
Mass C=mlC

force(J,-mlB*g*a2>)
force(E,-mlC*g*a2>)

%Define Inertia Tensor
Inertia B, (1/3)*mlB*LA^2, 0, (1/3)*mlB*LA^2
Inertia C, (1/3)*mlB*LB^2, 0, (1/3)*mlB*LB^2

%Define Torque
Torque(B,b3>*TB)
Torque(C,c3>*TC)

V_O_A> = 0>
P_O_BO> = 1/2*LA*B2>
V_J_B> = EXPRESS(V_J_A>,B)
P_J_CO> = 1/2*LB*C2>

v2pts(A,B,O,BO)
v2pts(B,C,J,CO)
V_CO_A> = EXPRESS(V_CO_B>,A)
```

```
% Accelerations
A_O_A> = 0>
A_J_B> = EXPRESS(A_J_A>,B)
a2pts(A,B,O,BO)
a2pts(B,C,J,CO)
A_CO_A> = EXPRESS(A_CO_B>,A)


zero =fr()+frstar()
kane()

R_OE> = R_OJ> + R_JE>
EXPRESS(R_OE>,A)

mag = mag(R_OE>)
R_OE_x = cos(q1)*mag(R_OE>)
R_OE_y = sin(q1)*mag(R_OE>)


% Set units of each component
units g=m/sec^2,la=m, lb=m, mlB=kg, mlC=kg,
t=sec,q1=deg,q2=deg,q1=rad/sec,u2=rad/sec, TB = N*m, TC = N*m

% Set the values of the constants
input la=1, lb=2, g=9.81, mlB=10, mlC=20

% Set the starting value
input q1=10, q2=10,u1=0,u2=0,TB=1,TC = 1

%set the general simulation factors
input tfinal=16, integstp=0.1, absErr=1.0E-07, relErr=1.0E-07

%output the value
output t, q1,q2,u1,u2,R_OE_x, R_OE_y

% save the code as matlab code
code dynamics() project2.m,subs
```

Appendix of Matlab code

```
function project2
SolveOrdinaryDifferentialEquations
% File  project2.m  created by Autolev 4.1 on Sat Dec 09 12:35:57 2023


%=============================================================================
==
function VAR = ReadUserInput
global   g LA LB mlB mlC TB TC;
global   q1 q2 u1 u2;
global   q1p q2p u1p u2p R_OE_x R_OE_y;
global   DEGtoRAD RADtoDEG COEF RHS SolutionToAxEqualsB;
global   TINITIAL TFINAL INTEGSTP PRINTINT ABSERR RELERR;

%-------------------------------+--------------------------+--------------
-----+-----------------
% Quantity                     | Value                    | Units
| Description
%-------------------------------|--------------------------|--------------
-----|-----------------
g                              =  9.81;                   % m/sec^2
Constant
LA                             =  1;                      % m
Constant
LB                             =  2;                      % m
Constant
mlB                            =  10;                     % kg
Constant
mlC                            =  20;                     % kg
Constant
TB                             =  1;                      % N*m
Constant
TC                             =  1;                      % N*m
Constant

q1                             =  10;                     % rad/sec
Initial Value
q2                             =  10;                     % deg
Initial Value
u1                             =  0;                      % UNITS
Initial Value
u2                             =  0;                      % rad/sec
Initial Value

TINITIAL                       =  0.0;                    % UNITS
Initial Time
TFINAL                         =  16;                     % UNITS
Final Time
INTEGSTP                       =  0.1;                    % UNITS
Integration Step
PRINTINT                       =  1;                      % Positive
Integer    Print-Integer
```

```
ABSERR                          =  1.0E-07;               %
Absolute Error
RELERR                          =  1.0E-07;               %
Relative Error
%-------------------------------+-------------------------+-------------
-----+-----------------

% Unit conversions
Pi       = 3.141592653589793;
DEGtoRAD = Pi/180.0;
RADtoDEG = 180.0/Pi;
q2 = q2*DEGtoRAD;

% Reserve space and initialize matrices
COEF = zeros(2,2);
RHS = zeros(1,2);

% Evaluate constants
% Set the initial values of the states
VAR(1) = q1;
VAR(2) = q2;
VAR(3) = u1;
VAR(4) = u2;



