

An Introduction to Three-Dimensional, Rigid Body Dynamics

James W. Kamman, PhD

Volume II: Kinetics

Unit 7

Introduction to Modeling Mechanical System Kinetics using MATLAB[®] Scripts, Simulink[®], and Simscape Multibody[®]

Summary

This unit provides an introduction to *modeling* mechanical system kinetics using *MATLAB scripts*, Simulink *models*, and *Simscape Multibody models*. For an introduction to these modeling techniques, see Unit 10 of Volume I for applications in mechanical systems kinematics. The examples presented in this unit assume the reader is familiar with the programming concepts presented in Volume I.

As presented in Volume I, MATLAB scripts are *text-based programs* written in the MATLAB programming language. Simulink models are *block-diagram-based programs* that run in the MATLAB environment. Simscape Multibody models are *block-diagram-based, multibody dynamics programs* that run in the MATLAB/Simulink environment. MATLAB scripts can be used *alone* or *in conjunction with* Simulink and Simscape models.

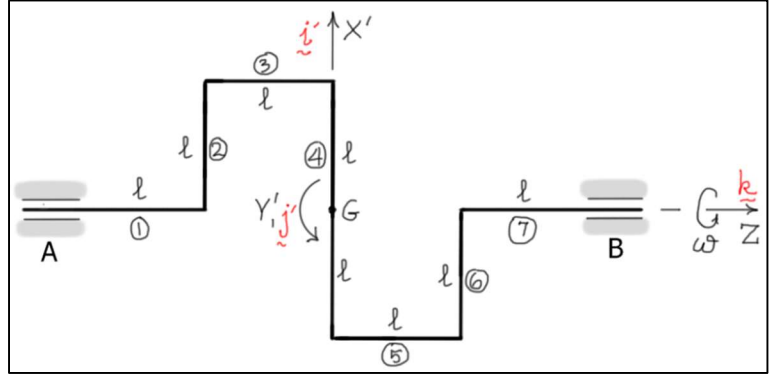
Three of the six models presented herein simulate the *free motion* of an *upright bicycle*. The models are developed using the equations of motion developed in Unit 5 of this volume.

Page Count	Examples	Models	Exercises
57	3	6	4

Trademarks: MATLAB and Simulink, and Simscape Multibody are all registered trademarks of The MathWorks, Inc. The MathWorks does not warrant the accuracy of the examples given in this volume.

Example 1: Rotating Simple Crank Shaft

The figure shows the simple crank shaft from Example 1 of Unit 2 of this volume. The crank shaft consists of seven segments, each considered to be a **slender bar**. Each segment of length ℓ has mass m . There are **six segments** of length ℓ and **one segment** of length 2ℓ (segment 4). The **mass center** of the system G is located on the axis of rotation.



In Unit 2, the elements of the **third column** of the **inertia matrix** of the crank shaft were calculated using the **parallel axes theorems** for moments and products of inertia and were found to be as follows.

$$\boxed{I_{X'Z}^G = -2m\ell^2} \quad \boxed{I_{Y'Z}^G \equiv 0} \quad \boxed{I_{ZZ}^G = \frac{10}{3}m\ell^2}$$

The **remaining nonzero entries** of the inertia matrix can be calculated similarly as follows. The segment numbers are shown to clarify the calculations.

$$I_{X'X'}^G = 2 \left[\overset{\textcircled{1}}{\frac{1}{12}m\ell^2} + \overset{\textcircled{7}}{m\left(\frac{3}{2}\ell\right)^2} \right] + 2 \left[\overset{\textcircled{2}}{m\ell^2} \right] + 2 \left[\overset{\textcircled{3}}{\frac{1}{3}m\ell^2} \right] = 2 \left(\frac{1}{12} + \frac{9}{4} + 1 + \frac{1}{3} \right) m\ell^2 = \frac{22}{3}m\ell^2$$

$$\begin{aligned} I_{Y'Y'}^G &= 2 \left[\overset{\textcircled{1}}{\frac{1}{12}m\ell^2} + \overset{\textcircled{7}}{m\left(\frac{3}{2}\ell\right)^2} \right] + 4 \left[\overset{\textcircled{2}}{\frac{1}{12}m\ell^2} + \overset{\textcircled{6}}{m\left(\ell^2 + \frac{1}{4}\ell^2\right)} \right] + \left[\overset{\textcircled{4}}{\frac{1}{12}(2m)(2\ell)^2} \right] \\ &= \left[2 \left(\frac{1}{12} + \frac{9}{4} \right) + 4 \left(\frac{1}{12} + 1 + \frac{1}{4} \right) + \frac{8}{12} \right] m\ell^2 \\ &= \left(\frac{2}{12} + \frac{54}{12} + \frac{4}{12} + \frac{48}{12} + \frac{12}{12} + \frac{8}{12} \right) m\ell^2 \\ &\Rightarrow \boxed{I_{Y'Y'}^G = \frac{32}{3}m\ell^2} \end{aligned}$$

$$\boxed{I_{ZX'}^G = I_{X'Z}^G = -2m\ell^2}$$

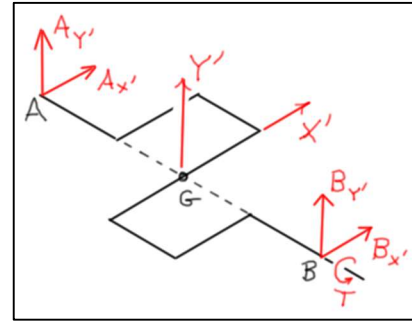
The **remaining** products of inertia are **all zero** due to symmetry of the system about the $X'Z$ plane. Using these results, the **inertia matrix** of the crank shaft can be written as

$$[I_G] = \begin{bmatrix} I_{X'X'}^G & -I_{X'Y'}^G & -I_{X'Z}^G \\ -I_{Y'X'}^G & I_{Y'Y'}^G & -I_{Y'Z}^G \\ -I_{ZX'}^G & -I_{ZY'}^G & I_{ZZ}^G \end{bmatrix} = m\ell^2 \begin{bmatrix} \frac{22}{3} & 0 & 2 \\ 0 & \frac{32}{3} & 0 \\ 2 & 0 & \frac{10}{3} \end{bmatrix}$$

Given the **free-body-diagram** shown below, the following results were found using the Newton/Euler equations of motion. Note that the **weight force** was **not included** to focus the analysis on the **dynamic loads only**, that is, loads due solely to the motion and asymmetry of the crank shaft.

$$\begin{aligned} \sum (\underline{F} \cdot \underline{i}') &= A_{x'} + B_{x'} = 0 \\ \sum (\underline{F} \cdot \underline{j}') &= A_{y'} + B_{y'} = 0 \end{aligned} \quad (\text{force equations})$$

$$\begin{aligned} \sum (\underline{M} \cdot \underline{i}') &= -4 \ell B_{y'} = 2m \ell^2 \dot{\omega} \\ \sum (\underline{M} \cdot \underline{j}') &= 4 \ell B_{x'} = 2m \ell^2 \omega^2 \\ \sum (\underline{M} \cdot \underline{k}) &= T = \frac{10}{3} m \ell^2 \dot{\omega} \end{aligned} \quad (\text{moment equations})$$



Solving these equations gives the following results for the **support force components** and the **driving torque**.

$$\begin{aligned} B_{x'} &= -\frac{1}{2} m \ell \omega^2 = -A_{x'} \\ B_{y'} &= -\frac{1}{2} m \ell \dot{\omega} = -A_{y'} \end{aligned} \quad (\text{support force components}) \quad T = \frac{10}{3} m \ell^2 \dot{\omega} \quad (\text{driving torque})$$

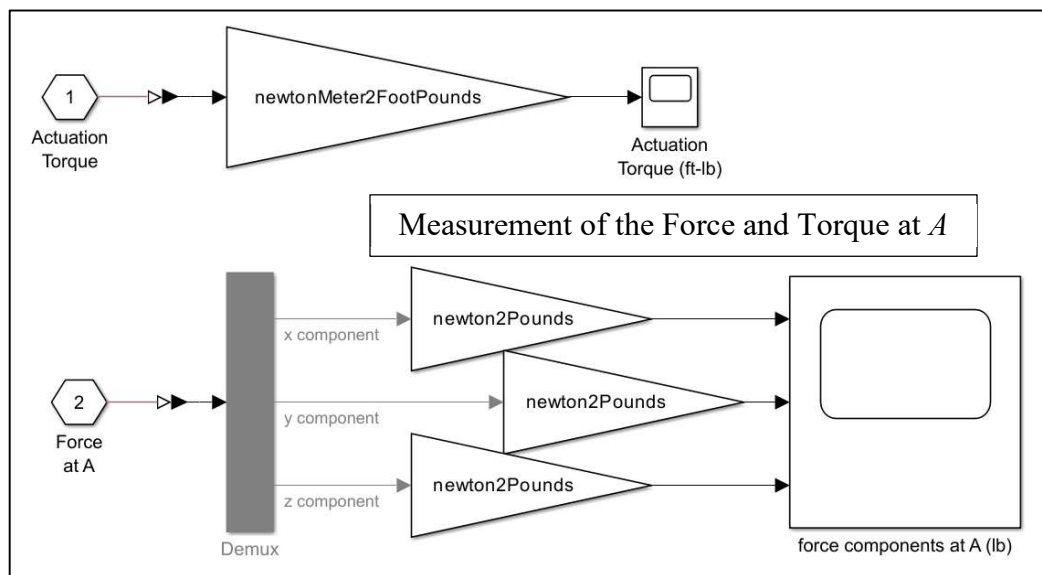
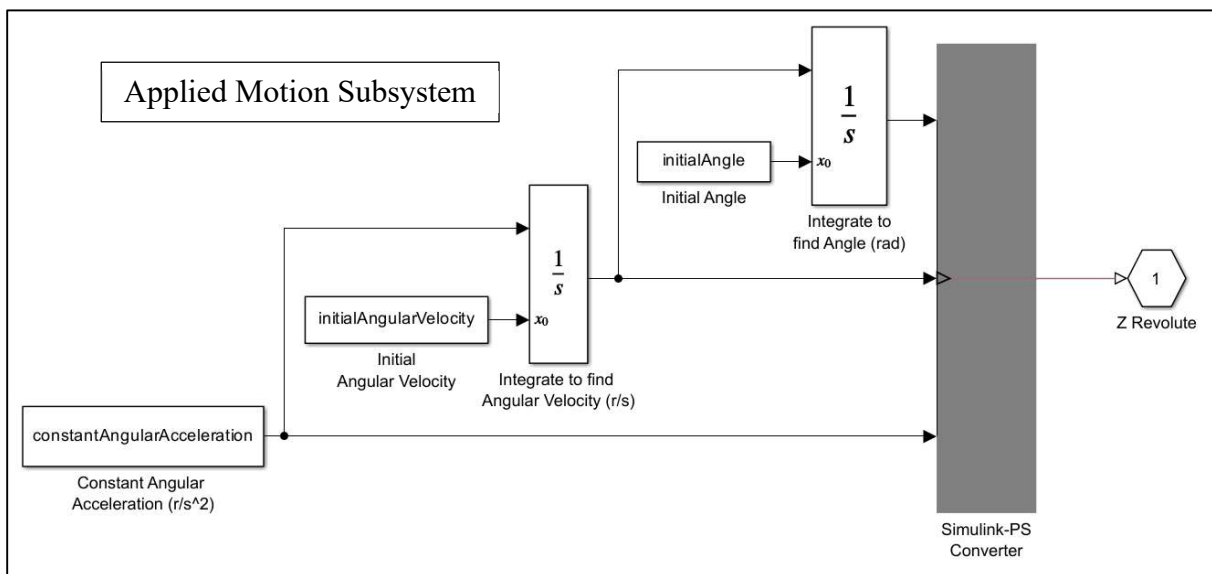
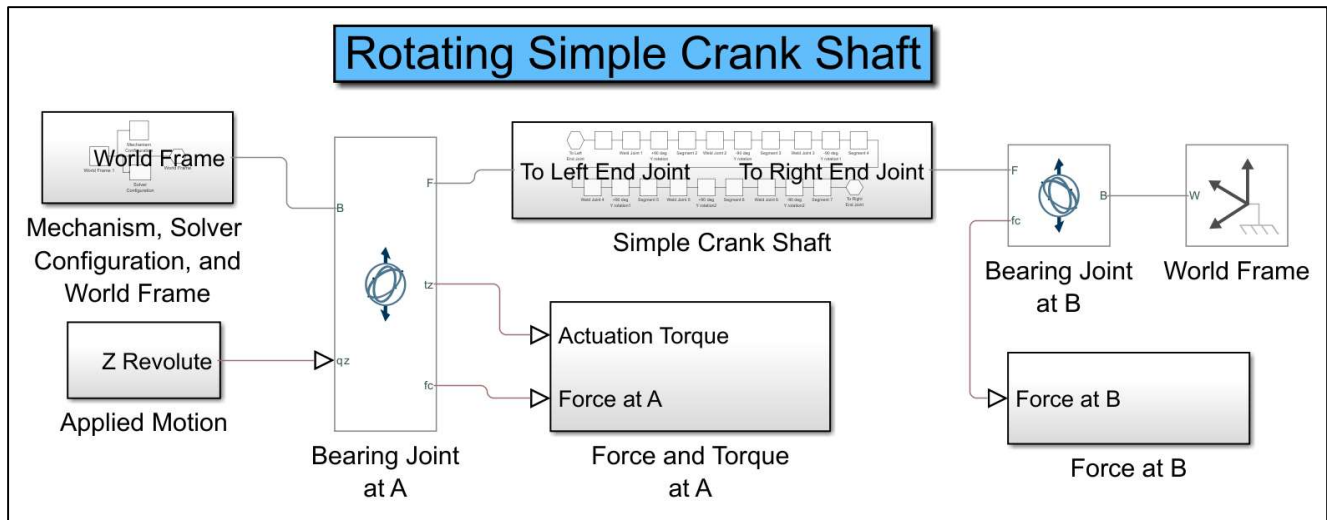
In this example, a **Simscape Multibody** model is used to simulate the motion, support forces, and driving torque on the simple crank shaft. A MATLAB **script** is used to **initialize variables** used in that model, to **execute** the model and **plot** its results, and to **calculate** and **display** analytical results for comparison.

Simulink/Simscape Multibody Model:

The **top layer** of a Simscape model and **two** of its **subsystems** are shown in Panel 1 below. The subsystem in the **upper left** corner contains blocks for the **World Frame**, **Mechanism Configuration**, and **Solver Configuration**. The **World Frame** (or ground frame) is a fixed, orthogonal, right-handed coordinate frame in which the physical model is constructed. All world frame blocks in the model refer to the same frame. The **Mechanism Configuration** block can be used to set the local acceleration of gravity and its direction in the world frame. This setting applies to all the bodies in the model. In this example, the **gravity vector** is defined to be $[0, 0, 0]$, so the **weights** of the system's components are **not included** in the analysis. The resulting reaction forces are due only to dynamic loads.

The **Solver Configuration** block is **required** for all models, and it can be used to **set tolerances** that determine how **accurately** model constraints are maintained during the simulation of the model. For example, some points of a physical system should **ideally coincide** throughout its motion; however, in the simulation of the motion using a Simscape model, the distance between these points must simply be **less than** the set **tolerance**. **Smaller tolerances** give **more accurate results** but will require **longer execution times**. See also Example 4 of Unit 10 of Volume I of this eBook.

The **motion subsystem** in the lower left corner creates motion signals for the angle, angular velocity, and angular acceleration of the crank shaft that **actuates** the system through the **bearing joint** at A. The **measurement subsystem** for the bearing joint at A **records** the actuation torque and the components of the **reaction force** associated with the bearing joint. The **measurement subsystem** for the bearing joint at B records the components of the **reaction force** associated with that joint. All internal calculations are done in SI units, so the measurement subsystems **convert** the results into English units.



The crank shaft is made of *cylindrical steel bars* with a *weight density* of $0.284 \text{ (lb/in}^3\text{)}$ and a *diameter* of 0.5 (in) . Six segments have length $\ell = 6 \text{ (in)}$ and one segment (segment 4) has length $2\ell = 12 \text{ (in)}$. All segments are assumed to be *slender*. The bearing joint at *A* is located at the origin of the world frame, the bearing joint at *B* has coordinates $(0,0,24)$ inches, and the *mass center* *G* of the crank shaft is located halfway between the two joints.

To understand the *reaction forces* and *actuation torque* being recorded, it is helpful to consider the *details* of the *bearing joints*. Both joints lie on the *Z* axis of the world frame and have *one translational* and *three rotational* degrees of freedom. The translational degrees of freedom are both along the world *Z* axis, so the joints *cannot move* in either the *X* or *Y* directions. The *Z* rotational motion of the joint at *A* is *driven* by the applied motion subsystem shown in Panel 1. The measured *actuation torque* is the torque *required* to produce this motion, and the *reaction force components* are those required to *hold* the joints in place. There are *no forces* in the *z* direction, so force components in this direction are expected to be zero.

Block Parameters: Bearing Joint at A

Bearing Joint

Settings

Description

NAME

VALUE

> Z Prismatic Primitive (Pz)

> X Revolute Primitive (Rx)

> Y Revolute Primitive (Ry)

> Z Revolute Primitive (Rz)

> State Targets

> Internal Mechanics

> Limits

> Actuation

Torque

Automatically Computed

Motion

Provided by Input

> Sensing

Position

Velocity

Acceleration

Actuator Torque

Lower-Limit Torque

Upper-Limit Torque

> Mode Configuration

> Composite Force/Torque Sensing

Direction

Base on Follower

Resolution Frame

Follower

Constraint Force

Constraint Torque

Total Force

Total Torque

Block Parameters: Bearing Joint at B

Bearing Joint

Settings

Description

NAME

VALUE

> Z Prismatic Primitive (Pz)

> X Revolute Primitive (Rx)

> Y Revolute Primitive (Ry)

> Z Revolute Primitive (Rz)

> State Targets

> Internal Mechanics

> Limits

> Actuation

Torque

None

Motion

Automatically Computed

> Sensing

Position

Velocity

Acceleration

Actuator Torque

Lower-Limit Torque

Upper-Limit Torque

> Mode Configuration

> Composite Force/Torque Sensing

Direction

Base on Follower

Resolution Frame

Follower

Constraint Force

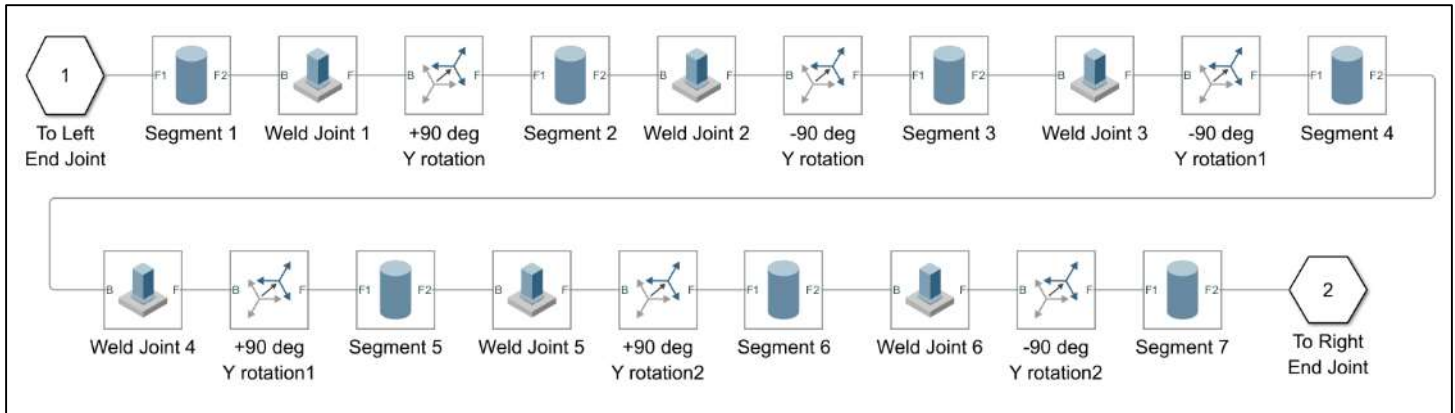
Constraint Torque

Total Force

Total Torque

Using this means of support, the **reaction force components** measured at each joint will be the **same** as those in the analysis above so long as they are measured in the **coordinate system** of the **follower** (i.e., the crank shaft). Note in the Block Parameters window shown above for the bearing joint at *A*, the **Z rotational motion** is listed as “Provided by Input” and the **torque** is listed as “Automatically Computed”. The **constraint force** is the force of the “Base on Follower” and the **resolution frame** is the “Follower Frame”. The Z rotational motion at *B* is listed as “Automatically Computed” with no applied torque. The constraint force is labeled the same as at *A*.

Simscape Multibody Model – Panel 2: (the simple crank shaft subsystem)



Solid : Segment 1

Description

Represents a solid combining a geometry, an inertia and mass, a graphics component, and rigidly attached frames into a single unit. A solid is the common building block of rigid bodies. The Solid block obtains the inertia from the geometry and density, from the geometry and mass, or from an inertia tensor that you specify.

In the expandable nodes under Properties, select the types of geometry, inertia, graphic features, and frames that you want and their parameterizations.

Port R is a frame port that represents a reference frame associated with the geometry. Each additional created frame generates another frame port.

Properties

Geometry

Radius	diameterMeters/2	m	▼	Compile-time	▼
Length	segmentLengthMeters	m	▼	Compile-time	▼






Export

Inertia

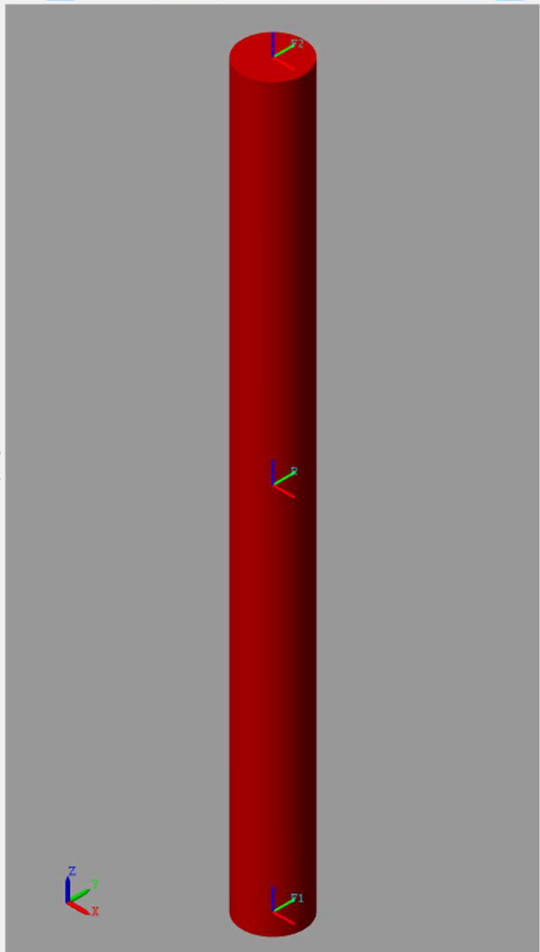
Type	Custom	▼
Mass	segmentMassKilograms	kg ▼ Compile-time ▼
Center of Mass	[0 0 0]	m ▼ Compile-time ▼
Moments of Inertia	[inertiaXX, inertiaYY, inertiaZZ]	kg*m^2 ▼ Compile-time ▼
Products of Inertia	[0 0 0]	kg*m^2 ▼ Compile-time ▼

Graphic

Frames

Show Port R	<input type="checkbox"/>
Frame1	F1  
Frame2	F2  
New Frame	

OK Cancel Help Apply

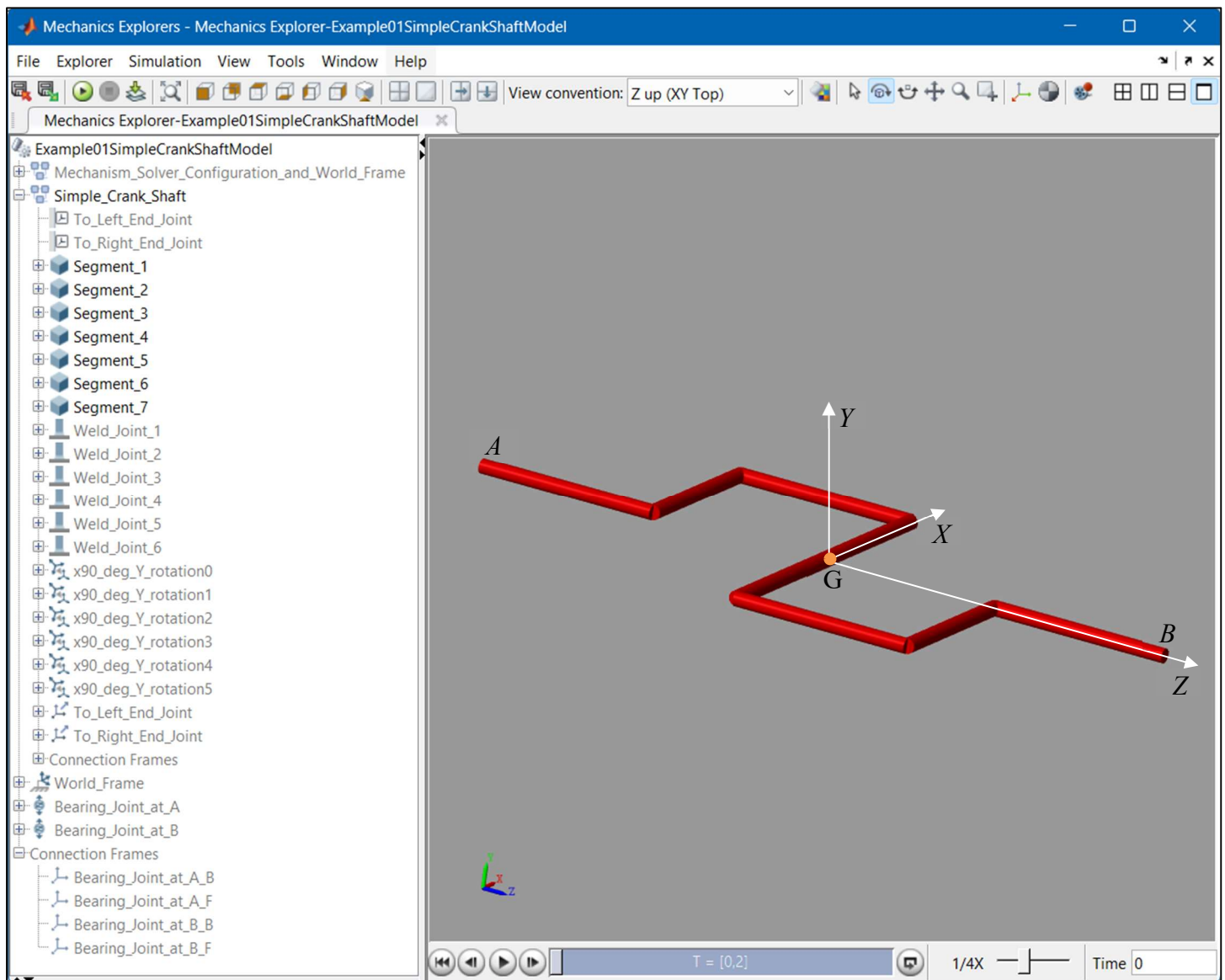


Copyright © James W. Kamman, 2024

Volume II – Unit 7: page 5/57

The subsystem that *builds* the simple crank shaft is shown in Panel 2 above. It consists of a series of cylindrical bodies connected with “weld” joints. Rigid Transform blocks are used to rotate by ± 90 (deg) to force the local z axes to be along the directions (long axes) of the cylinders. Segments 1, 2, 3, 5, 6, and 7 all have length $\ell = 6$ (in), and segment 4 has length $2\ell = 12$ (in). All segments have a weight density of 0.284 (lb/in³) and a diameter of $d = 0.5$ (in). The data window for segment 1 is also shown in Panel 2. Note that workspace variables are used to define the radius, length, mass, and moments of inertia. All are calculated in SI units by the associated script.