%=====================================================================
==
function OpenOutputFilesAndWriteHeadings
FileIdentifier = fopen('project2.1', 'wt');   if( FileIdentifier == -1 )
error('Error: unable to open file project2.1'); end
fprintf( 1,             '%%       t             q1             q2
u1             u2           R_OE_x         R_OE_y\n' );
fprintf( 1,             '%%     (sec)         (rad/sec)        (deg)
(UNITS)       (rad/sec)       (UNITS)        (UNITS)\n\n' );
fprintf(FileIdentifier, '%% FILE: project2.1\n%%\n' );
fprintf(FileIdentifier, '%%       t             q1             q2
u1             u2           R_OE_x         R_OE_y\n' );
fprintf(FileIdentifier, '%%     (sec)         (rad/sec)        (deg)
(UNITS)       (rad/sec)       (UNITS)        (UNITS)\n\n' );



%=====================================================================
==
% Main driver loop for numerical integration of differential equations
%=====================================================================
==
function SolveOrdinaryDifferentialEquations
global   g LA LB mlB mlC TB TC;
global   q1 q2 u1 u2;
global   q1p q2p u1p u2p R_OE_x R_OE_y;
global   DEGtoRAD RADtoDEG COEF RHS SolutionToAxEqualsB;
global   TINITIAL TFINAL INTEGSTP PRINTINT ABSERR RELERR;
```

```
OpenOutputFilesAndWriteHeadings
VAR = ReadUserInput;
OdeMatlabOptions = odeset( 'RelTol',RELERR, 'AbsTol',ABSERR,
'MaxStep',INTEGSTP );
T = TINITIAL;
PrintCounter = 0;
mdlDerivatives(T,VAR,0);
while 1,
  if( TFINAL>=TINITIAL & T+0.01*INTEGSTP>=TFINAL ) PrintCounter = -1; end
  if( TFINAL<=TINITIAL & T+0.01*INTEGSTP<=TFINAL ) PrintCounter = -1; end
  if( PrintCounter <= 0.01 ),
     mdlOutputs(T,VAR,0);
     if( PrintCounter == -1 ) break; end
     PrintCounter = PRINTINT;
  end
  [TimeOdeArray,VarOdeArray] = ode45( @mdlDerivatives, [T T+INTEGSTP],
VAR, OdeMatlabOptions, 0 );
  TimeAtEndOfArray = TimeOdeArray( length(TimeOdeArray) );
  if( abs(TimeAtEndOfArray - (T+INTEGSTP) ) >= abs(0.001*INTEGSTP) )
warning('numerical integration failed'); break;  end
  T = TimeAtEndOfArray;
  VAR = VarOdeArray( length(TimeOdeArray), : );
  PrintCounter = PrintCounter - 1;
end
mdlTerminate(T,VAR,0);



%=========================================================================
==
% mdlDerivatives: Calculates and returns the derivatives of the continuous
states
%=========================================================================
==
function sys = mdlDerivatives(T,VAR,u)
global   g LA LB mlB mlC TB TC;
global   q1 q2 u1 u2;
global   q1p q2p u1p u2p R_OE_x R_OE_y;
global   DEGtoRAD RADtoDEG COEF RHS SolutionToAxEqualsB;
global   TINITIAL TFINAL INTEGSTP PRINTINT ABSERR RELERR;

% Update variables after integration step
q1 = VAR(1);
q2 = VAR(2);
u1 = VAR(3);
u2 = VAR(4);
q1p = u1;
q2p = u2;

COEF(1,1) = -mlC*LA^2 - 0.5833333333333333*mlB*LA^2 -
0.3333333333333333*mlB*LB^2;
COEF(1,2) = -0.3333333333333333*LB*(LB*mlB-1.5*LA*mlC*cos(q2));
COEF(2,1) = -0.3333333333333333*LB*(LB*mlB-1.5*LA*mlC*cos(q2));
```

```
COEF(2,2) = -0.25*LB^2*(mlC+1.333333333333333*mlB);
RHS(1) = TB + TC + g*LA*mlB*sin(q1) + g*mlC*(LA*sin(q1)+LB*sin(q1+q2)) +
0.5*LA*LB*mlC*sin(q2)*u2^2;
RHS(2) = TC + g*LB*mlC*sin(q1+q2) + 0.5*LA*LB*mlC*sin(q2)*u1^2;
VARp(1) = COEF(1,1)*COEF(2,2) - COEF(1,2)*COEF(2,1);
SolutionToAxEqualsB(2) =(COEF(1,1)*RHS(2)-COEF(2,1)*RHS(1))/VARp(1);
SolutionToAxEqualsB(1) =(COEF(2,2)*RHS(1)-COEF(1,2)*RHS(2))/VARp(1);

% Update variables after uncoupling equations
u1p = SolutionToAxEqualsB(1);
u2p = SolutionToAxEqualsB(2);

% Update derivative array prior to integration step
VARp(1) = q1p;
VARp(2) = q2p;
VARp(3) = u1p;
VARp(4) = u2p;

sys = VARp';