The Mechanics Explorer window for the Simscape model is shown below. The left side of the window shows details of the model connectivity which can be used to further verify the correct construction of the model. The right side of the window shows the system in its initial configuration. The system is initially in the XZ plane of the world frame, and it rotates about the Z axis. Labels have been added to the MATLAB diagram for clarity.



MATLAB Script:

The first *two sections* of the MATLAB script are shown in Panel 1 below. The *first section* provides a *brief description* of the script's purpose and clears the workspace variables. The *second* defines the *diameter, length, mass, and moments of inertia* of the seven segments of the crank shaft. The mass and inertia properties are calculated based on the assumption the segments are all right circular cylinders made of steel bar with a weight density of $0.284 \text{ (lb/in}^3\text{)}$ and diameter of 0.5 (in) . The data are all converted to SI units for use in the Simscape model. The variables "newton2Pounds" and "newtonMeter2FootPounds" are used in the Simscape model to convert the output signals back into English units.

MATLAB Script – Panel 1:

```
%% Simple Crank Shaft Simulation
% This file calculates input data for the Simscape Multibody model
% 'Example01SimpleCrankShaftModel', executes the model and plots its results,
% and calculates analytical results and displays them for comparison
%
clear variables

% units conversion factors
inches2Meters      = 2.54/100;           % in to m conversion
feet2Meters        = 12*inches2Meters;   % ft to m conversion
pounds2Newtons     = 4.4482216153;       % lbs to N conversion
newton2Pounds      = 0.2248089431;       % N to lbs conversion
newtonMeter2FootPounds = 0.7375621493;   % N-m to ft-lbs conversion
accelerationGravity = 32.174;           % (ft/s^2)
accelerationGravityMeters = accelerationGravity*feet2Meters; % (m/s^2)
weightDensitySteel = 0.284;             % (lb/in^3)

%% mass and inertia properties
diameter           = 0.5; diameterMeters = diameter*inches2Meters; % (in,m)
weightPerUnitLength = weightDensitySteel*pi*(diameter^2)/4.0; % (lb/in)
segmentLength      = 6.0; % (in)
segmentLengthMeters = segmentLength*inches2Meters; % (m)
segmentWeight      = segmentLength*weightPerUnitLength; % (lb)
segmentWeightNewtons = segmentWeight*pounds2Newtons; % (N)
segmentMass        = segmentWeight/accelerationGravity; % (slugs)
segmentMassKilograms = segmentWeightNewtons/accelerationGravityMeters; % (kg)
numberOfSegments   = 7;
totalMass          = (numberOfSegments + 1)*segmentMass; % (slugs)
totalMassKilograms = (numberOfSegments + 1)*segmentMassKilograms; % (kg)

% inertias for segments 1, 2, 3, 5, 6, 7
inertiaXX = (1/12)*segmentMassKilograms*(segmentLengthMeters^2); % (kg-m^2)
inertiaXY = 0.0; inertiaXZ = 0.0; % (kg-m^2)
inertiaYY = inertiaXX; % (kg-m^2)
inertiaYX = 0.0; inertiaYZ = 0.0; % (kg-m^2)
inertiaZX = 0.0; inertiaZY = 0.0; inertiaZZ = 0.0; % (kg-m^2)

% inertias for segment 4
segment4MassKilograms = 2*segmentMassKilograms; % (kg)
segment4LengthMeters = 2*segmentLengthMeters;
inertiaXXs4 = (1/12)*segment4MassKilograms*(segment4LengthMeters^2); % (kg-m^2)
inertiaXys4 = 0.0; inertiaXzs4 = 0.0; % (kg-m^2)
inertiaYys4 = inertiaXXs4; % (kg-m^2)
inertiaYxs4 = 0.0; inertiaYzs4 = 0.0; % (kg-m^2)
inertiaZxs4 = 0.0; inertiaZys4 = 0.0; inertiaZZs4 = 0.0; % (kg-m^2)
:
```

The *next three sections* of the script are shown in Panel 2 below. The first defines variables used to generate the applied motion of the system. The second *executes* the Simscape Multibody model, and the third *plots* the results in three figures. The first and third are plots of the *constraint forces* at the bearings, and the second is a plot of the *actuation torque* at bearing *A*.

MATLAB Script – Panel 2:

```

                                :
%% motion properties
initialAngle      = 0.0;  %(rad)
initialAngularVelocity = 50.0; %(r/s)
constantAngularAcceleration = 10.0; %(r/s^2)

%% execute the simulink model
sim('Unit07Example01SimpleCrankShaftSimscapeModel')

%% Plot results from the simulation
% Plot the components of the force on the crankshaft at A
figure(1); clf;
subplot(3,1,1); plot(forceComponentsA.time(:),forceComponentsA.signals(1).values(:));
grid; xlabel('Time (sec)'); ylabel('X'-Component (lb)')

subplot(3,1,2); plot(forceComponentsA.time(:),forceComponentsA.signals(2).values(:));
grid; xlabel('Time (sec)'); ylabel('Y'-Component (lb)')

subplot(3,1,3); plot(forceComponentsA.time(:),forceComponentsA.signals(3).values(:));
grid; xlabel('Time (sec)'); ylabel('Z'-Component (lb)')
sgtitle('Force Components at A');

% Plot the components of the torque on the crankshaft at A
figure(2); clf;
plot(actuatorTorque(:,1),actuatorTorque(:,2));
grid; title('Actuator Torque at A');
xlabel('Time (sec)'); ylabel('Torque (ft-lb)')

% Plot the components of the force on the crankshaft at B
figure(3); clf;
subplot(3,1,1); plot(forceComponentsB.time(:),forceComponentsB.signals(1).values(:));
grid; xlabel('Time (sec)'); ylabel('X'-Component (lb)')

subplot(3,1,2); plot(forceComponentsB.time(:),forceComponentsB.signals(2).values(:));
grid; xlabel('Time (sec)'); ylabel('Y'-Component (lb)')

subplot(3,1,3); plot(forceComponentsB.time(:),forceComponentsB.signals(3).values(:));
grid; xlabel('Time (sec)'); ylabel('Z'-Component (lb)')
sgtitle('Force Components at B')
                                :

```

The *last two sections* of the script are shown in Panel 3 below. The first *calculates analytical results* (based on the equations presented above) at the *initial position*, and the second *displays* those results in the MATLAB *command window* for comparison with the Simscape Multibody model results at $t = 0$.

MATLAB Script – Panel 3:

```

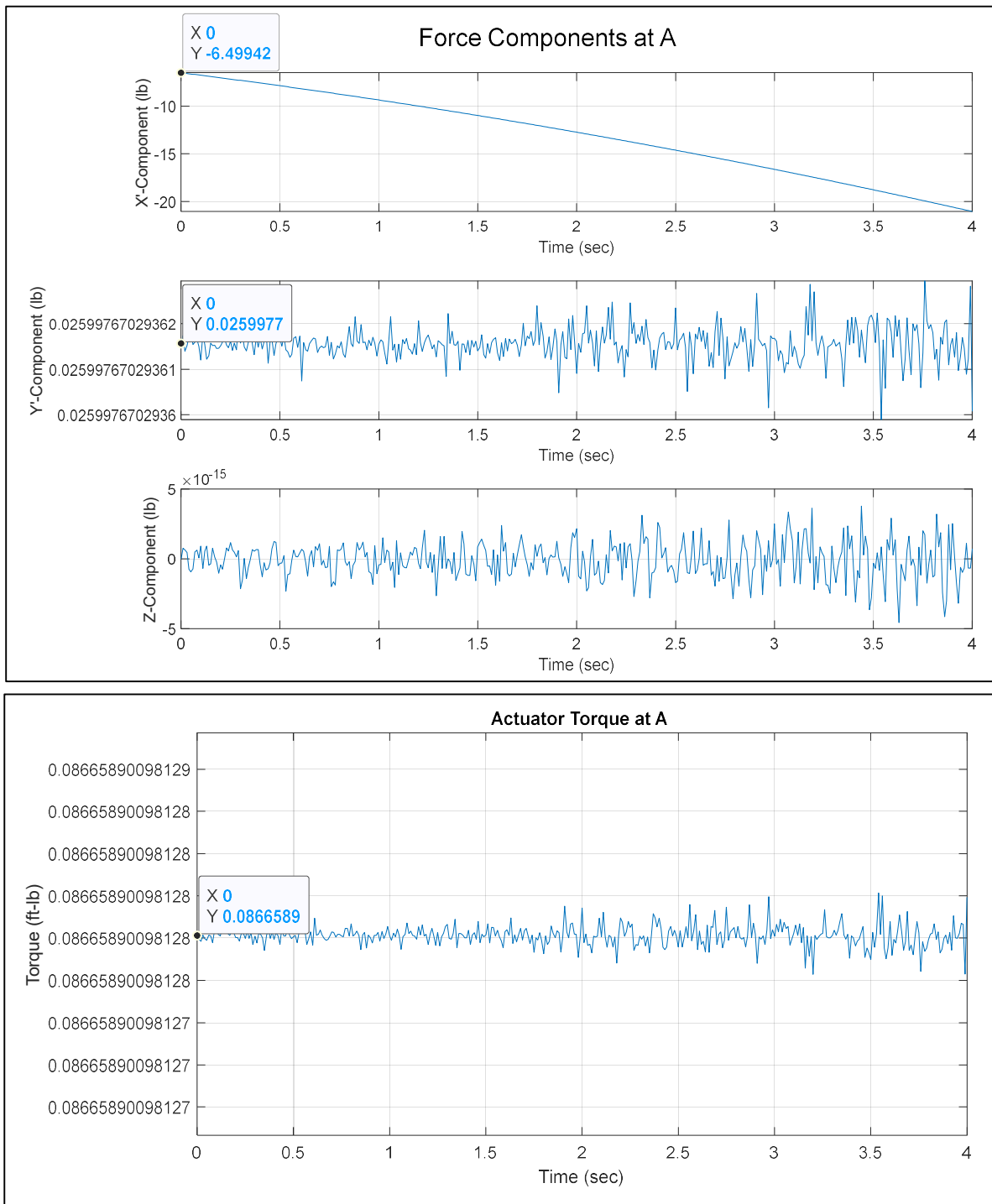
                                :
                                :
%% Display analytical results in Command Window

disp('Analytical Results (neglecting the weight force')
disp('-----')
disp('-')
disp('Initial Forces Applied to Crank Shaft at A:')
disp('-----')
forceString = ['X': ',num2str(forceAXPrime), ...
               ' Y': ',num2str(forceAYPrime)];
disp(forceString);
disp('-')
disp('Initial Forces Applied to Crank Shaft at B:')
disp('-----')
forceString = ['X': ',num2str(forceBXPrime), ...
               ' Y': ',num2str(forceBYPrime)];
disp(forceString);
disp('-')
disp('Initial Torques Applied to Crank Shaft at A:')
disp('-----')
torqueString = ['X': ',num2str(torqueSystemX), ...
                ' Y': ',num2str(torqueSystemY), ...
                ' Z': ',num2str(torqueSystemZ)];
disp(torqueString);
```

Simulation Results:

When the script is executed, a **geometric representation** of the system is displayed in the Mechanics Explorer window, and as the simulation progresses, the motion of the system is **animated**. The **constraint force components** and the actuator **torque** recorded by the measurement subsystem at *A* are shown in the diagrams below from zero to four seconds. The force components at *B* are not shown; they are simply the negatives of those found at *A*. Although producing those results may seem **redundant**, observation of those expected results can provide the analyst with some additional level of **confidence** in the results.

Note that (as expected) only the X' and Y' components of force are **nonzero**. The X' component of the force increases as the angular velocity increases, and the Y' component stays nearly constant. Note also that even though the Y' component of the force and the actuator torque should both be **constant**, and the Z component of the force should be **zero**, the plots clearly show some **fluctuation** in these values. These small fluctuations are **numerical error** (noise) associated with the **level of accuracy** of the calculations.



Analytical Results:

Analytical results are calculated in the **initial position** using the formulae presented above and **displayed** in the MATLAB command window. The results for the Y' components of the forces and the Z component of the torque are dependent on the angular acceleration, so they are **constant**. However, the X' component of the force depends on the angular velocity of the system. The value given below applies only to the initial position. Beyond that time, the X' components of the forces **increase quadratically** with the angular velocity. Note these results are **identical** to those found by the Simscape model at the **initial position**.

Analytical Results (neglecting the weight force)

Initial Forces Applied to Crank Shaft at A:

X': -6.4994 Y': 0.025998

Initial Forces Applied to Crank Shaft at B:

X': 6.4994 Y': -0.025998

Initial Torques Applied to Crank Shaft at A:

X': 0 Y': 0 Z: 0.086659

Example 2: Rotating Bar on a Rotating Frame

The system shown consists of **two bodies**, the frame F and bar B . Frame F rotates about the **fixed vertical direction** annotated by the unit vector \hat{k} . Bar B is pinned to and rotates about the **horizontal, rotating arm** of F . F rotates **relative** to the **ground** at a rate Ω (r/s) and B rotates **relative** to F at a rate of ω (r/s). The reference frames used in the analysis:

$$F: (\hat{n}_1, \hat{n}_2, \hat{k}) \quad B: (\hat{e}_1, \hat{e}_2, \hat{e}_3)$$

In Unit 2 of this volume, it was shown that if the **angular rate** Ω is **known** while the **motion** of B relative to the frame is **free**, the **forces** and **torques** on the pin at G required to maintain the motion and the differential equation governing the angle θ can be written as follows.

Unknown forces:

$$\sum \vec{F} \cdot \hat{e}_1 = F_1 = -m d \dot{\Omega} C_\theta \quad \sum \vec{F} \cdot \hat{n}_2 = F_2 = -m d \Omega^2 \quad \sum \vec{F} \cdot \hat{e}_3 = F_3 = -m d \dot{\Omega} S_\theta \quad (\text{expressed in } B)$$

$$\sum \vec{F} \cdot \hat{n}_1 = (F_1)_F = -m d \dot{\Omega} \quad \sum \vec{F} \cdot \hat{n}_2 = (F_2)_F = F_2 = -m d \Omega^2 \quad \sum \vec{F} \cdot \hat{n}_3 = (F_3)_F = 0 \quad (\text{expressed in } F)$$

Unknown torques:

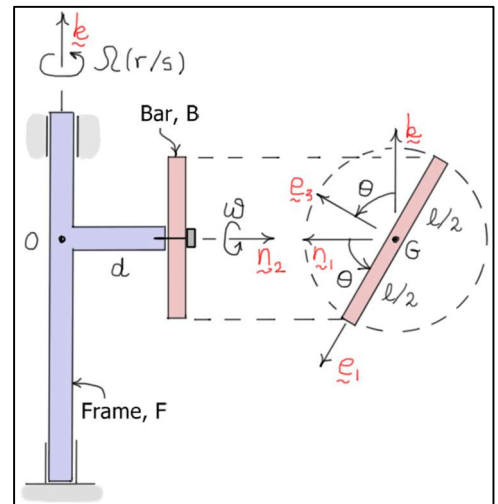
$$\sum \vec{M} \cdot \hat{e}_1 = T_1 = 0 \quad \sum \vec{M} \cdot \hat{e}_3 = T_3 = \frac{m \ell^2}{12} (\dot{\Omega} C_\theta - 2 \omega \Omega S_\theta) \quad (\text{expressed in } B)$$

$$\vec{T} = T_3 (S_\theta \hat{n}_1 + C_\theta \hat{k}) \quad (\text{expressed in } F)$$

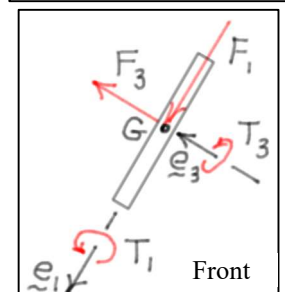
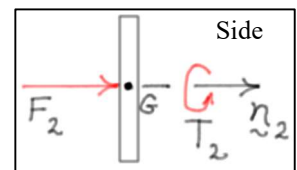
Differential equation for the angle θ :

$$\ddot{\theta} + \Omega^2 S_\theta C_\theta = \frac{T_2}{\left(\frac{1}{12} m \ell^2\right)} \quad (\text{torque } T_2 \text{ is assumed to be a } \mathbf{known} \text{ torque})$$

Note that to generate these equations, it was **assumed** that frame, F is **light** and that bar, B is **slender**. See Unit 2 of this volume for additional details.



Free Body Diagrams



In Unit 4 of this volume, using Lagrange's equations, the following **two differential equations** of motion were derived for this system.

$$\boxed{\begin{aligned} \left(md^2 + \frac{1}{12} m \ell^2 C_\theta^2 \right) \ddot{\phi} - \left(\frac{1}{6} m \ell^2 S_\theta C_\theta \right) \dot{\theta} \dot{\phi} &= M_\phi(t) \\ \left(\frac{1}{12} m \ell^2 \right) \ddot{\theta} + \left(\frac{1}{12} m \ell^2 S_\theta C_\theta \right) \dot{\phi}^2 &= M_\theta(t) \end{aligned}} \quad (\text{derived in Unit 4})$$

In these equations, angle ϕ represents the **angular rotation** of frame F about the \underline{k} direction, and angle θ represents the angular rotation of bar B about the horizontal arm of F (as shown in the figure above). The **second equation** is simply a restatement of the boxed equation above involving the torque T_2 (note that torque $M_\theta(t)$ is simply torque T_2). If, as assumed above, angle ϕ and its derivatives are **known**, then the first equation represents a means of calculating the **driving** (or **actuating**) **torque** acting on F about the \underline{k} direction.

$$\boxed{M_\phi(t) = \left(md^2 + \frac{1}{12} m \ell^2 C_\theta^2 \right) \dot{\Omega} - \left(\frac{1}{6} m \ell^2 S_\theta C_\theta \right) \omega \Omega}$$

Note that this equation was derived assuming frame F is **light**. If F is **not light**, it can be shown that the equation can be rewritten as follows.

$$\boxed{M_\phi(t) = \left((I_Z)_F + md^2 + \frac{1}{12} m \ell^2 C_\theta^2 \right) \dot{\Omega} - \left(\frac{1}{6} m \ell^2 S_\theta C_\theta \right) \omega \Omega}$$

Here, $(I_Z)_F$ is the moment of inertia of frame F about its axis of rotation (the Z axis).

In this example, **two** MATLAB **models** are presented of the rotating bar system. The **first model** is a Simscape Multibody model, and the **second** is a Simulink model. The Simulink model solves the basic equations found in Units 2 and 4. **Descriptions** of each of the models are presented below.

Simulink/Simscape Multibody Model:

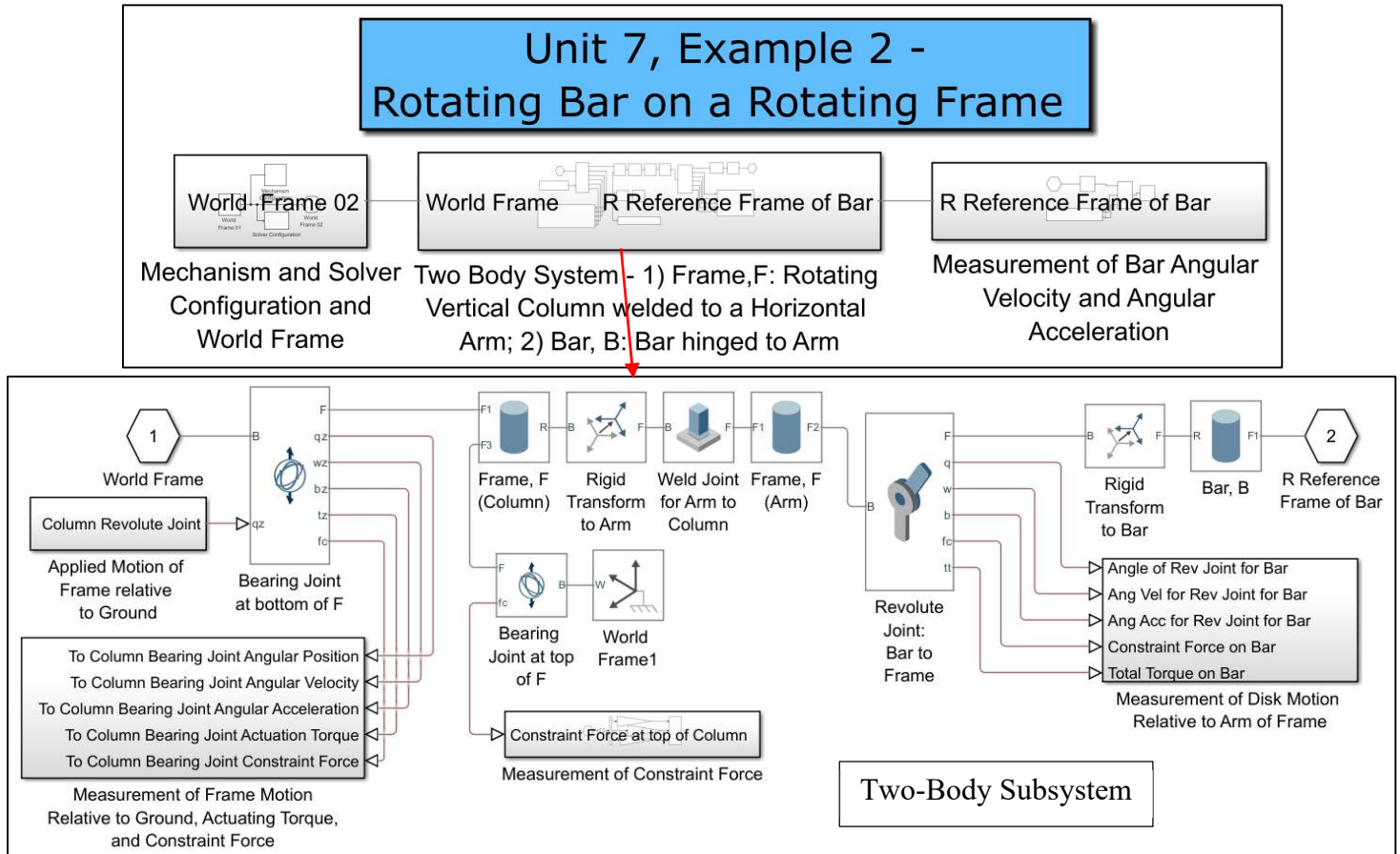
The Simscape Multibody model consists of **two bodies** – frame F (a **vertical column** with a **horizontal arm**), and **bar** B . The two segments of F are assumed to be **slender bars**. The **vertical column** is 24 (in) long and has a mass of 0.01 (slugs), and the **horizontal arm** is 12 (in) long and has a mass of 0.005 (slugs). The two segments are connected using a “weld” joint 18 (in) above the bottom support. Bar B is also assumed to be **slender** with length $\ell = 1$ (ft) and a mass of 0.2 (slugs). The column, arm, and bar all have a radius of 0.5 (in) for **visualization purposes**. The **top layer** of the model and the **two-body subsystem** are shown in Panel 1 below.