%=========================================================================
==
% mdlOutputs: Calculates and return the outputs
%=========================================================================
==
function Output = mdlOutputs(T,VAR,u)
global   g LA LB mlB mlC TB TC;
global   q1 q2 u1 u2;
global   q1p q2p u1p u2p R_OE_x R_OE_y;
global   DEGtoRAD RADtoDEG COEF RHS SolutionToAxEqualsB;
global   TINITIAL TFINAL INTEGSTP PRINTINT ABSERR RELERR;

% Evaluate output quantities
R_OE_x =
cos(q1)*((LA*sin(q1)+LB*sin(q1+q2))^2+(LA*cos(q1)+LB*cos(q1+q2))^2)^0.5;
R_OE_y =
sin(q1)*((LA*sin(q1)+LB*sin(q1+q2))^2+(LA*cos(q1)+LB*cos(q1+q2))^2)^0.5;

Output(1)=T;  Output(2)=q1;  Output(3)=(q2*RADtoDEG);  Output(4)=u1;
Output(5)=u2;  Output(6)=R_OE_x;  Output(7)=R_OE_y;
FileIdentifier = fopen('all');
WriteOutput( 1,                 Output(1:7) );
WriteOutput( FileIdentifier(1), Output(1:7) );



%=========================================================================
==
function WriteOutput( fileIdentifier, Output )
numberOfOutputQuantities = length( Output );
if numberOfOutputQuantities > 0,
  for i=1:numberOfOutputQuantities,
```

```
    fprintf( fileIdentifier, ' %- 14.6E', Output(i) );
  end
  fprintf( fileIdentifier, '\n' );
end



%=============================================================================
==
% mdlTerminate: Perform end of simulation tasks and set sys=[]
%=============================================================================
==
function sys = mdlTerminate(T,VAR,u)
FileIdentifier = fopen('all');
fclose( FileIdentifier(1) );
fprintf( 1, '\n Output is in the file project2.1\n' );
fprintf( 1, '\n To load and plot columns 1 and 2 with a solid line and
columns 1 and 3 with a dashed line, enter:\n' );
fprintf( 1, '    someName = load( ''project2.1'' );\n' );
fprintf( 1, '    plot( someName(:,1), someName(:,2), ''-'', someName(:,1),
someName(:,3), ''--'' )\n\n' );
sys = [];



%=============================================================================
==
% Sfunction: System/Simulink function from standard template
%=============================================================================
==
function [sys,x0,str,ts] = Sfunction(t,x,u,flag)
switch flag,
  case 0,  [sys,x0,str,ts] = mdlInitializeSizes;    % Initialization of
sys, initial state x0, state ordering string str, and sample times ts
  case 1,  sys = mdlDerivatives(t,x,u);             % Calculate the
derivatives of continuous states and store them in sys
  case 2,  sys = mdlUpdate(t,x,u);                  % Update discrete
states x(n+1) in sys
  case 3,  sys = mdlOutputs(t,x,u);                 % Calculate outputs in
sys
  case 4,  sys = mdlGetTimeOfNextVarHit(t,x,u);     % Return next sample
time for variable-step in sys
  case 9,  sys = mdlTerminate(t,x,u);               % Perform end of
simulation tasks and set sys=[]
  otherwise error(['Unhandled flag = ',num2str(flag)]);
end



%=============================================================================
==
% mdlInitializeSizes: Return the sizes, initial state VAR, and sample
times ts
```

```
%=====================================================================
==
function [sys,VAR,stateOrderingStrings,timeSampling] = mdlInitializeSizes
sizes = simsizes;             % Call simsizes to create a sizes structure
sizes.NumContStates  = 4;     % sys(1) is the number of continuous states
sizes.NumDiscStates  = 0;     % sys(2) is the number of discrete states
sizes.NumOutputs     = 7;     % sys(3) is the number of outputs
sizes.NumInputs      = 0;     % sys(4) is the number of inputs
sizes.DirFeedthrough = 1;     % sys(6) is 1, and allows for the output to
be a function of the input
sizes.NumSampleTimes = 1;     % sys(7) is the number of samples times (the
number of rows in ts)
sys = simsizes(sizes);        % Convert it to a sizes array
stateOrderingStrings = [];
timeSampling         = [0 0]; % m-by-2 matrix containing the sample times
OpenOutputFilesAndWriteHeadings
VAR = ReadUserInput
SolveOrdinaryDifferentialEquations
% File  project2.m  created by Autolev 4.1 on Sat Dec 09 12:35:57 2023