The subsystem on the left of the top layer defines the Mechanism and Solver Configurations and the world reference frame as described in Example 1 above. Also, as in that example, the **gravity vector** is defined to be $[0, 0, 0]$, so the **weights** of the system's components are **not included** in the analysis. The resulting reaction forces are due only to **dynamic loads**.

The subsystem in the center of the top layer contains the details of the two-body system. The **two bearing joints** that connect the **vertical column** of the frame to the **ground** are like those used in Example 1 above for the

simple crank shaft. The **bearing joints** allow for **three axes of rotation**, and **one axis of translation** along the world frame's Z axis. Using this method of support, the model can be used to calculate the **reaction forces** and **torques** at the two joints and the **actuating torque** at the lower support. The **motion** of the revolute joint of the lower bearing joint is **driven** by the applied motion subsystem which is like the one used in Example 1.

Simscape Multibody Model – Panel 1: (top layer and the two-body subsystem)



The Block Parameters windows for the **bearing joints** at the **bottom** and **top** of *F* are shown below. Note that the **motion** of the Z Revolute Primitive for the bottom joint is listed as “Provided by Input”, and its Actuation Torque is listed as “Automatically Computed”. The position (angle), velocity (angular rate), acceleration (angular acceleration), actuation torque, and constraint force are all being sensed (measured). The constraint force is listed as “Base on Follower” to be resolved in the “Follower” frame. This is the force the world frame applies to *F* with the components resolved in *F*. The **motion** of the Z Revolute Primitive of the top joint is listed as “Automatically Computed” and the actuating torque as zero (none). The constraint force for the top bearing joint is listed the same as the bottom one.

The Block Parameters window for the **revolute joint** that connects the **horizontal arm** to the **bar** is also shown below. The joint torque is listed as “None” (zero), and the motion is listed as “Automatically Computed”. Hence, this joint allows free rotation of the bar relative to *F* about the long axis of the arm. The constraint force and

torque are listed as “Base on Follower” to be resolved in the base frame. These are the constraint force and torque applied by F to the bar with components resolved in F .

Block Parameters: Bearing Joint at bottom of F

Bearing Joint

Auto Apply

SettingsDescription

NAME

VALUE

> Z Prismatic Primitive (Pz)

> X Revolute Primitive (Rx)

> Y Revolute Primitive (Ry)

> Z Revolute Primitive (Rz)

State Targets

Internal Mechanics

Limits

Actuation

Torque

Automatically Computed

Motion

Provided by Input

Sensing

Position

Velocity

Acceleration

Actuator Torque

Lower-Limit Torque

Upper-Limit Torque

Mode Configuration

Composite Force/Torque Sensing

Direction

Base on Follower

Resolution Frame

Follower

Constraint Force

Constraint Torque

Total Force

Total Torque

Block Parameters: Bearing Joint at top of F

Bearing Joint

Auto Apply

SettingsDescription

NAME

VALUE

> Y Revolute Primitive (Ry)

> Z Revolute Primitive (Rz)

State Targets

Internal Mechanics

Limits

Actuation

Torque

None

Motion

Automatically Computed

Sensing

Position

Velocity

Acceleration

Actuator Torque

Lower-Limit Torque

Upper-Limit Torque

Mode Configuration

Composite Force/Torque Sensing

Direction

Base on Follower

Resolution Frame

Follower

Constraint Force

Constraint Torque

Total Force

Total Torque

Block Parameters: Revolute Joint: Bar to Frame

Revolute Joint

Auto Apply

SettingsDescription

NAME

VALUE

> Z Revolute Primitive (Rz)

State Targets

Specify Position Target

Priority

Low (approximate)

Value

initialTheta

1.0472

rad

Compile-time

Specify Velocity Target

Priority

Low (approximate)

Value

initialThetaDot

3

rad/s

Compile-time

Internal Mechanics

Equilibrium Position

0

deg

Compile-time

Spring Stiffness

0

N*m/deg

Compile-time

Damping Coefficient

dampingCoefficient

0.0...

N*m*s/rad

Run-time

Limits

Actuation

Torque

None

Motion

Automatically Computed

Sensing

Mode Configuration

Composite Force/Torque Sensing

Direction

Base on Follower

Resolution Frame

Base

Constraint Force

Constraint Torque

Total Force

Total Torque

The initial angle θ and angular rate $\dot{\theta}$ of the revolute joint are listed under “State Targets”. As listed, these values are applied at compile time (initial time). A damping torque is applied to the joint under the heading of “Internal Mechanics”. The damping coefficient is listed as applied at “Run Time”, that is, throughout the entire simulation. The initial angle “initialTheta”, initial angular rate “initialThetaDot”, and the damping coefficient “dampingCoefficient” are all specified using variables defined in the associated MATLAB script.

As noted above, frame F consists of *two right, circular cylinders*. The *first cylinder* (vertical column) is aligned with the world frame’s Z axis. The *second* (horizontal arm) is connected using a “weld” joint to the vertical column. The *weld joint* is located at a *distance* of one-sixth of the column length *above* its geometric center, and the *arm* is pointed *horizontally outward* initially along the world frame’s Y axis. The block parameters for the vertical column and the Rigid Transform to Arm blocks are shown in Panel 2 below.

The *vertical column* has three reference frames. Frame $F1$ is at the bottom, $F3$ is at the top, and R is at the geometric center. $F1$ is connected to the *bottom bearing joint*, and $F3$ is connected to the *top bearing joint*. A Rigid Transform is used to *vertically translate* from R to the weld joint and to rotate by -90 (deg) about the X axis so the local z axis is pointed *horizontally outward* along the arm. The transformation is applied at “Compile-time” to build the frame. The *geometric* and *inertia parameters* of the vertical column and the *vertical translation* to the weld joint are specified by workspace variables defined in the associated script.

Simscape Multibody Model – Panel 2: (parameters for Vertical Column and Rigid Transforms to Arm and Bar)

The screenshot shows the Simscape Multibody Model interface. The left pane displays the properties of the 'Solid : Frame, F (Column)'. The right pane shows a 3D visualization of the vertical column with its reference frames.

Description
Represents a solid combining a geometry, an inertia and mass, a graphics component, and rigidly attached frames into a single unit. A solid is the common building block of rigid bodies. The Solid block obtains the inertia from the geometry and density, from the geometry and mass, or from an inertia tensor that you specify.

In the expandable nodes under Properties, select the types of geometry, inertia, graphic features, and frames that you want and their parameterizations.

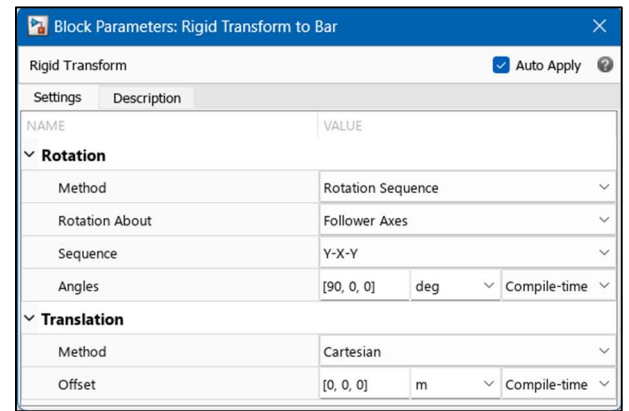
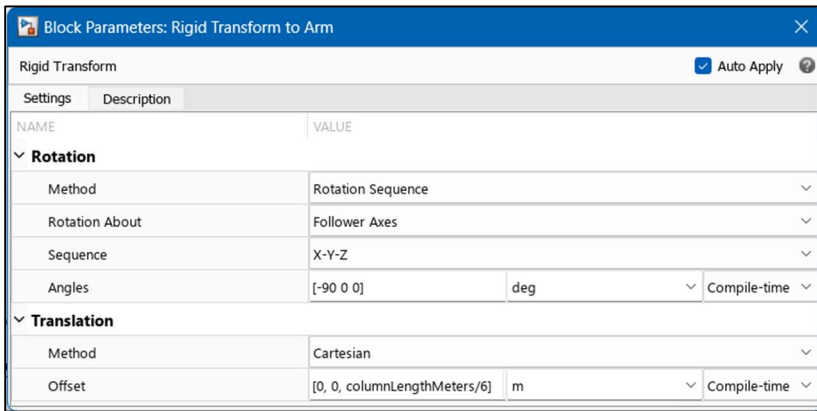
Port R is a frame port that represents a reference frame associated with the geometry. Each additional created frame generates another frame port.

Properties

Geometry			
Radius	columnRadiusMeters	m	Compile-time
Length	columnLengthMeters	m	Compile-time
Inertia			
Type	Custom		
Mass	columnMassKg	kg	Compile-time
Center of Mass	[0 0 0]	m	Compile-time
Moments of Inertia	[IxxColumn, IyyColumn, IzzColumn]	kg*m^2	Compile-time
Products of Inertia	[0 0 0]	kg*m^2	Compile-time
Graphic			
Frames			
Show Port R	<input checked="" type="checkbox"/>		
Frame1	F1		
Frame3	F3		
New Frame			

OK Cancel Help Apply

The 3D visualization on the right shows a vertical blue cylinder. A coordinate system is visible at the bottom of the cylinder, with the Z-axis pointing upwards and the X and Y axes pointing horizontally. The cylinder is centered on the Z-axis.

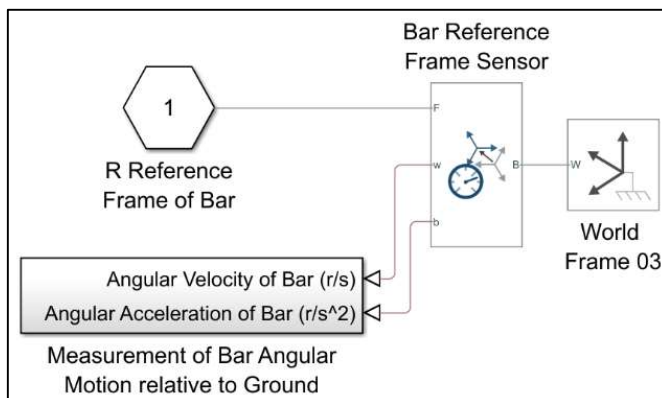
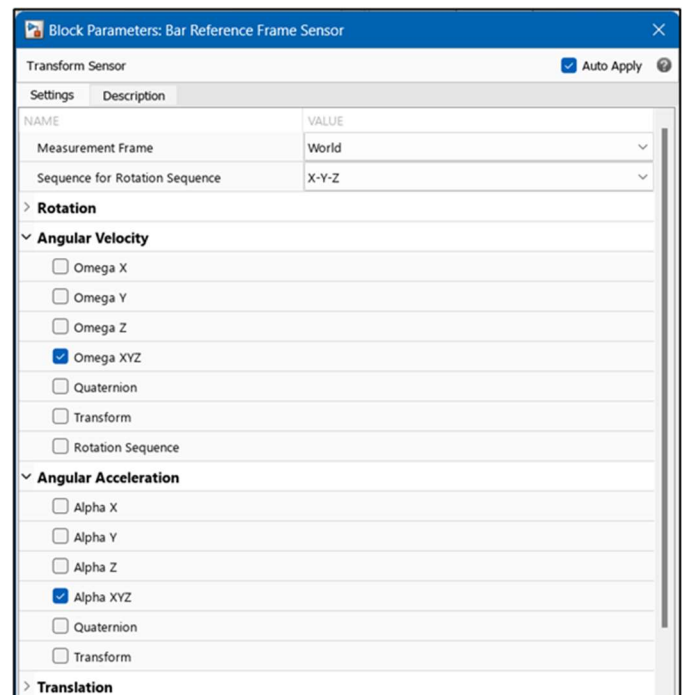


The **horizontal arm** also has three reference frames. *F1* is at the left end, *F2* is at the right end, and *R* is at the geometric center. *F1* is connected to the **weld joint**, and *F2* is connected to the **revolute joint** for the bar. *R* is not used. The revolute joint allows **free rotation** about the long axis of the arm. As with the vertical column, the geometric and inertia parameters are specified using workspace variables defined in the associated script.

To orient bar *B* correctly relative to the horizontal arm a rotation of 90 (deg) about the local *y* axis is performed. This guarantees the long axis of *B* is **perpendicular** to the axis of the revolute joint. See the Block Parameters window above in Panel 2 for the Rigid Transform to Bar block. Again, the geometric and inertia parameters are specified using workspace variables defined in the associated script.

There are **four measurement** subsystems in the model. **Two** of the measurement subsystems are for the bearing joints connecting the **column** to the **ground**, **one** is for the connection between the **arm** and the **bar**, and the **last** measures the angular velocity and angular acceleration of the bar relative to ground. These subsystems use PS-Simulink converters to export the signals to Simulink and conversion factors defined in the associated script to convert all the units of all angles to degrees and all forces and torques to English units.

The **third subsystem** in the top layer contains the details for the calculation of the angular velocity and angular acceleration of bar *B*. This is done using a Transform Sensor to measure the motion of the bar relative to the world frame. The components are resolved in the world frame.



MATLAB Script:

The associated MATLAB script is shown in Panels 1-3 below. **Panel 1** shows the *first five sections* of the script. The *first section* defines the purpose of the script, clears the workspace variables, defines a set of unit's conversion factors, and defines the time grid. The second section defines the geometric, mass and inertia properties of the vertical column, horizontal arm, and the bar. The third section defines the motion data for frame F . In particular, $\dot{\Omega} = \ddot{\phi} = 1 \text{ (r/s}^2\text{)} = \text{constant}$, $\Omega(t=0) = \dot{\phi} = 2\pi \text{ (r/s)}$, and $\phi(t=0) = 0$. The fourth section defines the initial conditions for the angle θ and the damping coefficient for the revolute joint's rotational damper. In particular, $\dot{\theta}(t=0) = 3 \text{ (r/s)}$, $\theta(t=0) = 60 \text{ (deg)}$, and the damping coefficient is $0.02 \text{ (ft-lb-s/rad)}$. The fifth section executes the Simscape Multibody model.

Panel 2 shows the script that plots the constraint force components and actuating torque applied to frame F at the bearing joints and the constraint force and torque applied to bar B at the revolute joint. The constraint force components at the bearing joints are resolved into components in the column's reference frame. These directions align with the \underline{n}_1 , \underline{n}_2 , and \underline{k} directions. The actuating torque is in the \underline{k} direction. The constraint force and torque acting on B at the revolute joint are resolved into the arm's reference frame. This frame is rotated by 90 (deg) from the column's reference frame, so the arm's x axis is along the \underline{n}_1 direction, the arm's z axis is along the \underline{n}_2 direction, and the arm's y axis is along the $-\underline{k}$ direction. The script makes these corrections to plot these results in the column's frame.

Panel 3 shows the script that plots θ the angle of the bar and the world components of the angular velocity and angular acceleration of the bar.

Recall that *scope blocks* in the *measurement subsystems* of the Simscape Multibody model *record* the simulation results in the MATLAB workspace for later plotting by the script.

MATLAB Script – Panel 1: (define mass and inertia and motion data, damping coefficient, and execute)

```
% This M-file:
% 1) Defines the input data for a Simscape Multibody model of
% Example 2 of Unit 7, Volume II
% 2) Executes the Simscape model
% 3) Plots results.

clear variables

% units' conversion factors
deg2rad = pi/180; % deg to rad conversion
rad2deg = 180/pi; % rad to deg conversion
inches2Meters = 2.54/100; % in to m conversion
slugs2Kg = 14.5939029; % slug to kg conversion
newton2Pounds = 0.2248089431; % N to lbs conversion
newtonMeter2FootPounds = 0.7375621493; % N-m to ft-lbs conversion
footPounds2NewtonMeter = 1.3558179483; % ft-lbs to N-m conversion

% Time data
timeMax = 10.0; % (sec)
timeIncrement = 0.01; % (sec)

%% Geometric, Mass, and Inertia data - all bodies are all treated as slender bars

% Column data
columnRadius = 0.5; columnRadiusMeters = columnRadius*inches2Meters; % (in,m)
columnLength = 24; columnLengthMeters = columnLength*inches2Meters; % (in,m)
columnMass = 0.01; columnMassKg = columnMass*slugs2Kg; % (slug,kg)
IxxColumn = (1/12)*columnMassKg*(columnLengthMeters^2); % (kg-m^2)
IyyColumn = IxxColumn; % (kg-m^2)
IzzColumn = (1/2)*columnMassKg*(columnRadiusMeters^2); % (kg-m^2)

% Arm data
armRadiusMeters = columnRadiusMeters; % (m)
armLengthMeters = 0.5*columnLengthMeters; % (m)
armMassKg = 0.5*columnMassKg; % (kg)
IxxArm = (1/12)*armMassKg*(armLengthMeters^2); % (kg-m^2)
IyyArm = IxxArm; % (kg-m^2)
IzzArm = (1/2)*armMassKg*(armRadiusMeters^2); % (kg-m^2)

% Bar data
barRadiusMeters = columnRadiusMeters; % (m)
barLength = 12.0; barLengthMeters = barLength*inches2Meters; % (in, m)
barMass = 0.2; barMassKg = barMass*slugs2Kg; % (slug,kg)
IxxBar = (1/12)*barMassKg*(barLengthMeters^2); % (kg-m^2)
IyyBar = IxxBar; % (kg-m^2)
IzzBar = 0.0; % assuming a slender bar

%% Motion data for Frame, F
initialPhi = 0.0; % (rad)
initialCapOmega = 2*pi; % (rad/sec)
capOmegaDot = 1.0; % (rad/s^2)

%% Initial motion data for Bar, B
initialTheta = 60.0*deg2rad; % (rad)
initialThetaDot = 3.0; % (r/s)
dampingCoefficient = 0.02*footPounds2NewtonMeter; % (N-m-s/rad)

%% Execute the SimMechanics model
sim('Unit07Example02RotatingBarRotatingFrameModelSimscape.slx')
```

MATLAB Script – Panel 2: (plot the force and torque components on the frame and the bar)

```

:
%% Plot the simulation results
% Plot the components of the force on the bottom of the column (A)
figure(1); clf;
subplot(3,1,1); plot(forceComponentsA.time(:),forceComponentsA.signals(1).values(:));
grid on; xlabel('Time (sec)'); ylabel('n_1 Component (lb)')

subplot(3,1,2); plot(forceComponentsA.time(:),forceComponentsA.signals(2).values(:));
grid on; xlabel('Time (sec)'); ylabel('n_2 Component (lb)')

subplot(3,1,3); plot(forceComponentsA.time(:),forceComponentsA.signals(3).values(:));
grid on; xlabel('Time (sec)'); ylabel('k Component (lb)')
sgtitle('Force Components at Bottom of Column');

% Plot the components of the torque on the bottom of the column (A)
figure(2); clf;
plot(actuationTorqueA.time(:),actuationTorqueA.signals(1).values(:));
grid on; xlabel('Time (sec)'); ylabel('Actuation Torque (ft-lb)')
title('Actuation Torque at Bottom of the Column');

% Plot the components of the force on the top of the column (B)
figure(3); clf;
subplot(3,1,1); plot(forceComponentsB.time(:),forceComponentsB.signals(1).values(:));
grid on; xlabel('Time (sec)'); ylabel('n_1 Component (lb)')

subplot(3,1,2); plot(forceComponentsB.time(:),forceComponentsB.signals(2).values(:));
grid on; xlabel('Time (sec)'); ylabel('n_2 Component (lb)')

subplot(3,1,3); plot(forceComponentsB.time(:),forceComponentsB.signals(3).values(:));
grid on; xlabel('Time (sec)'); ylabel('k Component (lb)')
sgtitle('Force Components at Top of Column');

% Plot the components of the force on the bar
figure(4); clf;
subplot(3,1,1); plot(forceComponentsBar.time(:),forceComponentsBar.signals(1).values(:));
grid on; xlabel('Time (sec)'); ylabel('n_1 Component (lb)')

subplot(3,1,2); plot(forceComponentsBar.time(:),forceComponentsBar.signals(3).values(:));
grid on; xlabel('Time (sec)'); ylabel('n_2 Component (lb)')

subplot(3,1,3); plot(forceComponentsBar.time(:),-forceComponentsBar.signals(2).values(:));
grid on; xlabel('Time (sec)'); ylabel('k Component (lb)')
sgtitle('Force Components on the Bar');

% Plot the components of the torque on the bar
figure(5); clf;
subplot(3,1,1); plot(torqueComponentsBar.time(:),torqueComponentsBar.signals(1).values(:));
grid on; xlabel('Time (sec)'); ylabel('n_1 Component (ft-lb)')

subplot(3,1,2); plot(torqueComponentsBar.time(:),torqueComponentsBar.signals(3).values(:));
grid on; xlabel('Time (sec)'); ylabel('n_2 Component (ft-lb)')

subplot(3,1,3); plot(torqueComponentsBar.time(:),-torqueComponentsBar.signals(2).values(:));
grid on; xlabel('Time (sec)'); ylabel('k Component (ft-lb)')
sgtitle('Torque Components on Bar');
:

```

MATLAB Script – Panel 3: (plot the angle of the bar and the bar's angular velocity and angular acceleration)

```
% Plot the angle of the bar, theta
figure(6); clf; plot(angleBar.time(:),angleBar.signals.values(:)); grid on;
title('Angle of the Bar, Theta'); xlabel('Time (sec)'); ylabel('Angle (deg)')

% Plot the world frame components of the bar angular velocity
figure(7); clf;
subplot(3,1,1); plot(barAngularVelocity.time(:),barAngularVelocity.signals(1).values(:));
grid on; xlabel('Time (sec)'); ylabel('X Component (ft-lb)')

subplot(3,1,2); plot(barAngularVelocity.time(:),barAngularVelocity.signals(2).values(:));
grid on; xlabel('Time (sec)'); ylabel('Y Component (ft-lb)')

subplot(3,1,3); plot(barAngularVelocity.time(:),barAngularVelocity.signals(3).values(:));
grid on; xlabel('Time (sec)'); ylabel('Z Component (ft-lb)')
sgtitle('Angular Velocity of Bar - World Frame Components');

% Plot the world frame components of the bar angular acceleration
figure(8); clf;
subplot(3,1,1); plot(barAngularAcceleration.time(:),barAngularAcceleration.signals(1).values(:));
grid on; xlabel('Time (sec)'); ylabel('X Component (ft-lb)')

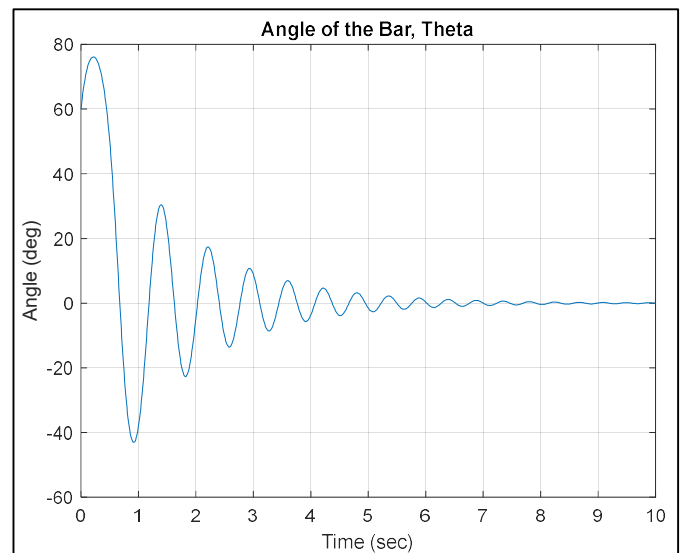
subplot(3,1,2); plot(barAngularAcceleration.time(:),barAngularAcceleration.signals(2).values(:));
grid on; xlabel('Time (sec)'); ylabel('Y Component (ft-lb)')

subplot(3,1,3); plot(barAngularAcceleration.time(:),barAngularAcceleration.signals(3).values(:));
grid on; xlabel('Time (sec)'); ylabel('Z Component (ft-lb)')
sgtitle('Angular Acceleration of Bar - World Frame Components');
```

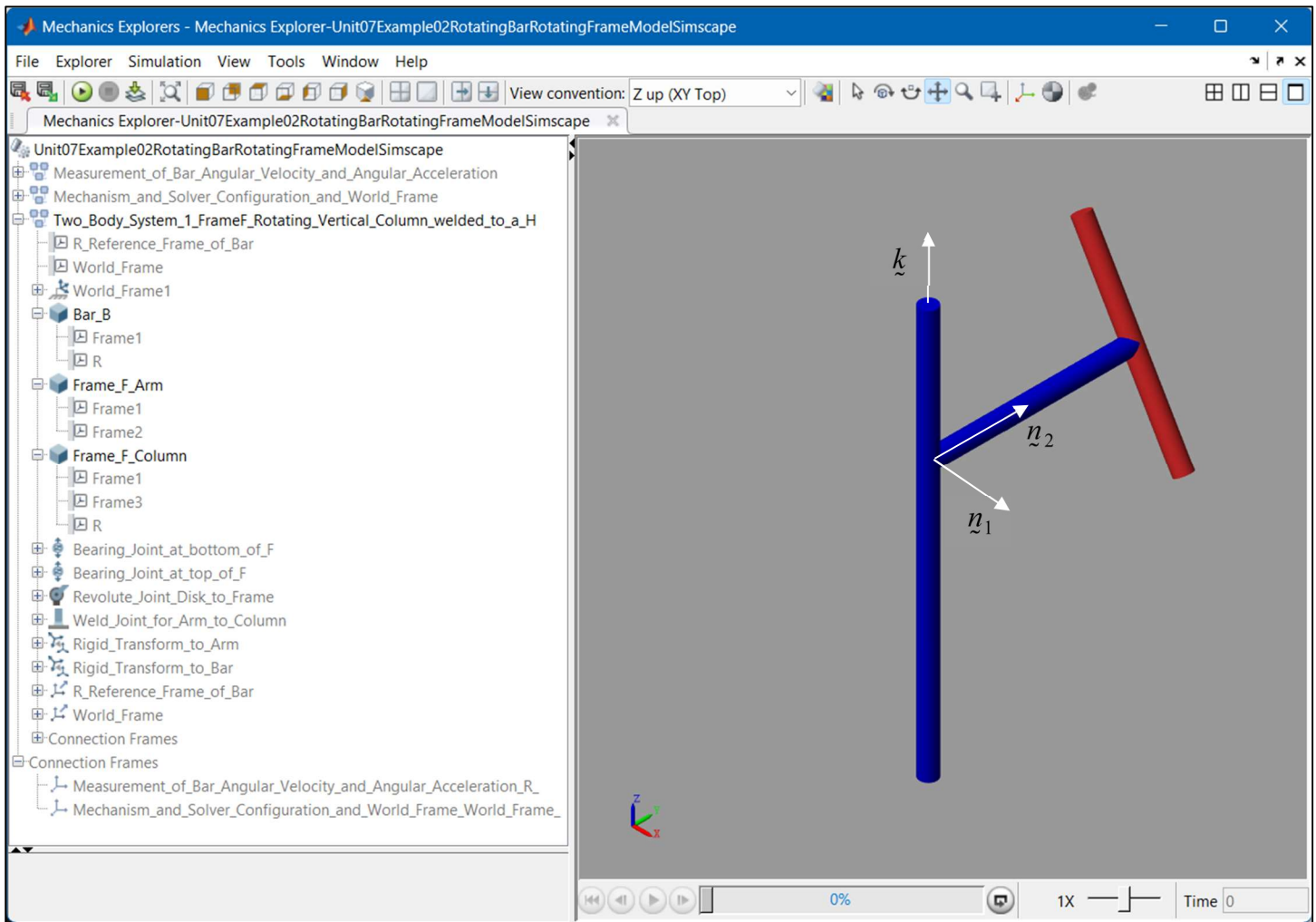
Simulation Results:

When the script is executed, a **geometric representation** of the motion of the system is **displayed** in the right half of the Mechanics Explorer window as shown below. As shown, the system is in its initial configuration ($t = 0$). As the simulation progresses, the motion of the system is animated. The unit vector set $F: (\underline{n}_1, \underline{n}_2, \underline{k})$ fixed in the frame has been added to the diagram for comparison with the original diagram. Note that the frame rotates about the about the world Z axis (\underline{k} direction), and the bar rotates relative to the frame about its horizontal arm (\underline{n}_2 direction).

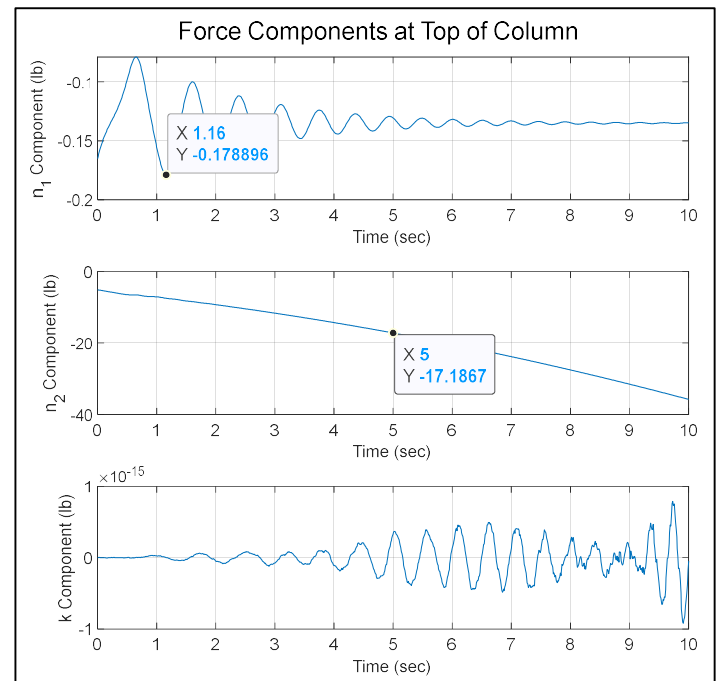
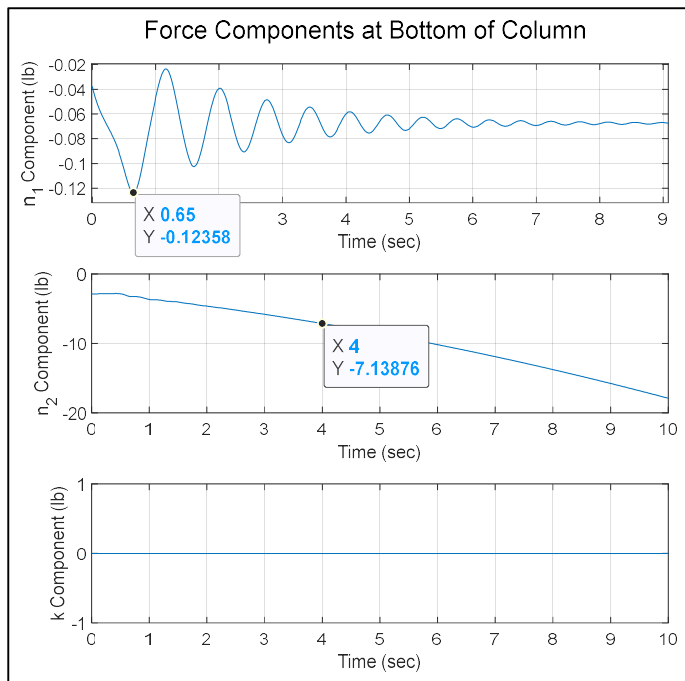
The first result shown to the right is $\theta(t)$ the angle of the bar relative to the horizontal. Note that as specified in the M-file, the angle starts at 60 (deg) and begins to increase (recall the initial value of $\dot{\theta}$ is 3 (rad/sec)). As the frame rotates, the simulation results show the bar is attracted to and finally **settles** in the **steady-state** position $\theta = 0$. The **existence** of this equilibrium position is easily **verified** by setting $\dot{\theta}$ and $\ddot{\theta}$ to **zero** in the differential equation of motion for the θ and **solving** the resulting **algebraic equation**.



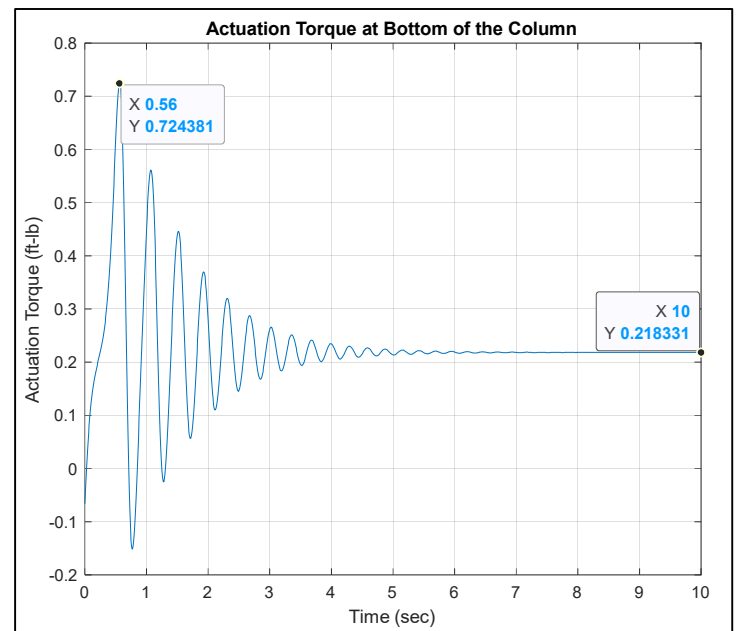
That process yields two equilibrium positions, one at $\theta = 0$ and one at $\theta = \frac{\pi}{2}$. It appears from these results that $\theta = 0$ is a stable equilibrium position.



The next set of results (shown below) are the *force components acting on the column* at the *lower* and *upper* bearing supports. Note the n_1 components of the forces *oscillate* and *settle* into relatively small steady state values, whereas the n_2 components *continue to increase* (in absolute value) as the angular speed of the frame increases. The steady state n_1 components likely depend *linearly* on the *angular acceleration* of the frame, while the steady state n_2 components likely depend *quadratically* on its *angular speed*. The *weight forces* were *not included* in the analysis, so the k components of the forces are *zero*.



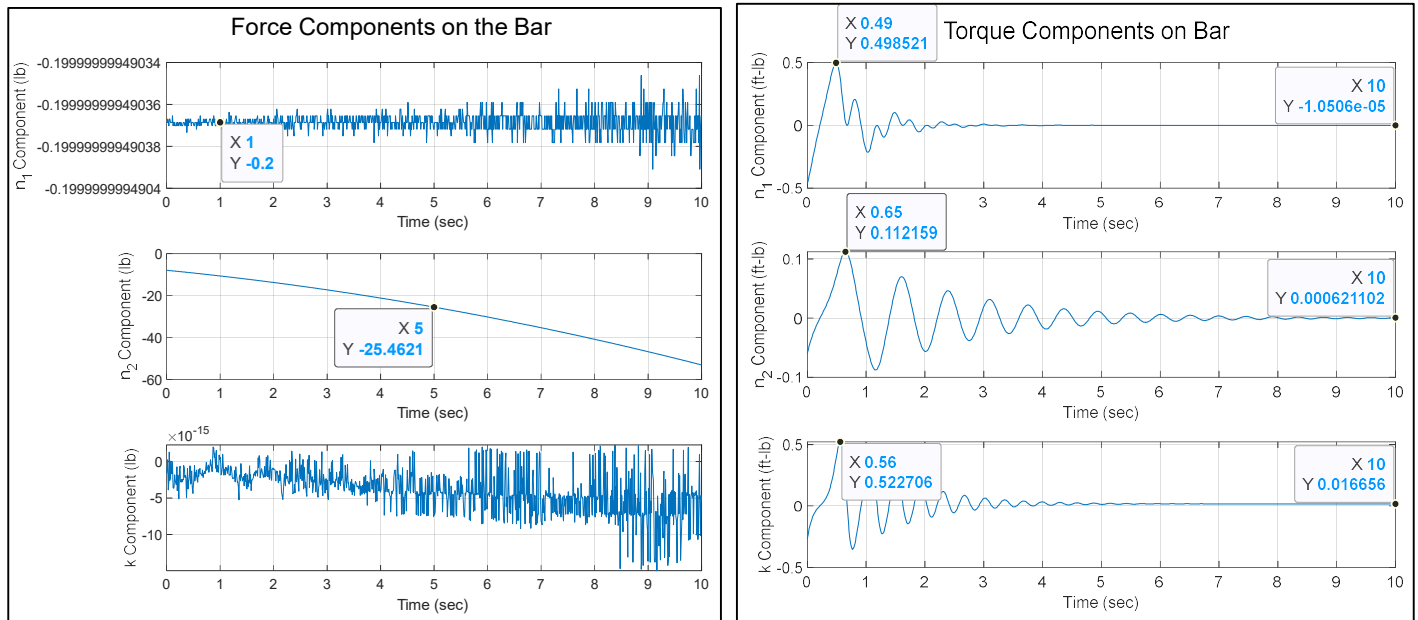
The *actuating torque acting on the column* at the *bottom support* is shown in the diagram to the right. Note that it *oscillates* and *settles* into a *steady state value* associated with the *constant angular acceleration* of the frame. The *bearing joints* only *restrict translational motion* of the frame (along the \underline{n}_1 and \underline{n}_2 directions), so the only *non-zero constraint torque* generated at these locations is at the *lower support* due to the *motion actuation*. The *torque components* at the *upper support* are all *zero*.



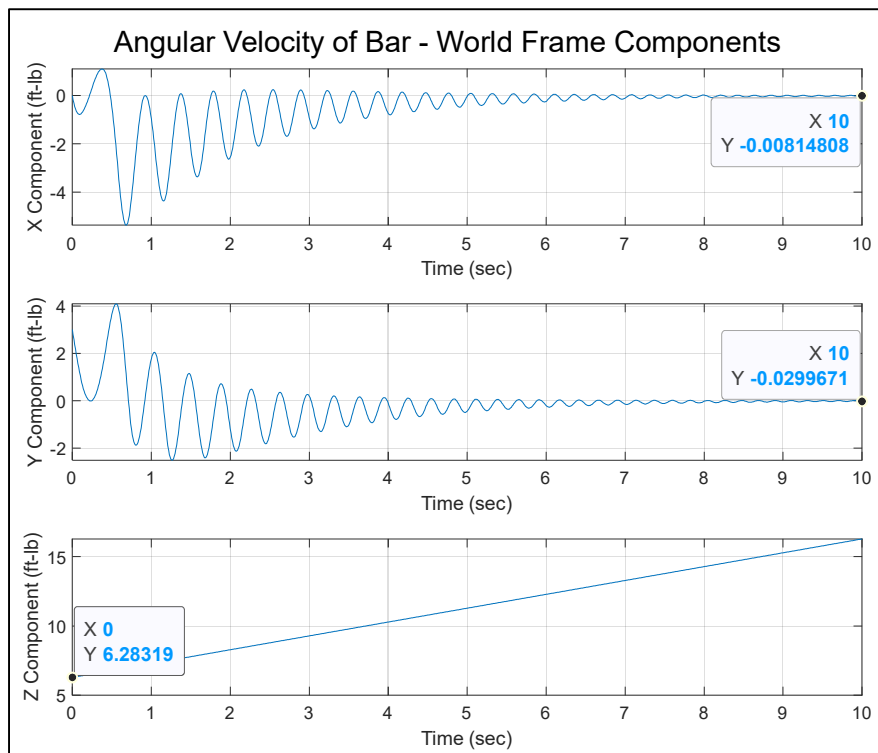
The *force* and *torque components* that the *frame exerts on the bar* through the pin at *G* are shown below. The components are resolved in the *F* frame. The force component in the \underline{n}_1 direction remains approximately *constant* at -0.2 (lb) while the \underline{n}_2 component continues to *increase* with the angular acceleration of *F*. The \underline{k} component is approximately zero.

The *torque component* in the \underline{n}_2 direction is the *damping torque* associated with the damper located between the frame and the bar at the revolute joint. This torque *linearly opposes* the angular velocity of the bar *relative* to the frame, so its value *oscillates* and *settles* to *zero* as the bar acquires its steady-state position relative to *F*. The *torque components* in the \underline{n}_1 and \underline{k} directions are *constraint torque* components. These components *initially*

oscillate and then settle to become constant. The n_1 component settles to *zero* while the k component settles to 0.01666 (ft-lb) as the bar reaches its steady-state position relative to F .



The *world components* of the *angular velocity* of bar B are shown in the plot below. As expected, the X and Y components *initially oscillate* but then *settle* to *zero* as the bar's motion relative to frame F settles to zero. The Z component starts 2π (r/s) consistent with the initial angular velocity of F . Its value increases with the constant angular acceleration of F .



Simulink Model:

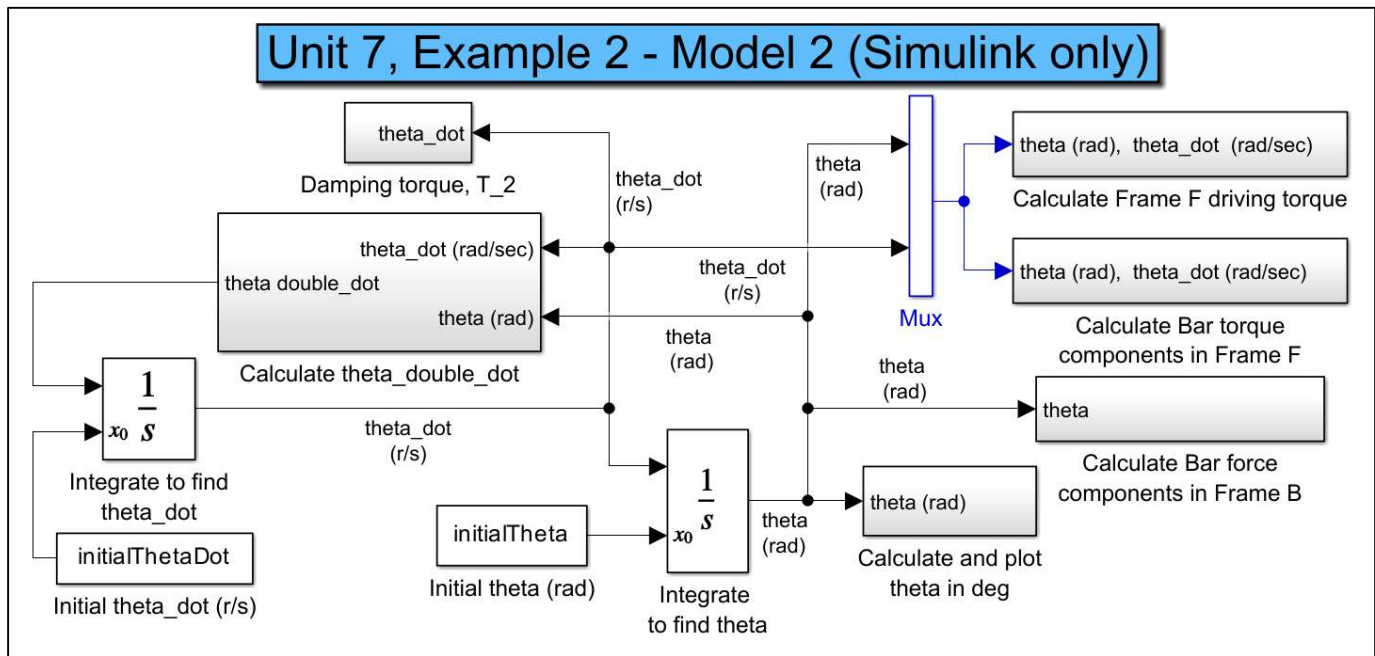
The **Simulink only** model is based strictly on the **analytical equations** presented above. Specifically, given frame F has **angular velocity** $\Omega(t)$, **constant angular acceleration** $\dot{\Omega}_0$, **damping torque** $T_2 = -b\dot{\theta}$, and initial values for θ and $\dot{\theta}$, it **solves the differential equation**

$$\ddot{\theta} + \Omega^2 S_{\theta} C_{\theta} = \frac{T_2}{\left(\frac{1}{12} m \ell^2\right)} = \frac{T_2}{I_{\text{bar}}}$$

to find $\theta(t)$. Then it uses the specified values of $\dot{\Omega}$, Ω , m , ℓ , and d to find the **unknown force** and **torque components** acting on the **bar** and the **driving torque** on the **frame**. The **angular motion** of the frame ($\dot{\Omega}$ and Ω), the **length** of the horizontal arm (d), the **mass** and **length** of the bar (m and ℓ), and the **damping coefficient** b are all as defined in Model 1. A MATLAB script controls the execution of the model.

The **overall structure** (upper layer) of the model is shown in Panel 1 below. Note in the lower left of the diagram there are two **integrators** (annotated by “ $\frac{1}{s}$ ”). The **first** integrates the signal $\ddot{\theta}(t)$ to get $\dot{\theta}(t)$, and the **second** integrates the signal $\dot{\theta}(t)$ to get $\theta(t)$. The **initial values** of $\dot{\theta}(t)$ and $\theta(t)$ are provided by the **workspace variables** “initialThetaDot” and “initialTheta”, respectively.

Simulink Model – Panel 1: (top layer with five subsystems)



Given **current values** for $\dot{\theta}$ and θ , the current value of $\ddot{\theta}$ is calculated using the differential equation. That is,

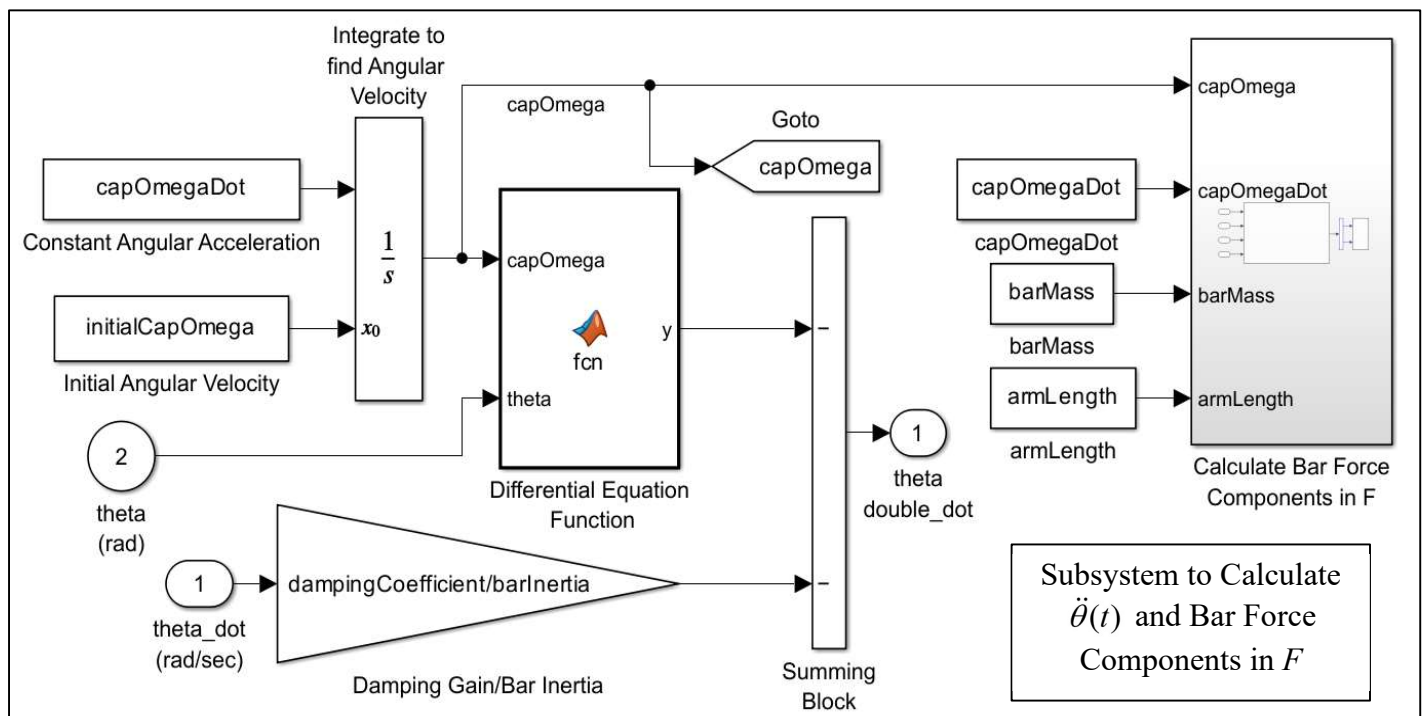
$$\ddot{\theta} = -\Omega^2 S_{\theta} C_{\theta} - \left(b/I_{\text{bar}}\right)\dot{\theta}$$

This calculation is done in the **subsystem** labelled “Calculate theta_double_dot”. The model has **six** other subsystems that use the signals $\theta(t)$ and $\dot{\theta}(t)$ to calculate and plot the **angle** $\theta(t)$ in degrees, the **damping torque** $T_2(t)$, the **bar torque** components resolved in F , the **bar force** components resolved in F , the **bar force** components resolved in B , and the **driving torque** applied to the frame.