%=====================================================================
==
function VAR = ReadUserInput
global   g LA LB mlB mlC TB TC;
global   q1 q2 u1 u2;
global   q1p q2p u1p u2p R_OE_x R_OE_y;
global   DEGtoRAD RADtoDEG COEF RHS SolutionToAxEqualsB;
global   TINITIAL TFINAL INTEGSTP PRINTINT ABSERR RELERR;

%-------------------------------+--------------------------+-------------
-----+-----------------
% Quantity                     | Value                    | Units
| Description
%-------------------------------|--------------------------|-------------
-----|-----------------
g                              =  9.81;                   % m/sec^2
Constant
LA                             =  1;                      % m
Constant
LB                             =  2;                      % m
Constant
mlB                            =  10;                     % kg
Constant
mlC                            =  20;                     % kg
Constant
TB                             =  1;                      % N*m
Constant
TC                             =  1;                      % N*m
Constant

q1                             =  10;                     % rad/sec
Initial Value
```

```
q2                              =  10;                    % deg
Initial Value
u1                              =  0;                     % UNITS
Initial Value
u2                              =  0;                     % rad/sec
Initial Value

TINITIAL                        =  0.0;                   % UNITS
Initial Time
TFINAL                          =  16;                    % UNITS
Final Time
INTEGSTP                        =  0.1;                   % UNITS
Integration Step
PRINTINT                        =  1;                     % Positive
Integer    Print-Integer
ABSERR                          =  1.0E-07;               %
Absolute Error
RELERR                          =  1.0E-07;               %
Relative Error
%-------------------------------+-------------------------+-------------
-----+-----------------

% Unit conversions
Pi       = 3.141592653589793;
DEGtoRAD = Pi/180.0;
RADtoDEG = 180.0/Pi;
q2 = q2*DEGtoRAD;

% Reserve space and initialize matrices
COEF = zeros(2,2);
RHS = zeros(1,2);

% Evaluate constants
% Set the initial values of the states
VAR(1) = q1;
VAR(2) = q2;
VAR(3) = u1;
VAR(4) = u2;



%=========================================================================
==
function OpenOutputFilesAndWriteHeadings
FileIdentifier = fopen('project2.1', 'wt');   if( FileIdentifier == -1 )
error('Error: unable to open file project2.1'); end
fprintf( 1,             '%%       t             q1             q2
u1             u2           R_OE_x         R_OE_y\n' );
fprintf( 1,             '%%     (sec)         (rad/sec)        (deg)
(UNITS)       (rad/sec)       (UNITS)         (UNITS)\n\n' );
fprintf(FileIdentifier, '%% FILE: project2.1\n%%\n' );
fprintf(FileIdentifier, '%%       t             q1             q2
u1             u2           R_OE_x         R_OE_y\n' );
```

```
fprintf(FileIdentifier, '%%     (sec)        (rad/sec)        (deg)
(UNITS)       (rad/sec)       (UNITS)        (UNITS)\n\n' );



%=========================================================================
==
% Main driver loop for numerical integration of differential equations
%=========================================================================
==
function SolveOrdinaryDifferentialEquations
global   g LA LB mlB mlC TB TC;
global   q1 q2 u1 u2;
global   q1p q2p u1p u2p R_OE_x R_OE_y;
global   DEGtoRAD RADtoDEG COEF RHS SolutionToAxEqualsB;
global   TINITIAL TFINAL INTEGSTP PRINTINT ABSERR RELERR;

OpenOutputFilesAndWriteHeadings
VAR = ReadUserInput;
OdeMatlabOptions = odeset( 'RelTol',RELERR, 'AbsTol',ABSERR,
'MaxStep',INTEGSTP );
T = TINITIAL;
PrintCounter = 0;
mdlDerivatives(T,VAR,0);
while 1,
  if( TFINAL>=TINITIAL & T+0.01*INTEGSTP>=TFINAL ) PrintCounter = -1; end
  if( TFINAL<=TINITIAL & T+0.01*INTEGSTP<=TFINAL ) PrintCounter = -1; end
  if( PrintCounter <= 0.01 ),
     mdlOutputs(T,VAR,0);
     if( PrintCounter == -1 ) break; end
     PrintCounter = PRINTINT;
  end
  [TimeOdeArray,VarOdeArray] = ode45( @mdlDerivatives, [T T+INTEGSTP],
VAR, OdeMatlabOptions, 0 );
  TimeAtEndOfArray = TimeOdeArray( length(TimeOdeArray) );
  if( abs(TimeAtEndOfArray - (T+INTEGSTP) ) >= abs(0.001*INTEGSTP) )
warning('numerical integration failed'); break;  end
  T = TimeAtEndOfArray;
  VAR = VarOdeArray( length(TimeOdeArray), : );
  PrintCounter = PrintCounter - 1;
end
mdlTerminate(T,VAR,0);