The subsystem that calculates $\ddot{\theta}$ is shown in Panel 2 below. In the **upper left corner**, an **integrator** is used to **generate** the signal $\Omega(t)$ given its **initial value** Ω_0 and the **constant value** of $\dot{\Omega}$. The values of Ω_0 and $\dot{\Omega}$ are given by the **workspace variables** “initialCapOmega” and “capOmegaDot”, respectively. The resulting signal is sent to the MATLAB function labelled “Differential Equation” that calculates the signal “ $\Omega^2 S_{\theta} C_{\theta}$ ”. The **lower portion** of the block calculates the signal “ $(b/I_{\text{bar}})\dot{\theta}$ ”. The **negatives** of these two signals are added to give $\ddot{\theta}(t)$. Note that the signal $\Omega(t)$ is also **sent** to a “Goto” block so the signal is **available** at other **locations** in the model, and it is sent to a subsystem to calculate the force components that F applies to B at the revolute joint (resolved in F).

MATLAB **functions** are simply **scripts** used to perform the necessary calculations. The “Differential Equation Function” script is shown below.

Simulink Model – Panel 2: (subsystem to calculate $\ddot{\theta}$ and bar force components in F)

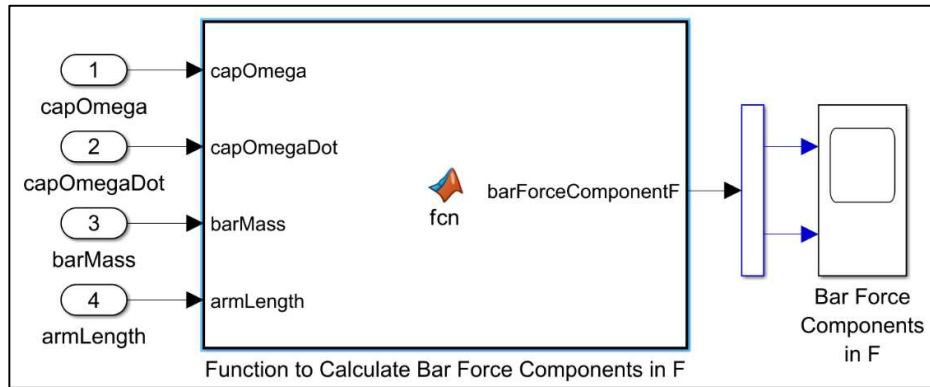


Differential Equation Function

```
function y = fcn(capOmega,theta)
y = (capOmega^2)*sin(theta)*cos(theta);
```

Details of the subsystems and function scripts that calculate the **bar force components** in F and B are shown in panel 3 below. Note the bar force arrays are sized and initialized using the **zeros** MATLAB function. Note also that the components in F depend only on the mass of B , length of the arm of F , and the applied motion to F . The components in B also depend on the angle θ . The components are sent to scopes to record the values for later plotting.

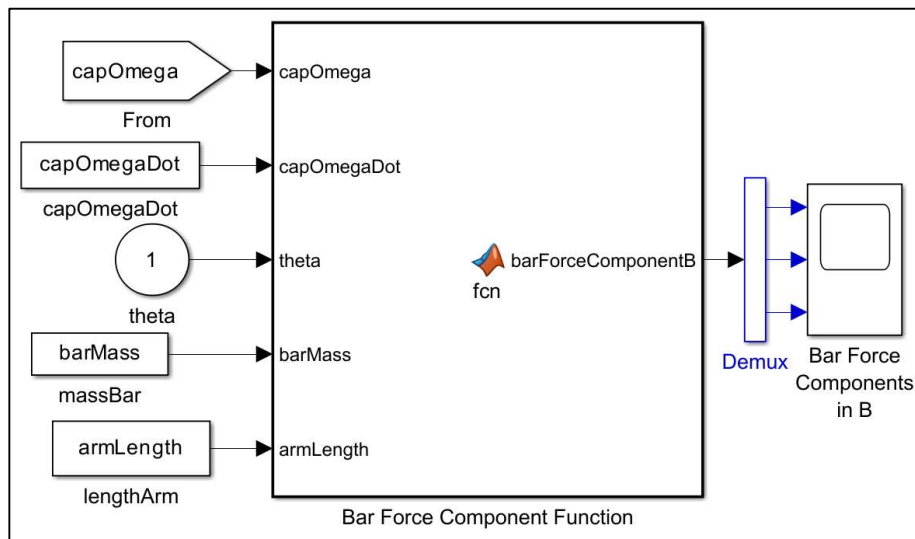
Simulink Model – Panel 3: (subsystems to calculate bar force components in F and B)



```
function barForceComponentF = fcn(capOmega, capOmegaDot, barMass, armLength)

barForceComponentF = zeros(2,1);

barForceComponentF(1) = -barMass*armLength*capOmegaDot;
barForceComponentF(2) = -barMass*armLength*(capOmega^2);
```



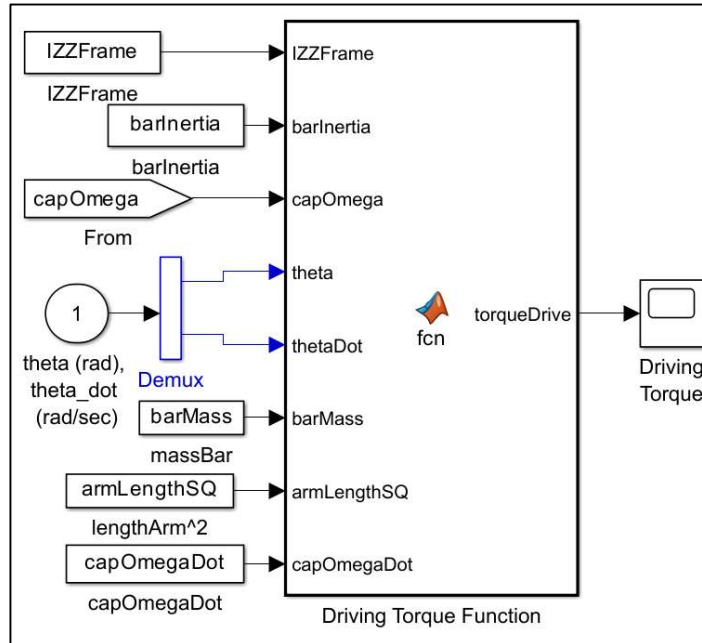
```
function barForceComponentB = fcn(capOmega, capOmegaDot, theta, barMass, armLength)

barForceComponentB = zeros(3,1);

barForceComponentB(1) = -barMass*armLength*capOmegaDot*cos(theta);
barForceComponentB(2) = -barMass*armLength*(capOmega^2);
barForceComponentB(3) = -barMass*armLength*capOmegaDot*sin(theta);
```

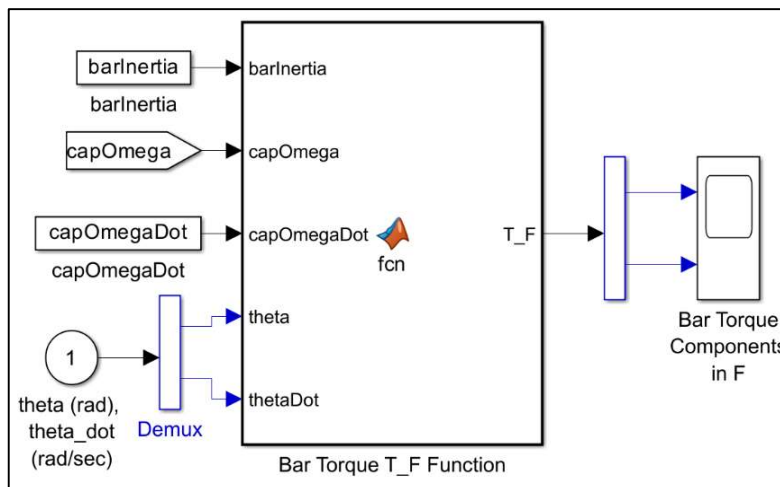
Details of the subsystems and function scripts that calculate the **actuating** (driving) **torque** applied to F and the **bar torque components** resolved in F are shown in Panel 4 below.

Simulink Model – Panel 4: (subsystems to calculate the actuating torque on F and constraint torque on B)



```
function torqueDrive = fcn(IZZFrame, barInertia, capOmega, theta, thetaDot, ...
    barMass, armLengthSQ, capOmegaDot)

torqueDrive = (IZZFrame*capOmegaDot) + (barMass*armLengthSQ*capOmegaDot) + ...
    (barInertia*((cos(theta))^2)*capOmegaDot) - ...
    (2*barInertia*sin(theta)*cos(theta)*thetaDot*capOmega);
```



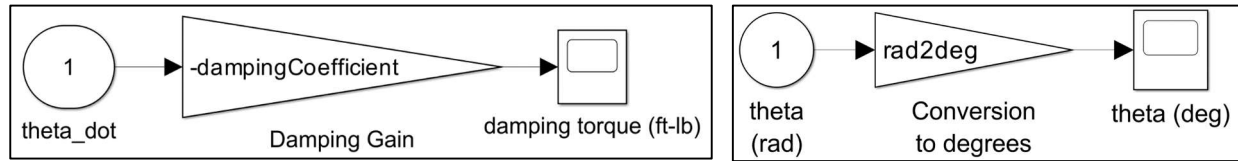
```
function T_F = fcn(barInertia, capOmega, capOmegaDot, theta, thetaDot)

T_F = zeros(2,1);

T_3 = barInertia*((capOmegaDot*cos(theta))-(2*capOmega*thetaDot*sin(theta)));

T_F(1) = T_3*sin(theta);
T_F(2) = T_3*cos(theta);
```

Details of the final two subsystems are shown below. The subsystem on the left calculates the linear damping torque, and the system on the right calculates the angle θ in degrees. In each case, the input signal is multiplied by a constant and sent to a scope for recording the signal.



MATLAB Script:

The *first three sections* of the MATLAB script are shown in Panel 1 below. The first section has a brief description of the *purpose* of the script, the second defines variables used for conversion of units, and the third defines geometric, mass, and moment of inertia data for frame F and bar B .

MATLAB Script – Panel 1: (statement of purpose, define workspace variables for use in the model)

```
%% Unit 7, Example 2 - Model 2

% This file defines the values of the input variables for a Simulink model
% of Unit 7, Example 2, executes the model, and plots results.
%
% The model uses Simulink to solve the model equations of the system
% dynamics. It assumes that the angular acceleration of the frame,
% F is constant. It also assumes the motion of the bar, B relative to F
% is free. The torque T_2 is assumed to be due to damping, only.

clear variables

%% units conversions

deg2rad      = pi/180; % deg to rad conversion
rad2deg      = 180/pi; % rad to deg conversion

%% Physical data for the frame, F and bar, B

% Column data
columnRadius = 0.5/12; % (ft)
columnMass   = 0.01;   % (slug)
IZZColumn    = (1/2)*columnMass*(columnRadius^2); % (slug-ft^2)

% Arm data
armLength    = 1.0;    % (ft)
armLengthSQ  = armLength^2; % (ft^2)
armMass      = 0.5*columnMass; % (slug)
IZZArm       = (1/3)*armMass*(armLength^2); % (slug-ft^2)

% Frame inertia about Z axis
IZZFrame = IZZColumn + IZZArm; % (slug-ft^2)

% Bar data
barMass    = 0.2; % (slugs)
barLength  = 1;   % (ft)
barInertia = (1/12)*barMass*(barLength^2); % (slug-ft^2)
:
:
```

Sections 4-6 of the script are shown in Panel 2 below. These sections define motion data for F , initial conditions for B , a damping coefficient for the revolute joint at the mass center of B , and then executes the Simulink model.

MATLAB Script – Panel 2: (motion data for F , initial conditions for B , damping coefficient, model execution)

```

                                :
%% Motion data for Frame, F

initialPhi      = 0.0; % (rad)
initialCapOmega = 2*pi; % (rad/sec)
capOmegaDot     = 1.0; % (rad/s^2)

%% Initial motion data for Bar, B

initialTheta     = 60.0*deg2rad; % (rad)
initialThetaDot  = 3.0;          % (r/s)
dampingCoefficient = 0.02;       % (ft-lb-s/rad)

%% Execute the Simulink model

sim('Unit07Example02Model2Simulink.slx')
                                :

```

The final section of the script is shown in Panel 3 below. This section displays plots of the actuating (driving) torque on F , the components of force applied to B by F resolved in both F and B frames, the components of torque applied to B by F resolved in F , the angle θ , and the damping torque applied to the bar in the η_2 direction.

Simulation Results:

The actuating (driving) torque on F , the force and torque components applied to B by F at the revolute joint, the angle θ , and the damping torque results produced by Model 2 (Simulink model) are all **identical** to those produced by Model 1 (Simscape Multibody model). Note that Model 2 does not include the forces acting on F at the bearing joints.

MATLAB Script – Panel 3: (plot the results)

```

:
%% Plot the time-varying results

% Plot the driving torque on frame, F
figure(9); clf;
plot(torqueDrive.time,torqueDrive.signals.values);
grid; title('Actuation Torque Acting on Frame, F');
xlabel('Time (sec)'); ylabel('Torque (ft-lb)')

% Plot the components of the force on the bar (expressed in F)
figure(10); clf;
subplot(2,1,1); plot(barForceComponentsF.time,barForceComponentsF.signals(1).values);
grid; xlabel('Time (sec)'); ylabel('n_1 Component (lb)')
subplot(2,1,2); plot(barForceComponentsF.time,barForceComponentsF.signals(2).values);
grid; xlabel('Time (sec)'); ylabel('n_2 Component (lb)')
sgtitle('Force Components on the Bar (expressed in F)');

% Plot the components of the force on the bar (expressed in B)
figure(11); clf;
subplot(3,1,1); plot(barForceComponentsB.time,barForceComponentsB.signals(1).values);
grid; xlabel('Time (sec)'); ylabel('e_1 Component (lb)')
subplot(3,1,2); plot(barForceComponentsB.time,barForceComponentsB.signals(2).values);
grid; xlabel('Time (sec)'); ylabel('e_2 (n_2) Component (lb)')
subplot(3,1,3); plot(barForceComponentsB.time,barForceComponentsB.signals(3).values);
grid; xlabel('Time (sec)'); ylabel('e_3 Component (lb)')
sgtitle('Force Components on the Bar (expressed in B)');

% Plot the torque components on the bar (expressed in F)
figure(12); clf;
subplot(2,1,1); plot(torqueBar.time,torqueBar.signals(1).values);
grid; xlabel('Time (sec)'); ylabel('n_1 Torque Component (ft-lb)')
subplot(2,1,2); plot(torqueBar.time,torqueBar.signals(2).values);
grid; xlabel('Time (sec)'); ylabel('k Torque Component (ft-lb)')
sgtitle('Torque Components Acting on Bar (expressed in F)');

% Plot the angle of the bar, theta
figure(13); clf;
plot(angleBar.time,angleBar.signals.values);
grid; title('Angle of the Bar, Theta');
xlabel('Time (sec)'); ylabel('Angle (deg)')

% Plot the damping torque in the n_2 direction at center of bar, B
figure(14); clf;
plot(torqueDamping.time,torqueDamping.signals.values);
grid; title('Damping Torque in n_2 direction Acting on Bar');
xlabel('Time (sec)'); ylabel('Torque (ft-lb)')
```

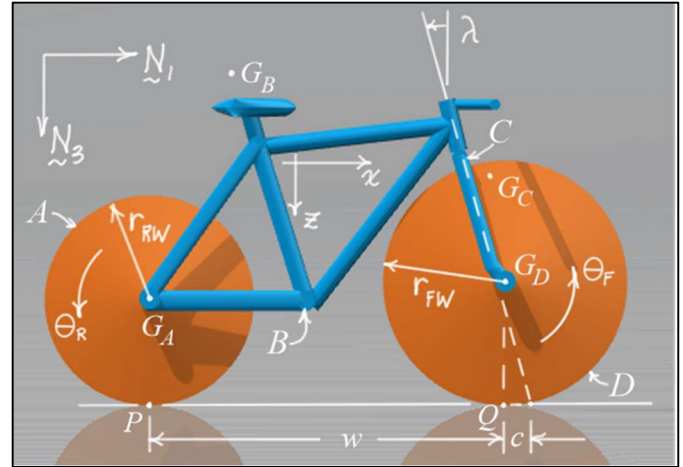
Example 3: Free Motion of an Upright, Two-Wheeled Bicycle

(See the bicycle references for more details)

This example includes *three models* of the *free motion* of an *upright bicycle*. The *first model* is a *MATLAB script* that uses MATLAB's *ODE45* to *integrate* the equations of motion (EOM). The *second* and *third models* consist of *MATLAB scripts* coupled with *Simulink models* to integrate the EOM. The EOM for the bicycle were developed using Kane's equations (or D'Alembert's Principle) in Unit 5 of this volume. The following sections provide information related to the *analytical model* of the bicycle including *definitions* of *variables*, *initial conditions*, and *geometric*, *mass*, and *inertia* data. Finally, details of the *three* numerical models are provided.

Analytical Model – Aligned Position Configuration:

The analysis that follows is for a *two-wheeled, upright* bicycle as shown. The bicycle is modeled as *four* interconnected bodies – the *rear wheel A*, the *rear frame and rider B*, the *steering column and fork C*, and the *front wheel D*. The bicycle is assumed to be *moving freely* on a horizontal surface with the rear wheel contacting the surface at point *P* and the front wheel contacting the surface at point *Q*. The distance between the contact points is the *wheelbase* *w*.



The steering column is assumed to be *tilted* relative to the vertical at an angle λ . This angle is a right-hand rotation about the $\underline{N}_2 = \underline{N}_3 \times \underline{N}_1$ direction. The *projection* of the steering axis onto the horizontal plane is assumed to be a distance *c* in front of the contact point *Q*. *Positive rotations* of the front and rear wheels are also measured about the \underline{N}_2 direction, so for forward motion of the bicycle, both angles θ_F and θ_R are *negative*.

In the configuration shown above, the *x* and *z* axes fixed in the rear frame *B* are *aligned* with the global (inertial) directions defined by the unit vector set $R: (\underline{N}_1, \underline{N}_2, \underline{N}_3)$. The unit vectors \underline{N}_1 and \underline{N}_2 are parallel to the horizontal plane, and the unit vector \underline{N}_3 points vertically downward.

The points G_A , G_B , G_C , and G_D are the *mass centers* of the bodies and are located using the following *geometric data* for the bicycle. Recall that body *B* includes both the *rear frame* and the *rider*. All the data given in the table are referenced to the *global directions*.

<i>w</i>	<i>c</i>	r_{RW}	r_{FW}	$\underline{r}_{G_A/P}$	$\underline{r}_{G_B/P}$	$\underline{r}_{G_C/P}$	$\underline{r}_{G_D/P}$	λ
1.02 (m)	0.08 (m)	0.3 (m)	0.35 (m)	$(0, -r_{RW})$	$(0.3, -0.9)$	$(0.9, -0.7)$	$(w, -r_{FW})$	18 (deg)

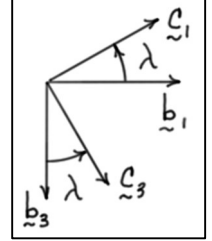
It can be shown using a *geometric analysis* that a *point S* located on the *steering axis* (common to both the rear frame and the steering column) can be located using the following equations.

Horizontal distance from G_A to S : $x_S = (w + c)C_\lambda^2 - r_{RW}S_\lambda C_\lambda$

Vertical distance from G_A to S : $z_S = x_S \tan(\lambda)$

Two additional sets of unit vectors are defined. The set $B: (\underline{b}_1, \underline{b}_2, \underline{b}_3)$ is fixed in the **rear frame** B , with \underline{b}_1 pointing in the x -direction (forward), \underline{b}_2 pointing in the y -direction (to the rider's right), and $\underline{b}_3 = \underline{b}_1 \times \underline{b}_2$.

In the figure above, the unit vectors of B are **aligned** with the unit vectors of R . A second set $C: (\underline{c}_1, \underline{c}_2, \underline{c}_3)$ is fixed in the **steering column** C . In the position where frame B is aligned with frame R , the unit vectors in frame C are directed as follows. Unit vectors \underline{c}_1 and \underline{c}_3 are at an angle λ with their counterparts in the B frame, and \underline{c}_2 is aligned with \underline{b}_2 .



For the analysis that follows, the **position vectors** of G_B and S relative to G_A are expressed in the B frame, and the position vectors of G_C and G_D relative to S are expressed in the C frame as follows.

$$\underline{r}_{G_B/G_A} = x_B \underline{b}_1 - z_B \underline{b}_3 \quad \underline{r}_{S/G_A} = x_S \underline{b}_1 - z_S \underline{b}_3$$

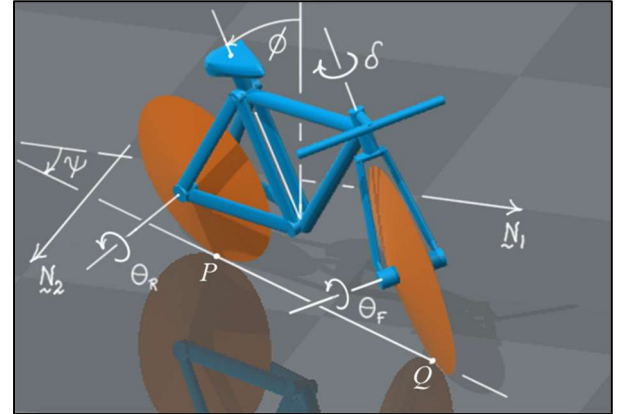
$$\underline{r}_{G_C/S} = x_C \underline{c}_1 + z_C \underline{c}_3 \quad \underline{r}_{G_D/S} = x_D \underline{c}_1 + z_D \underline{c}_3$$

Component	Value (m)
x_B	0.3
z_B	0.6
x_S	0.9067915590623501
z_S	0.2946344379171024
x_C	0.02610059280343250
z_C	-0.1023073115806087
x_D	0.03207142672761934
z_D	0.2676445084476887

The **fixed values** of the components of these vectors are provided in the adjacent table. The values are based on the data and formulae provided above.

Analytical Model – General Position Configuration:

The figure to the right shows the bicycle in a more **general configuration**. The figure illustrates **five** of the **six** angles used in the analysis. The rear frame B is oriented relative to the inertial frame R using a **3-1-2 orientation angle sequence**. The steering column C and rear wheel are each oriented relative to B by **single angles**, and the front wheel is oriented relative to C by a **single angle**.



The 3-1-2 orientation angles (ψ, ϕ, θ) orient the **rear frame** relative to R . They represent the **yaw**, **roll** (or lean), and **pitch** angles of B . The diagram shows **positive yaw** and **roll** (or lean) angles. The pitch angle is not shown. The value of the pitch angle is small and is determined to ensure the front wheel remains in contact with the horizontal surface. The diagram also shows a **positive steering angle** δ . So, for the angles shown, the bicycle is **rolling** and **turning** to the **right**. As noted earlier, the angles of the wheels relative to the frame are both **negative** for **forward motion**.

Analytical Model – Equations of Motion:

The *equations of motion* describing the *free motion* of the bicycle on a horizontal plane are *derived* and *presented* in Unit 5 of this volume. The equations are developed based on the *implicit assumption* that the point of the rear wheel that contacts the ground has *zero velocity*. This allows the equations to be written in terms of *six generalized coordinates*, the three orientation angles (ψ, ϕ, θ) of the rear frame, the steering angle (δ) , and the angles of spin (θ_R, θ_F) of the rear and front wheels relative to the frame.

As the bicycle has only *three degrees of freedom*, the application of Kane's equations (or D'Alembert's principle) provides only *three equations of motion*. These equations are supplemented with *three constraint equations* that require the *front wheel* remain in *contact* with, and *not slip*, on the horizontal plane. Finally, to track the *path* of the rear wheel contact point, the equations are supplemented with three equations that track the coordinates of that point. The x and y coordinates show the *path* of the contact point on the horizontal plane, and the z coordinate is provided as a numerical check, as its value should be *zero*.

For *full development* of the equations of motion, see Unit 5. Due to their length, the equations are *not* repeated here.

Analytical Model – Initial Conditions:

It is assumed that the roll angle ϕ , the rear wheel angle θ_R , and the steering angle δ form a set of three independent generalized coordinates. As independent coordinates, their *initial values* can be *specified independently*. The values of the rest of the coordinates must be chosen to be *consistent* with the *rolling constraints*.