%=========================================================================
==
% mdlDerivatives: Calculates and returns the derivatives of the continuous
states
%=========================================================================
==
function sys = mdlDerivatives(T,VAR,u)
global   g LA LB mlB mlC TB TC;
global   q1 q2 u1 u2;
```

```
global   q1p q2p u1p u2p R_OE_x R_OE_y;
global   DEGtoRAD RADtoDEG COEF RHS SolutionToAxEqualsB;
global   TINITIAL TFINAL INTEGSTP PRINTINT ABSERR RELERR;

% Update variables after integration step
q1 = VAR(1);
q2 = VAR(2);
u1 = VAR(3);
u2 = VAR(4);
q1p = u1;
q2p = u2;

COEF(1,1) = -mlC*LA^2 - 0.5833333333333333*mlB*LA^2 -
0.3333333333333333*mlB*LB^2;
COEF(1,2) = -0.3333333333333333*LB*(LB*mlB-1.5*LA*mlC*cos(q2));
COEF(2,1) = -0.3333333333333333*LB*(LB*mlB-1.5*LA*mlC*cos(q2));
COEF(2,2) = -0.25*LB^2*(mlC+1.333333333333333*mlB);
RHS(1) = TB + TC + g*LA*mlB*sin(q1) + g*mlC*(LA*sin(q1)+LB*sin(q1+q2)) +
0.5*LA*LB*mlC*sin(q2)*u2^2;
RHS(2) = TC + g*LB*mlC*sin(q1+q2) + 0.5*LA*LB*mlC*sin(q2)*u1^2;
VARp(1) = COEF(1,1)*COEF(2,2) - COEF(1,2)*COEF(2,1);
SolutionToAxEqualsB(2) =(COEF(1,1)*RHS(2)-COEF(2,1)*RHS(1))/VARp(1);
SolutionToAxEqualsB(1) =(COEF(2,2)*RHS(1)-COEF(1,2)*RHS(2))/VARp(1);

% Update variables after uncoupling equations
u1p = SolutionToAxEqualsB(1);
u2p = SolutionToAxEqualsB(2);

% Update derivative array prior to integration step
VARp(1) = q1p;
VARp(2) = q2p;
VARp(3) = u1p;
VARp(4) = u2p;

sys = VARp';



%=========================================================================
==
% mdlOutputs: Calculates and return the outputs
%=========================================================================
==
function Output = mdlOutputs(T,VAR,u)
global   g LA LB mlB mlC TB TC;
global   q1 q2 u1 u2;
global   q1p q2p u1p u2p R_OE_x R_OE_y;
global   DEGtoRAD RADtoDEG COEF RHS SolutionToAxEqualsB;
global   TINITIAL TFINAL INTEGSTP PRINTINT ABSERR RELERR;

% Evaluate output quantities
R_OE_x =
cos(q1)*((LA*sin(q1)+LB*sin(q1+q2))^2+(LA*cos(q1)+LB*cos(q1+q2))^2)^0.5;
```

```
R_OE_y =
sin(q1)*((LA*sin(q1)+LB*sin(q1+q2))^2+(LA*cos(q1)+LB*cos(q1+q2))^2)^0.5;

Output(1)=T;  Output(2)=q1;  Output(3)=(q2*RADtoDEG);  Output(4)=u1;
Output(5)=u2;  Output(6)=R_OE_x;  Output(7)=R_OE_y;
FileIdentifier = fopen('all');
WriteOutput( 1,                 Output(1:7) );
WriteOutput( FileIdentifier(1), Output(1:7) );



%=========================================================================
==
function WriteOutput( fileIdentifier, Output )
numberOfOutputQuantities = length( Output );
if numberOfOutputQuantities > 0,
  for i=1:numberOfOutputQuantities,
    fprintf( fileIdentifier, ' %- 14.6E', Output(i) );
  end
  fprintf( fileIdentifier, '\n' );
end



%=========================================================================
==
% mdlTerminate: Perform end of simulation tasks and set sys=[]
%=========================================================================
==
function sys = mdlTerminate(T,VAR,u)
FileIdentifier = fopen('all');
fclose( FileIdentifier(1) );
fprintf( 1, '\n Output is in the file project2.1\n' );
fprintf( 1, '\n To load and plot columns 1 and 2 with a solid line and
columns 1 and 3 with a dashed line, enter:\n' );
fprintf( 1, '    someName = load( ''project2.1'' );\n' );
fprintf( 1, '    plot( someName(:,1), someName(:,2), ''-'', someName(:,1),
someName(:,3), ''--'' )\n\n' );
sys = [];