The *distance* of the front wheel contact point Q from the horizontal surface is *independent* of the values of the x and y *coordinates* of the *rear contact point* P , the *yaw angle* of the bicycle (ψ) , and the *angle of the front wheel* relative to the front frame (θ_F) . Hence, their *initial values* can be *arbitrarily specified*. For *simplicity*, their *initial values* are all taken to be *zero*. That leaves the *initial value* of the *rear frame pitch* angle θ to be determined.

It is shown in Unit 5 that the following *non-linear, algebraic equation* can be *solved* to find the pitch angle θ , given *arbitrary values* of *roll angle* ϕ and *steering angle* δ .

$$\begin{aligned} & -\left(r_{RW} + x_S S_\theta + z_S C_\theta\right) C_\phi + x_D S_\phi S_\delta - x_D C_\phi C_\delta S_{\lambda+\theta} + z_D C_\phi C_{\lambda+\theta} \\ & + r_{FW} \left(\left(S_\phi S_\delta - C_\phi C_\delta S_{\lambda+\theta} \right)^2 + C_\phi^2 C_{\lambda+\theta}^2 \right)^{\frac{1}{2}} = 0 \end{aligned}$$

Note that this equation does not include θ_R the angle of the rear tire relative to the bicycle frame, and (it can be shown) that the *pitch angle* is *zero* if the *roll* and *steering angles* are *zero*.

Analytical Model – Mass and Inertia Data

The **masses** and **inertia matrices** of the four bodies are taken to be as follows.

$$\begin{aligned}
 & m_A = 2 \text{ (kg)} \quad m_B = 85 \text{ (kg)} \quad m_C = 4 \text{ (kg)} \quad m_D = 3 \text{ (kg)} \\
 & \begin{bmatrix} I_{G_A}^A \end{bmatrix} = \begin{bmatrix} I_{G_A}^B \end{bmatrix} = \begin{bmatrix} 0.0603 & 0 & 0 \\ 0 & 0.12 & 0 \\ 0 & 0 & 0.0603 \end{bmatrix} \text{ (kg-m}^2\text{)} \quad \begin{bmatrix} I_{G_B}^B \end{bmatrix} = \begin{bmatrix} 9.2 & 0 & 2.4 \\ 0 & 11 & 0 \\ 2.4 & 0 & 2.8 \end{bmatrix} \text{ (kg-m}^2\text{)} \\
 & \begin{bmatrix} I_{G_C}^B \end{bmatrix} = \begin{bmatrix} 0.05892 & 0 & -0.00756 \\ 0 & 0.06 & 0 \\ -0.00756 & 0 & 0.00708 \end{bmatrix} \text{ (kg-m}^2\text{)} \quad \begin{bmatrix} I_{G_D}^D \end{bmatrix} = \begin{bmatrix} I_{G_D}^B \end{bmatrix} = \begin{bmatrix} 0.1405 & 0 & 0 \\ 0 & 0.28 & 0 \\ 0 & 0 & 0.1405 \end{bmatrix} \text{ (kg-m}^2\text{)}
 \end{aligned}$$

To understand the **notation** used here for the **inertia matrices**, consider the matrix $\begin{bmatrix} I_{G_A}^B \end{bmatrix}$. This is the inertia matrix of body *A* (rear wheel) measured about its **mass center** in **directions** defined by reference frame $B: (\underline{b}_1, \underline{b}_2, \underline{b}_3)$.

Due to the **symmetry** of the wheel, this also represents $\begin{bmatrix} I_{G_A}^A \end{bmatrix}$ the inertia matrix about directions fixed in *A*.

Similarly, $\begin{bmatrix} I_{G_D}^B \end{bmatrix}$ the inertia matrix of body *D* (front wheel) measured about its mass center in directions defined by reference frame *B* is the same as $\begin{bmatrix} I_{G_D}^D \end{bmatrix}$ the inertia matrix about directions fixed in *D*.

The **inertias** of all the bodies are **referenced** to the **inertial directions** when the bicycle is in its **aligned position**. That is, they are referenced to the directions defined by the *B*-frame. Due to the **symmetries** of the **front** and **rear wheels**, **no transformations** are required to represent the inertia values in the local body directions. However, the inertia matrix $\begin{bmatrix} I_{G_C}^B \end{bmatrix}$ must be **transformed** into the frame $C: (\underline{c}_1, \underline{c}_2, \underline{c}_3)$ directions using a transformation matrix associated with a single rotation through the angle λ .

Using the results from Unit 1 of this volume, the **inertia matrix** for the **steering column** and **fork** (*B*) can be transformed as follows.

$$\begin{bmatrix} I_{G_C}^C \end{bmatrix} = \begin{bmatrix} C_\lambda & 0 & -S_\lambda \\ 0 & 1 & 0 \\ S_\lambda & 0 & C_\lambda \end{bmatrix} \begin{bmatrix} I_{G_C}^B \end{bmatrix} \begin{bmatrix} C_\lambda & 0 & S_\lambda \\ 0 & 1 & 0 \\ -S_\lambda & 0 & C_\lambda \end{bmatrix}$$

Model 1 – MATLAB Script Using ODE45 to Integrate the Equations of Motion

The MATLAB script consists of **two modules**. The **main** module defines a set of **global variables**, defines the **geometric** and **physical characteristics**, sets the **initial conditions**, **initializes** the **state vector**, calls the MATLAB function ODE45 to **integrate** the equations of motion over a specified time interval, and **plots** the results. As part of this process, the main module provides the name of the **function** that calculates the **derivative** of the **state vector**. This function is the **second module** of the script.

MATLAB Main Script – Panel 1: (operational description, clears workspace variables)

```
% This script solves the equations of motion (EOM) of a bicycle using ODE 45
%
% The state vector "y" for the bicycle is a 15x1 vector defined as
%
% y = [phi;thetaR;delta;psi;theta;thetaF;phidot;thetaRdot;deltaDot;
%      psiDot;thetaDot;thetaFdot;xP;yP;zP]
%
%      phi    - roll angle of rear frame
%      thetaR - rolling angle of the rear wheel
%      delta  - steering angle
%      psi    - yaw angle of rear frame
%      theta  - pitch angle of the rear frame
%      thetaF - rolling angle of the front wheel
%      xP,yP,zP - global coordinates of P (contact point of the rear wheel)
%
% The angles (phi,thetaR,delta) and their derivatives are taken as
% independent. The angles (psi,theta,thetaF) and their derivatives are taken
% as dependent. The EOM do not depend on the coordinates of P.
%
% Phase I of the script:
% 1. Defines the bicycle's geometric parameters
% 2. Defines the bicycle's mass and inertia properties
% 3. Defines the initial yaw angle (psi), the initial rear wheel rolling
%    angle (thetaR), and the initial front wheel rolling angle to all be
%    zero.
% 4. Given an initial rear-frame roll angle (phi) and steering angle
%    (delta), and initial rear-frame pitch angle is calculated to place
%    the front wheel in contact with the surface.
% 5. Initial values are specified for {u_I} the vector of initial
%    derivatives of (phi,thetaR,delta)
% 6. The bicycle transformation matrices, the C-frame components of N_3,
%    and the constraint equation matrices (C_1 and C_2) relating {u_D} the
%    derivatives of the dependent angles to {u_I} the derivatives of the
%    independent angles at the initial state are calculated.
% 7. Using the constraint equation matrices, {u_D} the derivatives of the
%    dependent angles are calculated.
% 8. The above results are used to define the initial state vector for the
%    bicycle.
%
% Phase II of the script:
% 1. Define the time space [tMin:tDelta:tMax] on which to present the
%    solution. Values of the elements of the state vector are provided at
%    each of the times within the time space.
% 2. Define a function handle for the function that computes dy/dt, the
%    derivative of the state vector
% 3. Call MATLAB function "ode45" to integrate the equations of motion on
%    the defined time interval.
%
% Phase III of the script:
%
% Plot the results of the integration process.
%
clear variables;
```

:

Main Script

Panel 1 above shows the **first section** of the main script that contains a brief functional description and clears the variables from the workspace. It notes that the state vector of the system is defined to be as follows.

$$y = \left[\phi \quad \theta_R \quad \delta \quad \psi \quad \theta \quad \theta_F \quad \dot{\phi} \quad \dot{\theta}_R \quad \dot{\delta} \quad \dot{\psi} \quad \dot{\theta} \quad \dot{\theta}_F \quad x_P \quad y_P \quad z_P \right]^T$$

Note that the **first twelve variables** are the **angles** and their **first derivatives**, and the **last three** are the global **coordinates** of point *P*. The **first three angles** are assumed to be **independent** variables, and the **last three** are

assumed to be *dependent* variables. The dependent variables are *related* to the independent ones using the *constraint equations* described above.

Panels 2 and 3 below show the next *nine sections* of the main script. The *first two sections* define a set of global variables and three constants used later in the script. The global variables are used in the “calculateLoopClosureEquation” and “bicycleEOM” functions. The *remaining sections* define a set of orientation angles for the starting configuration, initial coordinates of the rear wheel contact point P , a vector of independent generalized speeds, a set of bicycle geometric parameters, a set of bicycle mass and inertia properties, and a set of bicycle transformation matrices at the initial configuration.

The initial values of *five* of the *six* orientation angles are *chosen arbitrarily*. The initial value of the *sixth angle* (the pitch angle) is determined by *solving* the non-linear, algebraic *loop-closure equation* given above. This finds the *pitch angle* that forces the z coordinate of point Q to be *zero*. The solution is done using the MATLAB function “fzero”. The function “calculateLoopClosureEquation” calculates the z coordinate of Q as a function of the pitch angle. *Trigonometric functions* of the angles are also calculated for *later use*.

Panels 4-7 show details of the *functions* that define the bicycle’s *geometric parameters*, calculate the *loop closure equation*, define the bicycle *mass* and *inertia properties*, and calculate the bicycle *transformation matrices* at the initial angles.

Panel 8 shows the *next five sections* of the *main script*. The first finds the components of *unit vector* \underline{N}_3 in the C frame. The second uses those results to calculate the *constraint matrices* $[C_1]$, $[C_2]$, and $[J]$. In the next section, matrix $[J]$ is used to calculate values of the *dependent speeds* $(\dot{\psi}, \dot{\theta}, \dot{\theta}_F)$. The next section defines the system *state vector* at the *initial time* and displays the values in the MATLAB command window.

The last section in Panel 8 defines the parameters necessary to run the *differential equation solver* ODE45. First, it defines the *time grid* (or space) on which the solution will be calculated. The time grid has a starting value of “tMin”, a final value of “tMax”, and an increment of “tDelta”. ODE45 *numerically integrates* the equations of motion to find values of the *state vector* on this set of selected times, irrespective of the increment used in the numerical integration process. ODE45 may use smaller increments in this process to meet the *selected error criteria*. Next it defines the name of the MATLAB *function* that calculates the *derivative* of the *state vector* at each time step. In this example, the function name is “bicycleEOM”. Finally, before making the call to “ode45”, it defines options for the numerical integration process using the function “odeset”. In this example, the *relative error tolerance* (RelTol) is set to 1×10^{-5} and the *absolute error tolerance* (AbsTol) is set to 1×10^{-7} . See the MATLAB help feature for a description of these and other available options.

The call to “ode45” fills vector “t” and matrix “y”. In this case, “t” is a 401×1 vector of *time values* (the same values as in the vector “tSpace”), and “y” is a 2001×15 matrix of *state values*. Each *column* of “y” is the *time history* of one of the state variables. The first column is $\phi(t)$, the second is $\theta_R(t)$, the third is $\delta(t)$, etc.

MATLAB Main Script – Panel 2: (global variable definitions, acceleration of gravity, conversion factors)

```

:
%% Define global variables required for the analysis

global rRW rFW lambda xB zB xS zS xC zC xD zD
global sinLambda cosLambda
global sinPhi cosPhi sinDelta cosDelta
global mA mB mC mD aG
global inGA inGB inGC inGD

%% Define acceleration of gravity and radian and degree conversions
aG      = 9.81; % g (N/kg) or (m/s^2)
deg2Rad = pi/180; % conversion from degrees to radians
rad2Deg = 180/pi; % conversion from radians to degrees
:

```

MATLAB Main Script – Panel 3: (initial conditions; geometric, mass, inertia data; transformation matrices)

```

:
%% Define orientation angles for the bicycle and initial coordinates for P
phiDeg  = 0; % roll angle (deg)
deltaDeg = 0; % steering angle (deg)
phi      = phiDeg*deg2Rad; % roll angle(rad)
delta    = deltaDeg*deg2Rad; % steering angle(rad)
psi      = 0.0; % yaw angle (rad)
thetaR   = 0.0; % define initial thetaR to be zero (rad)
thetaF   = 0.0; % define initial thetaF to be zero (rad)

% initial coordinates for rear wheel contact point P
xP = 0.0; yP = 0.0; zP = 0.0;

%% Define u_I the vector of independent speeds
phiDot   = 0.50; % roll rate (rad/s)
thetaRDot = -15.3333333; % rear wheel spin rate(rad/s)
deltaDot  = 0.00; % steering rate(rad/s)
u_I       = [phiDot; thetaRDot; deltaDot];

%% Calculate sine and cosine of all angles, except theta
sinPsi    = sin(psi); cosPsi    = cos(psi);
sinPhi    = sin(phi); cosPhi    = cos(phi);
sinDelta  = sin(delta); cosDelta = cos(delta);

%% Define the geometric parameters of the bicycle
[rRW,rFW,w,c,lambda,xB,zB,xS,zS,xC,zC,xD,zD] = ...
    defineBicycleGeometricParameters(deg2Rad);
sinLambda = sin(lambda); cosLambda = cos(lambda);

%% Find the initial pitch angle, theta, that satisfies the loop closure equation
fun       = @calculateLoopClosureEquation;
theta0    = [-pi/10 pi/10];
theta     = fzero(fun,theta0);
sinTheta  = sin(theta); cosTheta  = cos(theta);
sinLambdaPTheta = sin(lambda+theta); cosLambdaPTheta = cos(lambda+theta);

%% Define the mass and inertia properties of the bicycle
[mA,mB,mC,mD,inGA,inGB,inGC,inGD] = ...
    defineBicycleMassInertiaProperties(sinLambda,cosLambda);

%% Calculate the bicycle transformation matrices...
[R_R2B,R_B2C,R_R2C] = calculateBicycleTransformationMatrices...
    (sinPsi,cosPsi,sinPhi,cosPhi,sinTheta,cosTheta,sinLambda,cosLambda,sinDelta,cosDelta);
:

```

MATLAB Script – Panel 4: (function to define bicycle's geometric parameters)

```
function [rRW,rFW,w,c,lambda,xB,zB,xS,zS,xC,zC,xD,zD] = defineBicycleGeometricParameters(deg2Rad)
% Define geometric parameters of the bicycle

rRW = 0.30; % (m)
rFW = 0.35; % (m)

w = 1.02; % (m)
c = 0.08; % (m)

lambdaDeg = 18; % fixed steering column tilt angle (deg)
lambda = lambdaDeg*deg2Rad; % fixed steering column tilt angle(rad)
sinLambda = sin(lambda); cosLambda = cos(lambda);

xB = 0.3; % (m)
zB = (0.9 - rRW); % (m)

xS = (((w + c)*cosLambda) - rRW*sinLambda)*cosLambda; % (m)
zS = xS*sinLambda/cosLambda; % (m)

tempMatrix = [cosLambda, 0, -sinLambda; 0, 1, 0; sinLambda, 0, cosLambda];

xHat_C = (0.9 - xS); zHat_C = (zS - (0.7 - rRW));
tempVector = tempMatrix*[xHat_C; 0; zHat_C];
xC = tempVector(1); zC = tempVector(3);

xHat_D = (w - xS); zHat_D = (rRW + zS - rFW);
tempVector = tempMatrix*[xHat_D; 0; zHat_D];
xD = tempVector(1); zD = tempVector(3);
```

MATLAB Script – Panel 5: (function to calculate loop closure equation)

```
function zQ = calculateLoopClosureEquation(theta)

global rRW rFW lambda xS zS xD zD
global sinPhi cosPhi sinDelta cosDelta

sinTheta = sin(theta); cosTheta = cos(theta);
sinLambdaPTheta = sin(lambda+theta); cosLambdaPTheta = cos(lambda+theta);

temp = rFW*sqrt( (((sinPhi*sinDelta)-(cosPhi*cosDelta*sinLambdaPTheta))^2) + ...
                ((cosPhi*cosLambdaPTheta)^2) );

zQ = -(rRW*cosPhi) - (xS*cosPhi*sinTheta) - (zS*cosPhi*cosTheta) + ...
      (xD*((sinPhi*sinDelta)-(cosPhi*cosDelta*sinLambdaPTheta))) + ...
      (zD*cosPhi*cosLambdaPTheta) + temp;
```

MATLAB Script – Panel 6: (function to define mass and inertia properties of the bicycle)

```
function [mA,mB,mC,mD,inGA,inGB,inGC,inGD] = defineBicycleMassInertiaProperties(sinLambda,cosLambda)
% All masses in (kg); All inertias in (kg-m^2); All inertia values given relative to global xyz;
% Inertia matrix for C is transformed into the C frame using the steering column angle lambda.

mA = 2.0; % mass
inGAxx = 0.0603; inGAyy = 0.12; inGAzz = inGAxx; % inertia values
inGA = [inGAxx, 0, 0; 0, inGAyy, 0; 0, 0, inGAzz]; % inertia matrix in B frame

mB = 85.0; % mass
inGBxx = 9.2; inGBxz = -2.4; inGByy = 11.0; inGBzz = 2.8; % inertia values
inGB = [inGBxx, 0, -inGBxz; 0, inGByy, 0; -inGBxz, 0, inGBzz]; % inertia matrix in B frame

mC = 4.0; % mass
R_B2Cprime = [cosLambda, 0, -sinLambda; 0, 1, 0; sinLambda, 0, cosLambda];
inGCxx = 0.05892; inGCxz = 0.00756; inGCyy = 0.06; inGCzz = 0.00708; % inertia values
inGC = [inGCxx, 0, -inGCxz; 0, inGCyy, 0; -inGCxz, 0, inGCzz]; % in global frame
inGC = R_B2Cprime*(inGC*R_B2Cprime'); % inertia matrix in C frame

mD = 3.0; % mass
inGDxx = 0.1405; inGDyy = 0.28; inGDzz = inGDxx; % inertia values
inGD = [inGDxx, 0, 0; 0, inGDyy, 0; 0, 0, inGDzz]; % inertia matrix in D frame
```

MATLAB Script – Panel 7: (function to calculate bicycle transformation matrices)

```
function [R_R2B,R_B2C,R_R2C] = calculateBicycleTransformationMatrices...
(sinPsi,cosPsi,sinPhi,cosPhi,sinTheta,cosTheta,sinLambda,cosLambda,sinDelta,cosDelta)

% Calculate transformation matrix from ground to rear frame B
R_R2B = zeros(3);
% first row
R_R2B(1,1) = (cosPsi*cosTheta) - (sinPsi*sinPhi*sinTheta);
R_R2B(1,2) = (sinPsi*cosTheta) + (cosPsi*sinPhi*sinTheta);
R_R2B(1,3) = -cosPhi*sinTheta;
% second row
R_R2B(2,1) = -sinPsi*cosPhi; R_R2B(2,2) = cosPsi*cosPhi; R_R2B(2,3) = sinPhi;
% third row
R_R2B(3,1) = (cosPsi*sinTheta) + (sinPsi*sinPhi*cosTheta);
R_R2B(3,2) = (sinPsi*sinTheta) - (cosPsi*sinPhi*cosTheta);
R_R2B(3,3) = cosPhi*cosTheta;

% Calculate transformation matrix from rear frame B to front frame C
R_B2C = zeros(3);
% first row
R_B2C(1,1) = cosLambda*cosDelta; R_B2C(1,2) = sinDelta; R_B2C(1,3) = -sinLambda*cosDelta;
% second row
R_B2C(2,1) = -cosLambda*sinDelta; R_B2C(2,2) = cosDelta; R_B2C(2,3) = sinLambda*sinDelta;
% third row
R_B2C(3,1) = sinLambda; R_B2C(3,2) = 0.0; R_B2C(3,3) = cosLambda;

% Calculate transformation matrix from ground to front frame C
R_R2C = R_B2C*R_R2B;
```

MATLAB Main Script – Panel 8: (N_3 ; constraint matrices; dependent speeds; initial state; ODE45 integration loop)

```

                                :
                                :
%% Define the components of N_3 in the C frame
[N_3] = findCFrameComponentsofN3(R_R2C);

%% Calculate the constraint matrices C_1, C_2, and J
[C_1,C_2] = calculateConstraintMatrices...
(rRW,rFW,xS,zS,xD,zD,sinPhi,cosPhi,sinTheta,cosTheta,N_3,R_B2C);

% Calculate the matrix J
J = -inv(C_2)*C_1;

%% Calculate u_D the vector of dependent speeds
u_D = J*u_I;
psiDot = u_D(1);
thetaDot = u_D(2);
thetaFDot = u_D(3);

%% Define the initial state vector, statey0
statey0 = [phi; thetaR; delta; psi; theta; thetaF;
           phiDot; thetaRDot; deltaDot; psiDot; thetaDot; thetaFDot;
           xP; yP; zP];
disp('Initial state vector, statey0')
disp('=====')
disp(statey0)

%% Integration loop using ode45
tMin = 0.0; % starting time (sec)
tMax = 20.0; % stopping time (sec)
tDelta = 0.01; % time increment for output vector
tSpace = tMin:tDelta:tMax; % solution space

functionName = @bicycleEOM;
options = odeset('RelTol',1e-5,'AbsTol',1e-7);
[t,y] = ode45(functionName,tSpace,statey0,options);
                                :
                                :
```

Panels 9-10 show the *functions* required to calculate the C frame components of unit vector \underline{N}_3 and the bicycle constraint matrices. Details of the function “bicycleEOM” are provided below.

Panel 11 below shows details required to *plot* the *time histories* of elements of the state vector. Plots 1-16 are plots of *single variables*, and plots 17-19 are plots of *multiple variables*. Note that angles are converted from radians to degrees before plotting.