%=========================================================================
==
% Sfunction: System/Simulink function from standard template
%=========================================================================
==
function [sys,x0,str,ts] = Sfunction(t,x,u,flag)
switch flag,
  case 0,  [sys,x0,str,ts] = mdlInitializeSizes;    % Initialization of
sys, initial state x0, state ordering string str, and sample times ts
  case 1,  sys = mdlDerivatives(t,x,u);             % Calculate the
derivatives of continuous states and store them in sys
```

```
  case 2,  sys = mdlUpdate(t,x,u);                  % Update discrete
states x(n+1) in sys
  case 3,  sys = mdlOutputs(t,x,u);                 % Calculate outputs in
sys
  case 4,  sys = mdlGetTimeOfNextVarHit(t,x,u);     % Return next sample
time for variable-step in sys
  case 9,  sys = mdlTerminate(t,x,u);               % Perform end of
simulation tasks and set sys=[]
  otherwise error(['Unhandled flag = ',num2str(flag)]);
end



%=============================================================================
==
% mdlInitializeSizes: Return the sizes, initial state VAR, and sample
times ts
%=============================================================================
==
function [sys,VAR,stateOrderingStrings,timeSampling] = mdlInitializeSizes
sizes = simsizes;             % Call simsizes to create a sizes structure
sizes.NumContStates  = 4;     % sys(1) is the number of continuous states
sizes.NumDiscStates  = 0;     % sys(2) is the number of discrete states
sizes.NumOutputs     = 7;     % sys(3) is the number of outputs
sizes.NumInputs      = 0;     % sys(4) is the number of inputs
sizes.DirFeedthrough = 1;     % sys(6) is 1, and allows for the output to
be a function of the input
sizes.NumSampleTimes = 1;     % sys(7) is the number of samples times (the
number of rows in ts)
sys = simsizes(sizes);        % Convert it to a sizes array
stateOrderingStrings = [];
timeSampling         = [0 0]; % m-by-2 matrix containing the sample times
OpenOutputFilesAndWriteHeadings
VAR = ReadUserInput
SolveOrdinaryDifferentialEquations
% File  project2.m  created by Autolev 4.1 on Sat Dec 09 12:35:57 2023


%=============================================================================
==
function VAR = ReadUserInput
global   g LA LB mlB mlC TB TC;
global   q1 q2 u1 u2;
global   q1p q2p u1p u2p R_OE_x R_OE_y;
global   DEGtoRAD RADtoDEG COEF RHS SolutionToAxEqualsB;
global   TINITIAL TFINAL INTEGSTP PRINTINT ABSERR RELERR;

%-------------------------------+--------------------------+--------------
-----+-----------------
% Quantity                      | Value                    | Units
| Description
%-------------------------------|--------------------------|--------------
-----|-----------------
```

```
g                               =  9.81;                  % m/sec^2
Constant
LA                              =  1;                     % m
Constant
LB                              =  2;                     % m
Constant
mlB                             =  10;                    % kg
Constant
mlC                             =  20;                    % kg
Constant
TB                              =  1;                     % N*m
Constant
TC                              =  1;                     % N*m
Constant

q1                              =  10;                    % rad/sec
Initial Value
q2                              =  10;                    % deg
Initial Value
u1                              =  0;                     % UNITS
Initial Value
u2                              =  0;                     % rad/sec
Initial Value

TINITIAL                        =  0.0;                   % UNITS
Initial Time
TFINAL                          =  16;                    % UNITS
Final Time
INTEGSTP                        =  0.1;                   % UNITS
Integration Step
PRINTINT                        =  1;                     % Positive
Integer    Print-Integer
ABSERR                          =  1.0E-07;               %
Absolute Error
RELERR                          =  1.0E-07;               %
Relative Error
%-------------------------------+-------------------------+--------------
-----+-----------------

% Unit conversions
Pi       = 3.141592653589793;
DEGtoRAD = Pi/180.0;
RADtoDEG = 180.0/Pi;
q2 = q2*DEGtoRAD;

% Reserve space and initialize matrices
COEF = zeros(2,2);
RHS = zeros(1,2);

% Evaluate constants
% Set the initial values of the states
VAR(1) = q1;
VAR(2) = q2;
VAR(3) = u1;
```

```
VAR(4) = u2;