MATLAB Script – Panel 9: (function to calculate the C frame components of \underline{N}_3)

```
function [N_3] = findCFrameComponentsofN3(R_R2C)

N_3 = zeros(3,1);
N_3(1) = R_R2C(1,3); N_3(2) = R_R2C(2,3); N_3(3) = R_R2C(3,3);
```

MATLAB Script – Panel 10: (function to calculate the constraint matrices)

```
function [C_1,C_2] = calculateConstraintMatrices...
    (rRW,rFW,xS,zS,xD,zD,sinPhi,cosPhi,sinTheta,cosTheta,N_3,R_B2C)
%% Calculate temporary variables for matrices C_1 and C_2

tempScalar1 = rRW + (xS*sinTheta) + (zS*cosTheta);
tempScalar2 = cosPhi*((xS*cosTheta) - (zS*sinTheta));
tempScalar3 = sqrt((N_3(1)^2) + (N_3(3)^2));
tempScalar4 = (rRW*cosTheta) + zS;
tempScalar5 = (rRW*sinTheta) + xS;

tempMatrix1 = [0, zD, 0; -zD, 0, xD; 0, -xD, 0];
tempMatrix2 = [cosTheta, 0, 0; 0, 0, 0; sinTheta, 0, 0];
tempMatrix3 = [0, N_3(3), 0; -N_3(3), 0, N_3(1); 0, -N_3(1), 0];
tempMatrix4 = [(-cosPhi*sinTheta), 0, 0; sinPhi, 1, 0;
    ( cosPhi*cosTheta), 0, 0];

%% Calculate the matrix C_1

tempMatrix = [0, -rRW*cosTheta, 0; tempScalar1, 0, 0;
    0, -rRW*sinTheta, 0];
C1A = R_B2C*tempMatrix;

tempMatrix = [0, 0, 0; 0, 0, 0; 0, 0, 1];
C1B = tempMatrix1*((R_B2C*tempMatrix2) + tempMatrix);

tempMatrix = [0, 0, 0; 0, 0, N_3(1); 0, 0, 0];
C1C = (tempMatrix3*R_B2C*tempMatrix2) + tempMatrix;
C1C = rFW*C1C/tempScalar3;
C_1 = C1A + C1B + C1C;

%% Calculate the matrix C_2

tempMatrix = [-(sinPhi*tempScalar4), -tempScalar4, 0;
    tempScalar2, 0, 0; -(sinPhi*tempScalar5), -tempScalar5, 0];
C2A = R_B2C*tempMatrix;

C2B = tempMatrix1*R_B2C*tempMatrix4;

tempMatrix = [0, 0, N_3(3); 0, 0, 0; 0, 0, -N_3(1)];
C2C = (tempMatrix3*R_B2C*tempMatrix4) + tempMatrix;
C2C = rFW*C2C/tempScalar3;
C_2 = C2A + C2B + C2C;
```

MATLAB Main Script – Panel 11: (plot results)

```

:
%% Plot the results
% Plot the yaw, roll, and pitch angles of the rear frame
figure(1); clf; temp = rad2Deg*y(:,4); plot(t,temp); grid on;
xlabel('Time (s)'); ylabel('Yaw angle (deg)'); title('Bicycle Yaw Angle (psi) vs. Time');

figure(2); clf; temp = rad2Deg*y(:,1); plot(t,temp); grid on;
xlabel('Time (s)'); ylabel('Roll angle (deg)'); title('Bicycle Roll Angle (phi) vs. Time');

figure(3); clf; temp = rad2Deg*y(:,5); plot(t,temp); grid on;
xlabel('Time (s)'); ylabel('Pitch angle (deg)'); title('Bicycle Pitch Angle (theta) vs. Time');

% Plot the steering angle
figure(4); clf; temp = rad2Deg*y(:,3); plot(t,temp); grid on;
xlabel('Time (s)'); ylabel('Steering angle (deg)'); title('Bicycle Steering Angle (delta) vs. Time');

% Plot the yaw rate, roll rate, and pitch rate of the rear frame
figure(5); clf; plot(t,y(:,10)); grid on;
xlabel('Time (s)'); ylabel('Yaw rate (rad/s)'); title('Bicycle Yaw Rate (psidot) vs. Time');

figure(6); clf; plot(t,y(:,7)); grid on;
xlabel('Time (s)'); ylabel('Roll rate (rad/s)'); title('Bicycle Roll Rate (phidot) vs. Time');

figure(7); clf; plot(t,y(:,11)); grid on;
xlabel('Time (s)'); ylabel('Pitch rate (rad/s)'); title('Bicycle Pitch Rate (thetadot) vs. Time');

% Plot the steering rate
figure(8); clf; plot(t,y(:,9)); grid on;
xlabel('Time (s)'); ylabel('Steering rate (rad/s)'); title('Bicycle Steering Rate (deltadot) vs. Time');

% Plot the rolling angle and rolling angle rate of the rear wheel
figure(9); clf; temp = rad2Deg*y(:,2); plot(t,temp); grid on; xlabel('Time (s)');
ylabel('Rolling Angle of Rear Wheel (deg)'); title('Rolling Angle of Rear Wheel (thetaR) vs. Time');

figure(10); clf; plot(t,y(:,8)); grid on; xlabel('Time (s)'); ylabel('Rolling Rate of Rear Wheel (rad/s)');
title('Rolling Rate of Rear Wheel (thetaRDot) vs. Time');

% Plot the rolling angle and rolling angle rate of the front wheel
figure(11); clf; temp = rad2Deg*y(:,6); plot(t,temp); grid on; xlabel('Time (s)');
ylabel('Rolling Angle of Front Wheel (deg)'); title('Rolling Angle of Front Wheel (thetaF) vs. Time');

figure(12); clf; plot(t,y(:,12)); grid on; xlabel('Time (s)'); ylabel('Rolling Rate of Front Wheel (rad/s)');
title('Rolling Rate of Front Wheel (thetaFDot) vs. Time');

% Plot the X,Y,Z coordinates of P and the XY path of P
figure(13); clf; plot(t,y(:,13)); grid on; xlabel('Time (s)'); ylabel('X-coordinate of P (m)');
title('Global X Coordinate of Rear Wheel Contact Point P');

figure(14); clf; plot(t,y(:,14)); grid on; xlabel('Time (s)'); ylabel('Y-coordinate of P (m)');
title('Global Y Coordinate of Rear Wheel Contact Point P');

figure(15); clf; plot(t,y(:,15)); grid on; xlabel('Time (s)'); ylabel('Z-coordinate of P (m)');
title('Global Z Coordinate of Rear Wheel Contact Point P');

figure(16); clf; plot(y(:,14),y(:,13)); grid on; axis('equal'); xlabel('Y-coordinate of P (m)');
ylabel('X-coordinate of P (m)'); title('Global Path of Rear Wheel Contact Point P');

%% Combined Plots
figure(17); clf; yawDeg = rad2Deg*y(:,4); rollDeg = rad2Deg*y(:,1); steerDeg = rad2Deg*y(:,3);
plot(t,yawDeg,'r',t,rollDeg,'g',t,steerDeg,'b'); grid on; xlabel('Time (s)'); ylabel('Angle (deg)');
legend('Yaw Angle','Roll Angle','Steering Angle','Location','east'); title('Bicycle Angles vs. Time');

figure(18); clf; yawRate = y(:,10); rollRate = y(:,7); steerRate = y(:,9);
plot(t,yawRate,'r',t,rollRate,'g',t,steerRate,'b'); grid on; xlabel('Time (s)'); ylabel('Angle Rate (r/s)');
legend('Yaw Angle Rate','Roll Angle Rate','Steering Angle Rate','Location','northeast');
title('Bicycle Angle Rates vs. Time');

figure(19); clf; rearWheelRate = y(:,8); frontWheelRate = y(:,12);
plot(t,rearWheelRate,'r',t,frontWheelRate,'g'); grid on; xlabel('Time (s)'); ylabel('Angle Rate (r/s)');
legend('Rear Wheel Angle Rate','Front Wheel Angle Rate','Location','east');
title('Bicycle Wheel Angle Rates vs. Time');
```

Function for Bicycle Equations of Motion: bicycleEOM

Details of the function “bicycleEOM” that *calculates* the *derivative* of the *state vector* are presented in the panels that follow. This function calls a *series* of *other functions* to complete the calculations.

Panel 1 shows the function statement and its *first two sections*. In the first section, *global variables* are defined. In the second section, values of the *elements* of the *state vector* are *unpacked*, the *independent* and *dependent speed vectors* are *defined*, and *trigonometric data* are calculated, all for later use. For *free motion* of the bicycle, the derivative of the state vector is a *function* of the state vector itself, but *not* explicitly time. Hence, the state vector is the only required argument of the function.

MATLAB Script for “bicycleEOM” Function – Panel 1: (global variables, state vector unload, trigonometric values)

```
function dydt = bicycleEOM(~,y)

    global rRW rFW lambda xB zB xS zS xC zC xD zD
    global sinLambda cosLambda
    global mA mB mC mD aG
    global inGA inGB inGC inGD

    %% Define the variables from the y-vector
    % angles
    phi = y(1); delta = y(3); % thetaR = y(2);
    psi = y(4); theta = y(5); % thetaF = y(6);
    % derivatives of the angles
    phiDot = y(7); thetaRDot = y(8); deltaDot = y(9);
    psiDot = y(10); thetaDot = y(11); thetaFDot = y(12);
    % coordinates of rear wheel contact point P
    % xP = y(13); yP = y(14); zP = y(15);
    % independent and dependent speed vectors
    u_I = [phiDot; thetaRDot; deltaDot];
    u_D = [psiDot; thetaDot; thetaFDot];
    % trigonometric data
    sinPsi = sin(psi); cosPsi = cos(psi);
    sinPhi = sin(phi); cosPhi = cos(phi);
    sinTheta = sin(theta); cosTheta = cos(theta);
    sinDelta = sin(delta); cosDelta = cos(delta);
    sinLambdaPTheta = sin(lambda+theta); cosLambdaPTheta = cos(lambda+theta);
    :
```

Panel 2 shows all the function calls required to *build* the *equations of motion*. The *first three function calls* to calculate the transformation matrices $[R_{R2B}]$, $[R_{B2C}]$, and $[R_{R2C}]$, find the *C*-frame components of the unit vector \underline{N}_3 , and calculate the constraint matrices $[C_1]$, $[C_2]$, and $[J]$ are the *same* as those required to find the *initial values* of the state vector discussed earlier. The *next four function calls* calculate the *partial angular velocity matrices* and *angular velocity vectors* of the bodies, and the *partial velocity matrices* and the *velocity vectors* of the *mass centers* of the bodies. The *eighth section* calculates the *generalized forces* associated with the weight forces of the bodies.

The next four sections *calculate* the *time derivatives* of the following – the *C*-frame components of \underline{N}_3 , the constraint matrices $[C_1]$, $[C_2]$, and $[J]$, the transformation matrix $[R_{B2C}]$, the partial angular velocity matrices of the bodies $[WA]$, $[WB]$, $[WC]$, and $[WD]$, and the partial velocity matrices of the mass centers of the body

$[VA]$, $[VB]$, $[VC]$, and $[VD]$. The *last two function calls* calculate the *generalized mass matrix* and the *right-side vector* of the differential equations of motion of the bicycle.

MATLAB Script for “bicycleEOM” Function – Panel 2: (function calls)

```

                                :
% Calculate the bicycle transformation matrices
[R_R2B,R_B2C,R_R2C] = calculateBicycleTransformationMatrices...
    (sinPsi,cosPsi,sinPhi,cosPhi,sinTheta,cosTheta,sinLambda,cosLambda,sinDelta,cosDelta);

% Define the components of N_3 in the C frame
[N_3] = findCFrameComponentsofN3(R_R2C);

% Calculate the constraint matrices C_1, C_2, and J
[C_1,C_2] = calculateConstraintMatrices...
    (rRW,rFW,xS,zS,xD,zD,sinPhi,cosPhi,sinTheta,cosTheta,N_3,R_B2C);

% Calculate the matrix J
J = -inv(C_2)*C_1;

% Calculate partial angular velocity matrices
[WA,WB,WC,WD] = calculatePartialAngularVelocities(sinPhi,cosPhi,sinTheta,cosTheta,R_B2C,J);

% Calculate the partial velocity matrices
[VA,VB,VC,VD] = calculateMassCenterPartialVelocities...
    (rRW,xB,zB,xS,zS,xC,zC,xD,zD,sinPhi,cosPhi,sinTheta,cosTheta,R_B2C,J);

% Calculate the angular velocities
[omegaA,omegaB,omegaC,omegaD] = calculateAngularVelocities(WA,WB,WC,WD,u_I);

% Calculate the velocities of the mass centers
[vGA,vGB,vGC,vGD] = calculateMassCenterVelocities(VA,VB,VC,VD,u_I);

% Calculate the generalized forces (due to weights)
[Fg] = calculateGeneralizedForces(VA,VB,VC,VD,R_R2B,R_R2C,mA,mB,mC,mD,aG);

% Calculate n31Dot and n33Dot the derivatives of N_3(1) and N_3(3)
[n31Dot,n33Dot] = calculateN3DotTerms(phiDot,deltaDot,thetaDot,...
    sinPhi,cosPhi,sinDelta,cosDelta,sinLambdaPTheta,cosLambdaPTheta);

% Calculate the derivatives of the constraint matrices
[C_1Dot,C_2Dot] = calculateConstraintMatricesDerivatives(phiDot,deltaDot,thetaDot,n31Dot,n33Dot,...
    rRW,rFW,xS,zS,xD,zD,sinPhi,cosPhi,sinLambda,cosLambda,sinDelta,cosDelta,...
    sinTheta,cosTheta,sinLambdaPTheta,cosLambdaPTheta,N_3);
JDot = -inv(C_2)*(C_1Dot + (C_2Dot*J));

% Calculate the derivatives of the partial angular velocity matrices
[WADot,WBDot,WCDot,WDDot,R_B2CDot] = calculatePartialAngularVelocityMatricesDerivatives(phiDot,deltaDot,thetaDot,...
    sinPhi,cosPhi,sinLambda,cosLambda,sinTheta,cosTheta,sinDelta,cosDelta,R_B2C,J,JDot,WB);

% Calculate the derivatives of the partial velocity matrices
[VADot,VBDot,VCDot,VDDot] = calculatePartialVelocityMatricesDerivatives...
    (phiDot,thetaDot,rRW,xB,zB,xS,zS,xC,zC,xD,zD,sinPhi,cosPhi,sinTheta,cosTheta,R_B2C,R_B2CDot,J,JDot);

% Calculate the generalized mass matrix, [M]
[gMassMatrix] = calculateGeneralizedMassMatrix(mA,mB,mC,mD,inGA,inGB,inGC,inGD,WA,WB,WC,WD,VA,VB,VC,VD);

% Calculate the RHS vector {f}
[fRHS,omegaBTilde] = calculateRightSideBicycleEOM(WA,WB,WC,WD,omegaA,omegaB,omegaC,omegaD,WADot,WBDot,WCDot,WDDot,...
    VA,VB,VC,VD,vGA,vGB,vGC,vGD,VADot,VBDot,VCDot,VDDot,mA,mB,mC,mD,inGA,inGB,inGC,inGD,thetaRDot,thetaFDot,u_I,Fg);
                                :

```

Panel 3 shows the *last two sections* of the function. The first of these *calculates* the *time derivatives* of the R -frame components of r_P the *position vector* of the *rear wheel contact point* P . As point P is to remain *in contact* with the surface at all times, the N_3 component is expected to be *approximately zero*, and the N_1 and N_2 components map out the *path* of P along the surface. The *last section* of the function *calculates* the *derivatives* of the *independent generalized speeds* from the generalized mass matrix “gMassMatrix” and the right-side vector “fRHS”. The *derivatives* of the *dependent generalized speeds* are *calculated* from the *independent speeds*, their *time derivatives*, and the *constraint matrix* $[J]$ and its time *derivative*. Finally, the independent speeds, the dependent speeds, the derivatives of the independent speeds, the derivatives of the dependent speeds, and the derivatives of the R -frame components of r_P are *concatenated* to form the *derivative* of the *state vector*.

MATLAB Script for “bicycleEOM” Function – Panel 3: (derivative of the state vector)

```

% Calculate the derivatives of the coordinates of the point P
tempVector = rRW*[-sinTheta; 0; cosTheta];
xPDot      = (R_R2B')*(vGA +(omegaBTilde*tempVector));

% Calculate the derivatives of the independent and dependent speeds
u_IDot = gMassMatrix\fRHS;
u_DDot = (J*u_IDot) + (JDot*u_I);

dydt = [u_I; u_D; u_IDot; u_DDot; xPDot];

```

Simulation Results

Initial Conditions

To *compare* with *previously published results* of free motion of the bicycle, the *initial conditions* are specified as in the following tables. The rear wheel is assumed to contact the surface at the origin, so the *initial coordinates* of P are $(0,0,0)$.

Angle	ϕ	θ_R	δ	ψ	θ	θ_F
Value (deg)	0	0	0	0	0	0

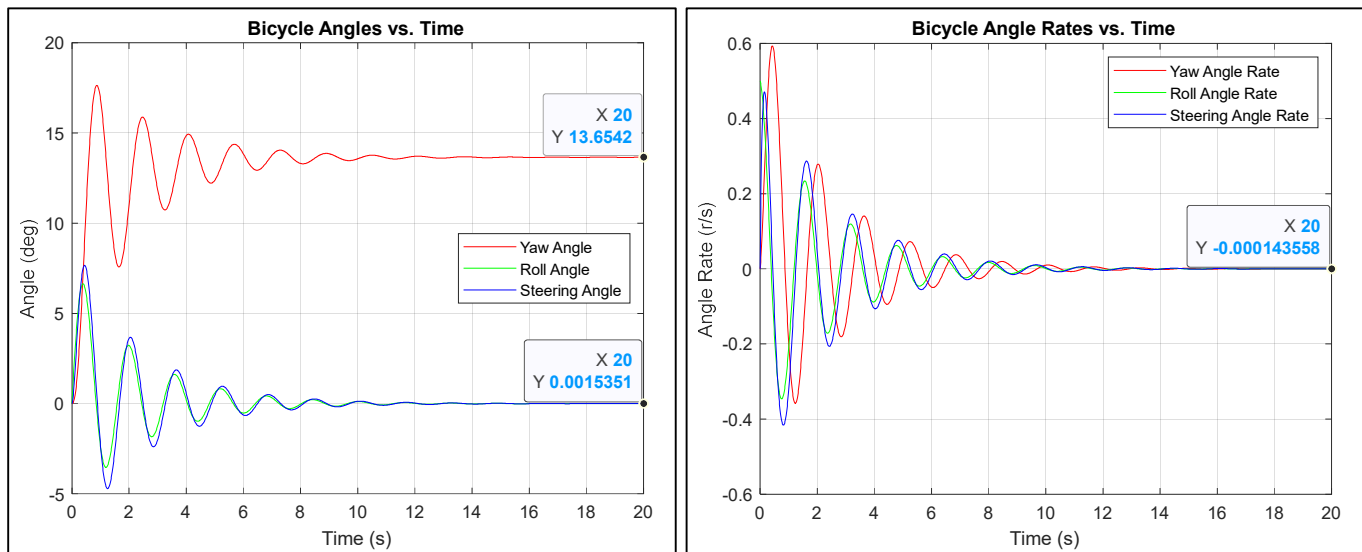
Angle Rate	$\dot{\phi}$	$\dot{\theta}_R$	$\dot{\delta}$	$\dot{\psi}$	$\dot{\theta}$	$\dot{\theta}_F$
Value (r/s)	0.5	-15.3333333	0	0	0	-13.1428571

The *initial pitch angle* θ is *calculated* from the *initial rear-frame roll angle*, the *initial steering angle*, and the *bicycle’s geometric parameters*. Given *zero* roll and steering angles, it is easy to show that the initial pitch angle is also *zero*. The angle rates $\dot{\phi}$, $\dot{\theta}_R$, and $\dot{\delta}$ are independently specified, and the rates $\dot{\psi}$, $\dot{\theta}$, and $\dot{\theta}_F$ are calculated from the independent rates using the constraint equations. The rate $\dot{\theta}_R = -15.3333333$ (r/s) corresponds to a *forward bicycle speed* of 4.6 (m/s).

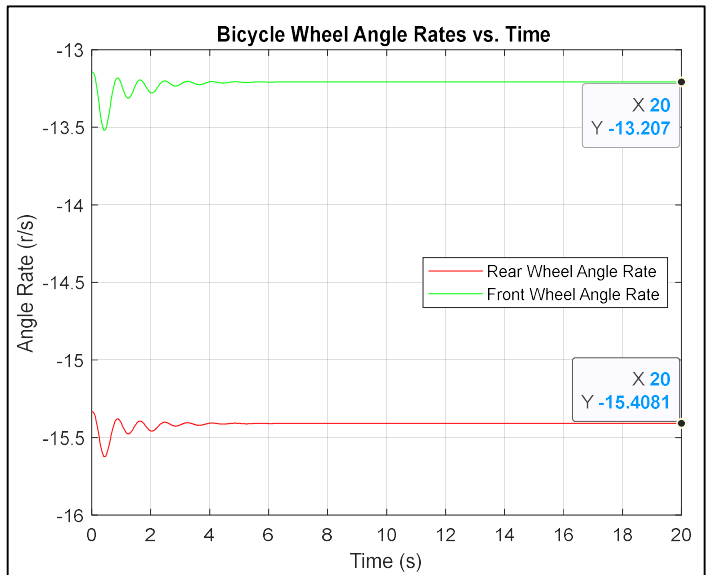
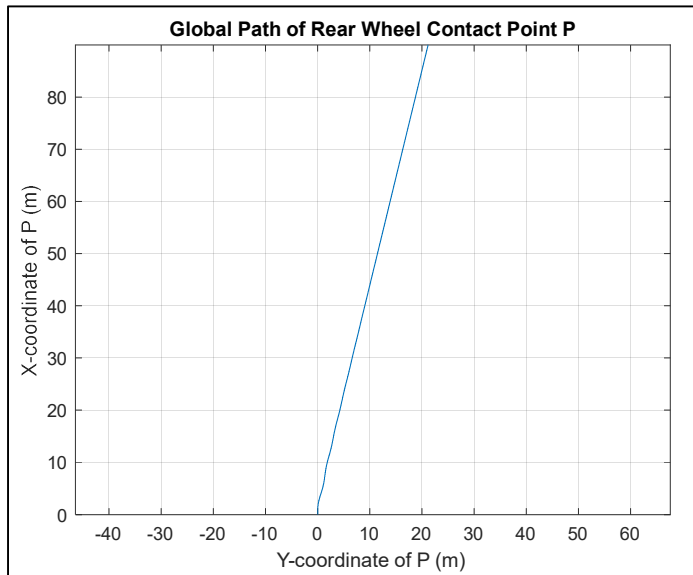
Results

Physically, the bicycle is initially *upright* and *moving forward* at 4.6 (m/s) when its motion is *perturbed* by a *roll rate* of 0.5 (r/s). The simulation results show *how* the bicycle *responds* with *no controlling input* from the rider. Note (from the results) that the motion is *stable* at this forward speed.

The *first set* of *plots* show time-histories of the *angles* ψ , ϕ , and δ and their *time derivatives*. The angles all exhibit *under-damped* responses and *nearly reach steady values* over a period of 20 seconds. The *yaw angle* is settling to a *non-zero steady-state value*, whereas the *roll* and *steering angles* are settling to *zero*. So, the bicycle *returns* to its *upright position* traveling in a slightly different direction, approximately 13.65 degrees to the right of its original direction. Note that *all angle rates* are settling to *zero*.



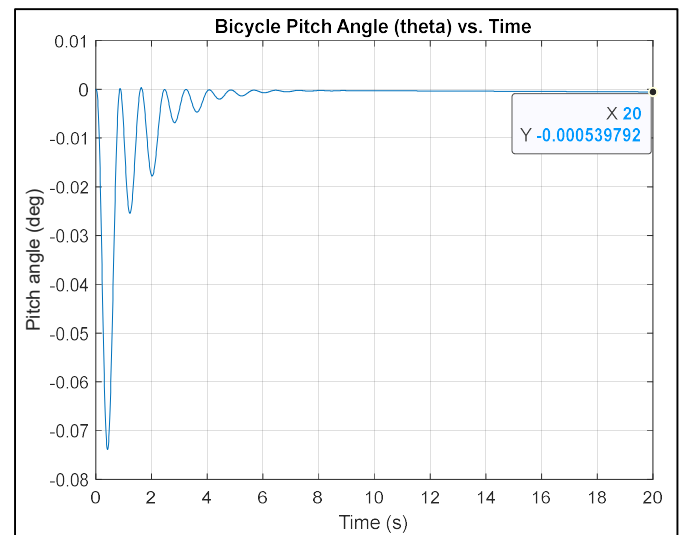
The *second set* of *plots* show the *path* of the *rear wheel contact point P* and the *angular rates* of the front and rear *wheels*. The plot on the left shows the *true shape* of the path of *P* and verifies the bicycle acquires straight-line motion roughly 14 degrees to the right of its original path. The plot on the right shows the *angular rates* of the *both wheels* also exhibit *under-damped* behavior that settles to a *new steady-state value*. The rear wheel starts at a rate of -15.3333333 (r/s) and settles to a final value of about -15.41 (r/s), and the front wheel starts at a rate of -13.1428571 (r/s) and settles to a final value of about -13.21 (r/s). This indicates the bicycle speeds up from 4.6 (m/s) to about 4.623 (m/s). As modeled, there are *no mechanisms* for *energy loss*, so the motion of the system must *conserve energy*. In this case, the *energy* of the *initial roll rate* is converted to a *higher forward speed*.



The *last plot* (to the right) shows the *pitch angle* of the rear frame. After 10 (s), the bicycle is nearly in an upright position, so the roll, pitch, and steering angles are all nearly zero. In *intermediate positions*, the *pitch angle* attains *small values* for *non-zero* roll and steering angles.

Validation

The results presented above are *identical* to those presented in the bicycle references listed below.



Model 2 – MATLAB Script Using Simulink to Integrate the Equations of Motion

Model 2 uses a MATLAB script very much like the script described above for Model 1. There are only *minor differences* between the two scripts. First, Model 2 calls the function “bicycleEOM” from its Simulink model, and not directly through the MATLAB script. Hence, the Model 2 script requires *global variables* to support only the function “calculateLoopClosureEquation”. The resulting global statements for the Model 2 script are shown here. No other global variables are required.

```
global rRW rFW lambda xS zS xD zD
global sinPhi cosPhi sinDelta cosDelta
```

Also, to provide the bicycle’s geometric, mass, and inertia data to the Simulink model, the Model 2 script defines the 1×11 array “geoParameters”, the 1×4 array “massParameters”, and the 3×12 array “inertiaParameters”. These additions are highlighted in **red** in Panel 1 below. The mass and inertia parameters represent the masses and inertia matrices of the four bicycle components. See the diagram above for the definitions of the bicycle components and the geometric parameters.

MATLAB Main Script – Panel 1: (define geometric parameters, masses, and inertia matrices)

```
% Define the geometric parameters of the bicycle

[rRW,rFW,w,c,lambda,xB,zB,xS,zS,xC,zC,xD,zD] = defineBicycleGeometricParameters(deg2Rad);
sinLambda = sin(lambda); cosLambda = cos(lambda);
geoParameters = [rRW,rFW,lambda,xB,zB,xS,zS,xC,zC,xD,zD];

% Find the initial pitch angle, theta

fun = @calculateLoopClosureEquation;
theta0 = [-pi/10 pi/10]; theta = fzero(fun,theta0);
sinTheta = sin(theta); cosTheta = cos(theta);
sinLambdaPTheta = sin(lambda+theta); cosLambdaPTheta = cos(lambda+theta);

% Define the mass and inertia properties of the bicycle

[mA,mB,mC,mD,inGA,inGB,inGC,inGD] = ...
    defineBicycleMassInertiaProperties(sinLambda,cosLambda);
massParameters = [mA,mB,mC,mD];
inertiaParameters = [inGA,inGB,inGC,inGD];
```

Finally, after the *starting time*, *stopping time*, and *time increment* are defined, the Model 2 script *executes* the Simulink model “**BicycleEOMSimulink01**”. Note that a time vector to define the solution space is not required.