%=========================================================================
==
function OpenOutputFilesAndWriteHeadings
FileIdentifier = fopen('project2.1', 'wt');   if( FileIdentifier == -1 )
error('Error: unable to open file project2.1'); end
fprintf( 1,             '%%       t             q1             q2
u1             u2           R_OE_x         R_OE_y\n' );
fprintf( 1,             '%%     (sec)         (rad/sec)        (deg)
(UNITS)       (rad/sec)       (UNITS)         (UNITS)\n\n' );
fprintf(FileIdentifier, '%% FILE: project2.1\n%%\n' );
fprintf(FileIdentifier, '%%       t             q1             q2
u1             u2           R_OE_x         R_OE_y\n' );
fprintf(FileIdentifier, '%%     (sec)         (rad/sec)        (deg)
(UNITS)       (rad/sec)       (UNITS)         (UNITS)\n\n' );



%=========================================================================
==
% Main driver loop for numerical integration of differential equations
%=========================================================================
==
function SolveOrdinaryDifferentialEquations
global   g LA LB mlB mlC TB TC;
global   q1 q2 u1 u2;
global   q1p q2p u1p u2p R_OE_x R_OE_y;
global   DEGtoRAD RADtoDEG COEF RHS SolutionToAxEqualsB;
global   TINITIAL TFINAL INTEGSTP PRINTINT ABSERR RELERR;

OpenOutputFilesAndWriteHeadings
VAR = ReadUserInput;
OdeMatlabOptions = odeset( 'RelTol',RELERR, 'AbsTol',ABSERR,
'MaxStep',INTEGSTP );
T = TINITIAL;
PrintCounter = 0;
mdlDerivatives(T,VAR,0);
while 1,
  if( TFINAL>=TINITIAL & T+0.01*INTEGSTP>=TFINAL ) PrintCounter = -1; end
  if( TFINAL<=TINITIAL & T+0.01*INTEGSTP<=TFINAL ) PrintCounter = -1; end
  if( PrintCounter <= 0.01 ),
     mdlOutputs(T,VAR,0);
     if( PrintCounter == -1 ) break; end
     PrintCounter = PRINTINT;
  end
  [TimeOdeArray,VarOdeArray] = ode45( @mdlDerivatives, [T T+INTEGSTP],
VAR, OdeMatlabOptions, 0 );
  TimeAtEndOfArray = TimeOdeArray( length(TimeOdeArray) );
  if( abs(TimeAtEndOfArray - (T+INTEGSTP) ) >= abs(0.001*INTEGSTP) )
warning('numerical integration failed'); break;  end
  T = TimeAtEndOfArray;
```

```
  VAR = VarOdeArray( length(TimeOdeArray), : );
  PrintCounter = PrintCounter - 1;
end
mdlTerminate(T,VAR,0);



%=============================================================================
==
% mdlDerivatives: Calculates and returns the derivatives of the continuous
states
%=============================================================================
==
function sys = mdlDerivatives(T,VAR,u)
global   g LA LB mlB mlC TB TC;
global   q1 q2 u1 u2;
global   q1p q2p u1p u2p R_OE_x R_OE_y;
global   DEGtoRAD RADtoDEG COEF RHS SolutionToAxEqualsB;
global   TINITIAL TFINAL INTEGSTP PRINTINT ABSERR RELERR;

% Update variables after integration step
q1 = VAR(1);
q2 = VAR(2);
u1 = VAR(3);
u2 = VAR(4);
q1p = u1;
q2p = u2;

COEF(1,1) = -mlC*LA^2 - 0.5833333333333333*mlB*LA^2 -
0.3333333333333333*mlB*LB^2;
COEF(1,2) = -0.3333333333333333*LB*(LB*mlB-1.5*LA*mlC*cos(q2));
COEF(2,1) = -0.3333333333333333*LB*(LB*mlB-1.5*LA*mlC*cos(q2));
COEF(2,2) = -0.25*LB^2*(mlC+1.333333333333333*mlB);
RHS(1) = TB + TC + g*LA*mlB*sin(q1) + g*mlC*(LA*sin(q1)+LB*sin(q1+q2)) +
0.5*LA*LB*mlC*sin(q2)*u2^2;
RHS(2) = TC + g*LB*mlC*sin(q1+q2) + 0.5*LA*LB*mlC*sin(q2)*u1^2;
VARp(1) = COEF(1,1)*COEF(2,2) - COEF(1,2)*COEF(2,1);
SolutionToAxEqualsB(2) =(COEF(1,1)*RHS(2)-COEF(2,1)*RHS(1))/VARp(1);
SolutionToAxEqualsB(1) =(COEF(2,2)*RHS(1)-COEF(1,2)*RHS(2))/VARp(1);

% Update variables after uncoupling equations
u1p = SolutionToAxEqualsB(1);
u2p = SolutionToAxEqualsB(2);

% Update derivative array prior to integration step
VARp(1) = q1p;
VARp(2) = q2p;
VARp(3) = u1p;
VARp(4) = u2p;

sys = VARp';
```

```
%=========================================================================
==
% mdlOutputs: Calculates and return the outputs
%=========================================================================
==
function Output = mdlOutputs(T,VAR,u)
global   g LA LB mlB mlC TB TC;
global   q1 q2 u1 u2;
global   q1p q2p u1p u2p R_OE_x R_OE_y;
global   DEGtoRAD RADtoDEG COEF RHS SolutionToAxEqualsB;
global   TINITIAL TFINAL INTEGSTP PRINTINT ABSERR RELERR;

% Evaluate output quantities
R_OE_x =
cos(q1)*((LA*sin(q1)+LB*sin(q1+q2))^2+(LA*cos(q1)+LB*cos(q1+q2))^2)^0.5;
R_OE_y =
sin(q1)*((LA*sin(q1)+LB*sin(q1+q2))^2+(LA*cos(q1)+LB*cos(q1+q2))^2)^0.5;

Output(1)=T;  Output(2)=q1;  Output(3)=(q2*RADtoDEG);  Output(4)=u1;
Output(5)=u2;  Output(6)=R_OE_x;  Output(7)=R_OE_y;
FileIdentifier = fopen('all');
WriteOutput( 1,                 Output(1:7) );
WriteOutput( FileIdentifier(1), Output(1:7) );



%=========================================================================
==
function WriteOutput( fileIdentifier, Output )
numberOfOutputQuantities = length( Output );
if numberOfOutputQuantities > 0,
  for i=1:numberOfOutputQuantities,
    fprintf( fileIdentifier, ' %- 14.6E', Output(i) );
  end
  fprintf( fileIdentifier, '\n' );
end



%=========================================================================
==
% mdlTerminate: Perform end of simulation tasks and set sys=[]
%=========================================================================
==
function sys = mdlTerminate(T,VAR,u)
FileIdentifier = fopen('all');
fclose( FileIdentifier(1) );
fprintf( 1, '\n Output is in the file project2.1\n' );
fprintf( 1, '\n To load and plot columns 1 and 2 with a solid line and
columns 1 and 3 with a dashed line, enter:\n' );
fprintf( 1, '    someName = load( ''project2.1'' );\n' );
fprintf( 1, '    plot( someName(:,1), someName(:,2), ''-'', someName(:,1),
someName(:,3), ''--'' )\n\n' );
sys = [];
```

```
%=============================================================================
==
% Sfunction: System/Simulink function from standard template
%=============================================================================
==
function [sys,x0,str,ts] = Sfunction(t,x,u,flag)
switch flag,
  case 0,  [sys,x0,str,ts] = mdlInitializeSizes;    % Initialization of
sys, initial state x0, state ordering string str, and sample times ts
  case 1,  sys = mdlDerivatives(t,x,u);             % Calculate the
derivatives of continuous states and store them in sys
  case 2,  sys = mdlUpdate(t,x,u);                  % Update discrete
states x(n+1) in sys
  case 3,  sys = mdlOutputs(t,x,u);                 % Calculate outputs in
sys
  case 4,  sys = mdlGetTimeOfNextVarHit(t,x,u);     % Return next sample
time for variable-step in sys
  case 9,  sys = mdlTerminate(t,x,u);               % Perform end of
simulation tasks and set sys=[]
  otherwise error(['Unhandled flag = ',num2str(flag)]);
end



%=============================================================================
==
% mdlInitializeSizes: Return the sizes, initial state VAR, and sample
times ts
%=============================================================================
==
function [sys,VAR,stateOrderingStrings,timeSampling] = mdlInitializeSizes
sizes = simsizes;             % Call simsizes to create a sizes structure
sizes.NumContStates  = 4;     % sys(1) is the number of continuous states
sizes.NumDiscStates  = 0;     % sys(2) is the number of discrete states
sizes.NumOutputs     = 7;     % sys(3) is the number of outputs
sizes.NumInputs      = 0;     % sys(4) is the number of inputs
sizes.DirFeedthrough = 1;     % sys(6) is 1, and allows for the output to
be a function of the input
sizes.NumSampleTimes = 1;     % sys(7) is the number of samples times (the
number of rows in ts)
sys = simsizes(sizes);        % Convert it to a sizes array
stateOrderingStrings = [];
timeSampling         = [0 0]; % m-by-2 matrix containing the sample times
OpenOutputFilesAndWriteHeadings
VAR = ReadUserInput
```