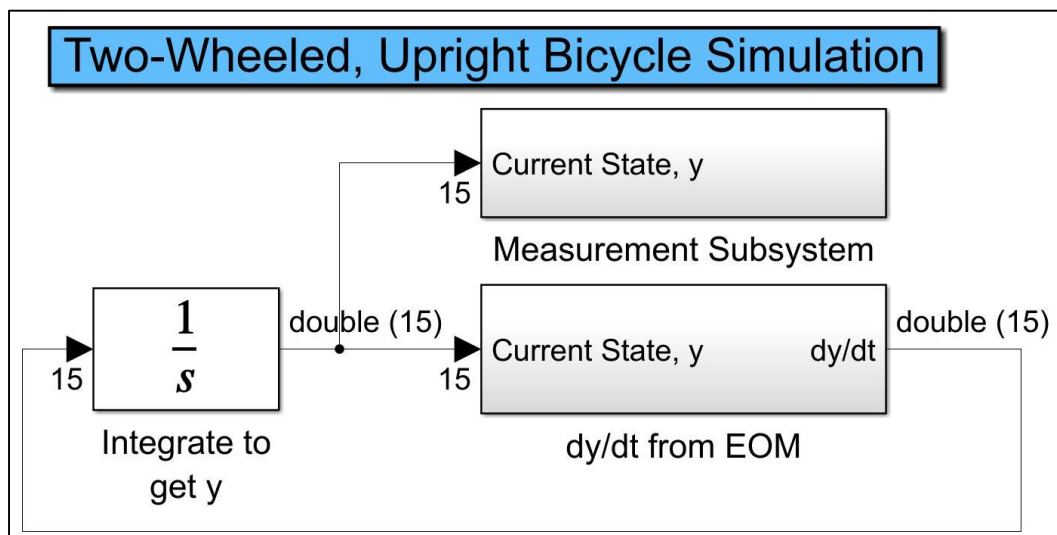
```
% Integration loop using ode45
tMin = 0.0; % starting time (sec)
tMax = 20.0; % stopping time (sec)
tDelta = 0.025; % time increment for output vector

sim('BicycleEOMSimulink01')
```

Simulink Model: “**BicycleEOMSimulink01**”

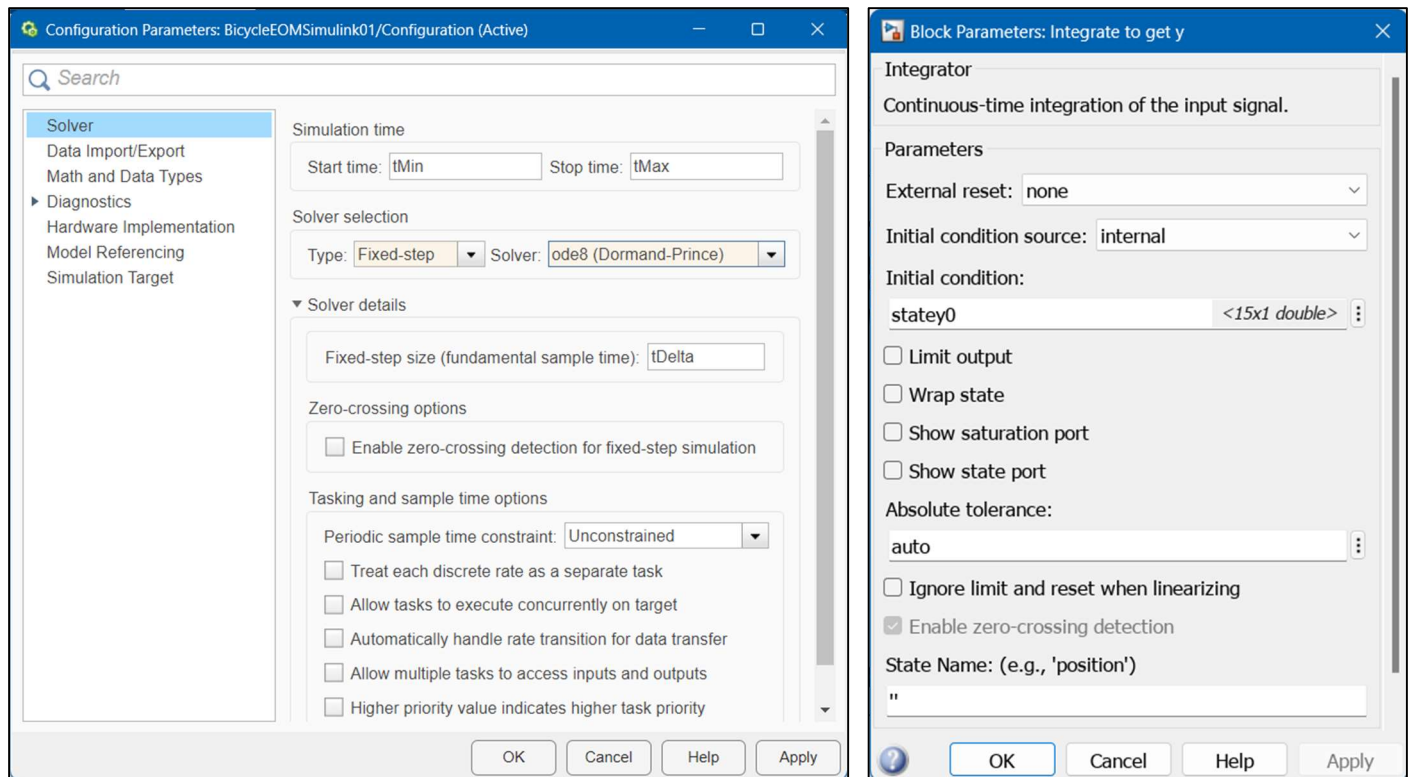
The *top layer* of the Simulink model is shown in Panel 1 below. The model structure is very simple. The lower loop *integrates* the *fifteen bicycle equations of motion*. The *initial state vector* is used to *specify* the *initial values* for the *numerical integration* and is used to *calculate* the *initial state vector derivative*. The state vector derivative is used to *estimate* the state vector at the next instant of time. This integration process continues from the *initial time* to the *final time* specified by the MATLAB script. Each new state vector is also sent to the measurement subsystem to save the results to the MATLAB workspace.

Simulink Model – Panel 1



The figure below on the left shows the model's *configuration parameters*. Specifically, it shows that the variables “tMin” and “tMax” provide values for the simulation *start time* and *stop time*, respectively. It also shows that the *numerical integration* is performed using a *fixed-step, Dormand-Prince, eighth-order method* with a *fixed step size* provided by the variable “tDelta”. The figure on the right shows the dialog box for the *integrator block*. The “Initial condition” section of the box indicates that the *initial conditions* of the fifteen state variables are provided by the 15×1 vector (or, array) “statey0”.

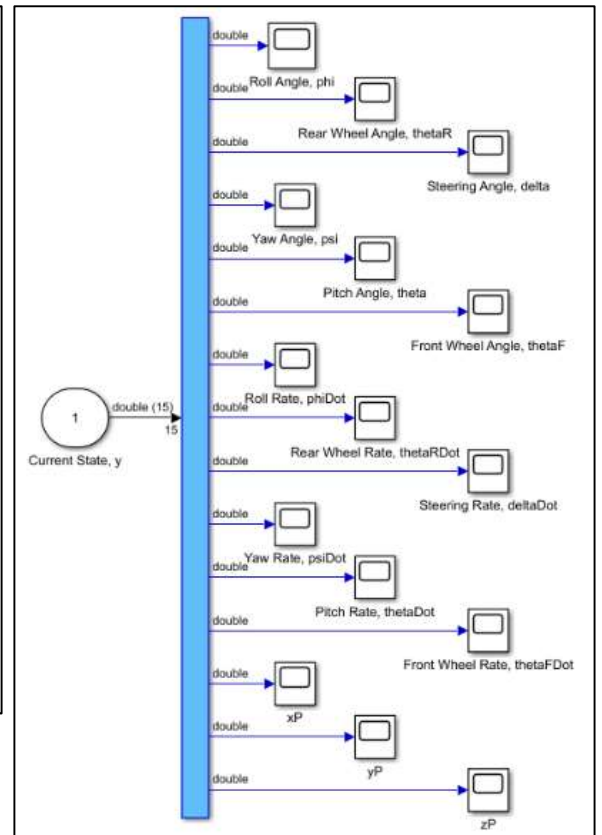
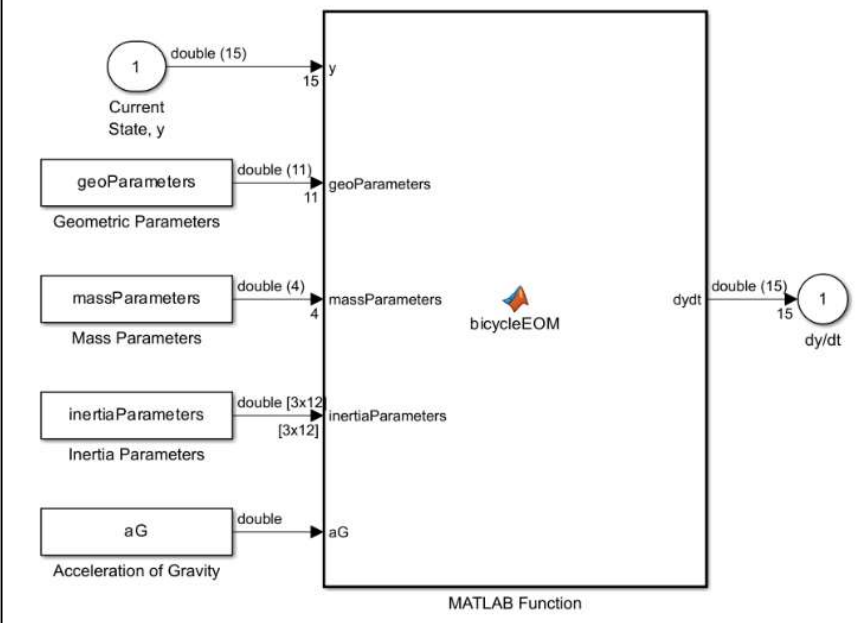
Note under the “Solver options” section of the model's configuration parameters dialog box that the solver “Type” and “Solver” are chosen using drop-down boxes. The solver type can be *fixed-step* or *variable-step*, and there are *many solvers* from which to choose. This is one of the *advantages* provided by Simulink.



Panel 2 below shows details of the *two subsystems*. The figure on the *left* shows details of the subsystem that calculates “dy/dt” the derivative of the state vector given the *current state vector*. In addition to the state vector, the *input signals* to this process are the *geometric parameters*, *masses*, and *inertia matrices* of the bicycle's components, and the *acceleration of gravity*. The derivative of the state vector is calculated in a single MATLAB function “bicycleEOM”. The figure on the right shows the *measurement subsystem* that records the current values of the state vector components in workspace arrays. The *names* of the workspace arrays are specified in the dialog boxes of the plotting scopes.

Simulink Model – Panel 2:

Calculate the Derivative of the State Vector given the Current State



The initial lines of the “bicycleEOM” function are shown in Panel 2 below. Note that elements of the state vector, geometric parameters, masses, inertia matrices, independent and dependent generalized speed vectors, and some trigonometric function values are *loaded* into the necessary variables for the calculations that follow. The rest of this function performs the *same calculations* as those described for the function of the same name in Model 1. The only difference is that the “bicycleEOM” function of Model 2 does not make any additional function calls – all calculations are done explicitly in this function.

To illustrate this point, the calculations done in the “bicycleEOM” function of Model 2 associated with the functions “calculateBicycleTransformationMatrices” and “findCFrameComponentsofN3” are shown in Panel 2. Note the calculations are not done by calling the two functions, but rather by doing the calculations explicitly within “bicycleEOM”. The remainder of the contents of the function are not shown here.

MATLAB Function Script – Panel 2: (Initial lines of function “bicycleEOM”)

```
function dydt = bicycleEOM(y,geoParameters,massParameters,inertiaParameters,aG)

%% Define the variables from the state vector y

% angles
phi = y(1); delta = y(3); psi = y(4); theta = y(5);

% derivatives of the angles
phiDot = y(7); thetaRDot = y(8); deltaDot = y(9); psiDot = y(10); thetaDot = y(11); thetaFDot = y(12);

% geometric parameters
% geoParameters = [rRW,rFW,lambda,xB,zB,xS,zS,xC,zC,xD,zD];
rRW = geoParameters(1); rFW = geoParameters(2); lambda = geoParameters(3);
xB = geoParameters(4); zB = geoParameters(5); xS = geoParameters(6); zS = geoParameters(7);
xC = geoParameters(8); zC = geoParameters(9); xD = geoParameters(10); zD = geoParameters(11);

% mass and inertia parameters
% massParameters = [mA,mB,mC,mD]; inertiaParameters = [inGA,inGB,inGC,inGD];
mA = massParameters(1); mB = massParameters(2); mC = massParameters(3); mD = massParameters(4);
inGA = inertiaParameters(:,1:3); inGB = inertiaParameters(:,4:6);
inGC = inertiaParameters(:,7:9); inGD = inertiaParameters(:,10:12);

% Independent and Dependent speeds
u_I = [phiDot; thetaRDot; deltaDot]; u_D = [psiDot; thetaDot; thetaFDot];

% trigonometric functions of the angles
sinPsi = sin(psi); cosPsi = cos(psi); sinPhi = sin(phi); cosPhi = cos(phi);
sinTheta = sin(theta); cosTheta = cos(theta);
sinDelta = sin(delta); cosDelta = cos(delta); sinLambda = sin(lambda); cosLambda = cos(lambda);
sinLambdaPTheta = sin(lambda+theta); cosLambdaPTheta = cos(lambda+theta);

%% Calculate the bicycle transformation matrices

% [R_R2B,R_B2C,R_R2C] = calculateBicycleTransformationMatrices;
% R_R2B = calculateTransformationMatrixFromAngles312;
% Compute the transformation matrix
R_R2B = zeros(3);

% first row
R_R2B(1,1) = (cosPsi*cosTheta) - (sinPsi*sinPhi*sinTheta);
R_R2B(1,2) = (sinPsi*cosTheta) + (cosPsi*sinPhi*sinTheta); R_R2B(1,3) = -cosPhi*sinTheta;

% second row
R_R2B(2,1) = -sinPsi*cosPhi; R_R2B(2,2) = cosPsi*cosPhi; R_R2B(2,3) = sinPhi;

% third row
R_R2B(3,1) = (cosPsi*sinTheta) + (sinPsi*sinPhi*cosTheta);
R_R2B(3,2) = (sinPsi*sinTheta) - (cosPsi*sinPhi*cosTheta); R_R2B(3,3) = cosPhi*cosTheta;

% R_B2C = calculateTransformationMatrixFromAngles23;
% Compute the transformation matrix
R_B2C = zeros(3);

% first row
R_B2C(1,1) = cosLambda*cosDelta; R_B2C(1,2) = sinDelta; R_B2C(1,3) = -sinLambda*cosDelta;
% second row
R_B2C(2,1) = -cosLambda*sinDelta; R_B2C(2,2) = cosDelta; R_B2C(2,3) = sinLambda*sinDelta;
% third row
R_B2C(3,1) = sinLambda; R_B2C(3,2) = 0.0; R_B2C(3,3) = cosLambda;

R_R2C = R_B2C*R_R2B;

%% Define the components of N_3 in the C frame

% [N_3] = findCFrameComponentsofN3(R_R2C);

N_3 = zeros(3,1);
N_3(1) = R_R2C(1,3); N_3(2) = R_R2C(2,3); N_3(3) = R_R2C(3,3);
      :
```

As the Simulink model executes, state vector data is *saved* to *arrays* in the MATLAB workspace. The *first column* of each array contains the *time values*, and the *second column* contains the values of the *state variable*. Panel 3 below shows the plot commands associated with the *first three figures*. Note that the angles are *converted* from radians to degrees before plotting.

MATLAB Main Script – Panel 3: (plot commands for the first three figures)

```
%% Plot the results

% Plot the yaw, roll, and pitch angles of the rear frame
figure(1); clf;
plot(psi(:,1),rad2Deg*psi(:,2)); grid on;
xlabel('Time (s)'); ylabel('Yaw angle (deg)');
title('Bicycle Yaw Angle (psi) vs. Time');

%
figure(2); clf;
plot(phi(:,1),rad2Deg*phi(:,2)); grid on;
xlabel('Time (s)'); ylabel('Roll angle (deg)');
title('Bicycle Roll Angle (phi) vs. Time');

%
figure(3); clf;
plot(theta(:,1),rad2Deg*theta(:,2)); grid on;
xlabel('Time (s)'); ylabel('Pitch angle (deg)');
title('Bicycle Pitch Angle (theta) vs. Time');
:
:
```

Model 2 Simulation Results:

The simulation results for Model 2 are *identical* to those presented above for Model 1.

Model 3 – MATLAB Script Using Simulink to Integrate the Equations of Motion

Model 3 is identical to Model 2, *except* in the way it calculates “dy/dt”. The *single function script* of Model 2 is *replaced* by a *detailed Simulink model* with *multiple function calls*. This makes the structure of the algorithm required to compute “dy/dt” more *explicit* and *observable* within the model.

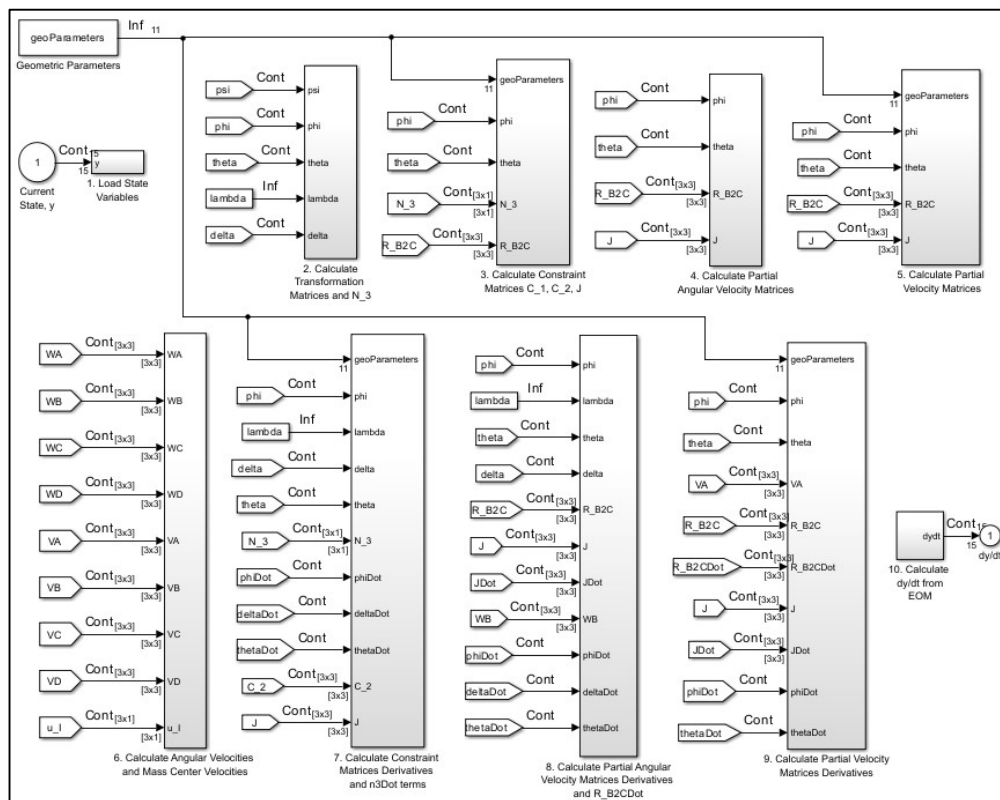
The MATLAB script and the top layer of the Simulink portion of Model 3 are identical to those of Model 2. The differences are in the subsystem required to calculate “dy/dt”. Panel 1 below shows the details of this subsystem. In this model, the work of the subsystem is *divided* into *ten additional subsystems*, labeled one through ten in the diagram. Details of each of these subsystems are shown below.

The contents of the *first two subsystems* are shown below in Panel 2. The first subsystem (diagram on the left) *loads variables* from the current state vector, then it defines the 3×1 vectors of *independent* and *dependent generalized speeds*. Note that some elements of the state vector are *not needed* to compute the equations of motion, so their signals are *terminated*. The second subsystem *calculates* and *stores* the 3×3 bicycle transformation matrices and the *C-frame components* of the *unit vector* \underline{N}_3 . Details of the MATLAB functions are not shown here. These calculations are the same as those presented earlier.

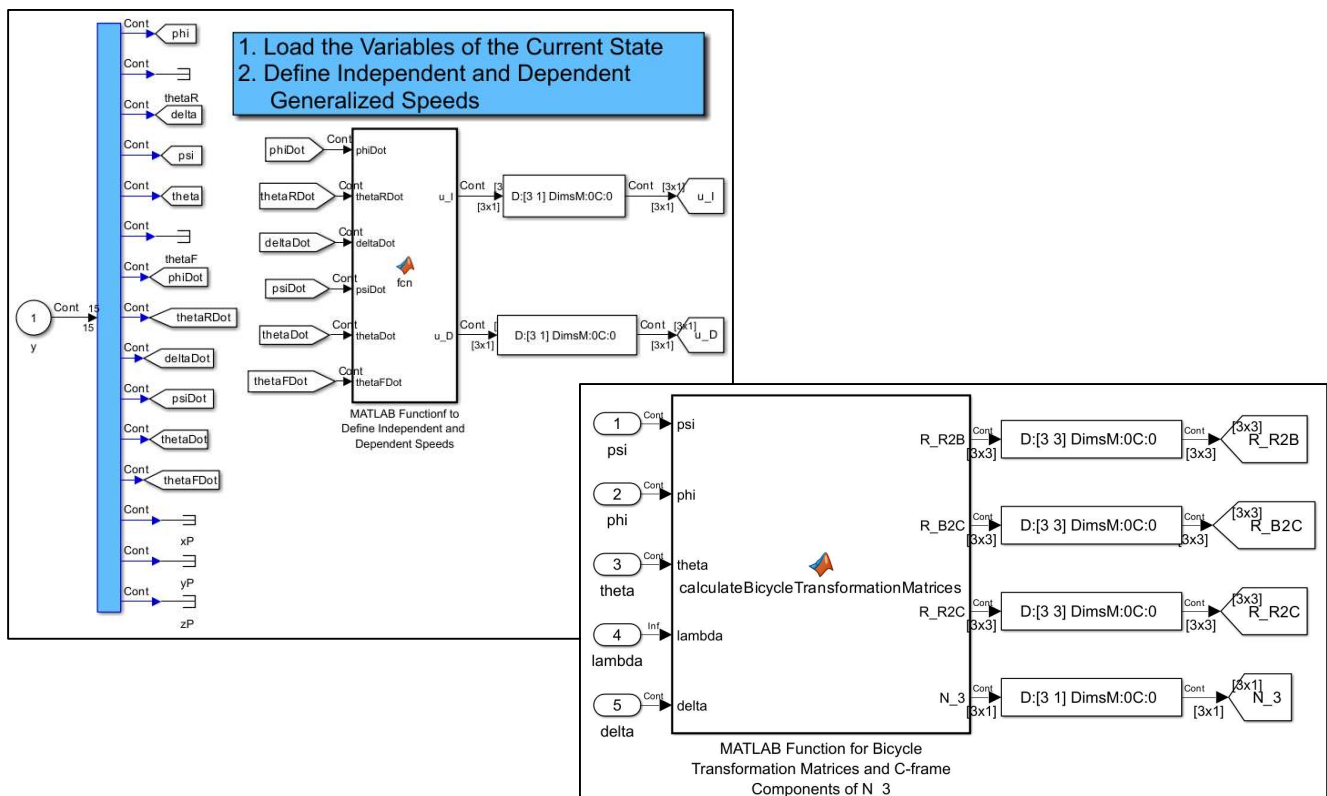
The contents of the *third through tenth subsystems* are shown below in Panels 3, 4, and 5. These subsystems calculate the *constraint matrices*, *partial angular velocity matrices*, *mass-center partial velocity matrices*, *angular velocity and mass center velocity vectors*, *derivatives of the constraint matrices*, *derivatives of the*

partial angular velocity matrices, and *derivatives of the mass-center partial velocity matrices*. The tenth subsystem *completes* the *calculation* of the 15×1 vector “dydt”, the derivative of the current state vector. Again, details of the MATLAB functions are not shown as they were previously discussed.

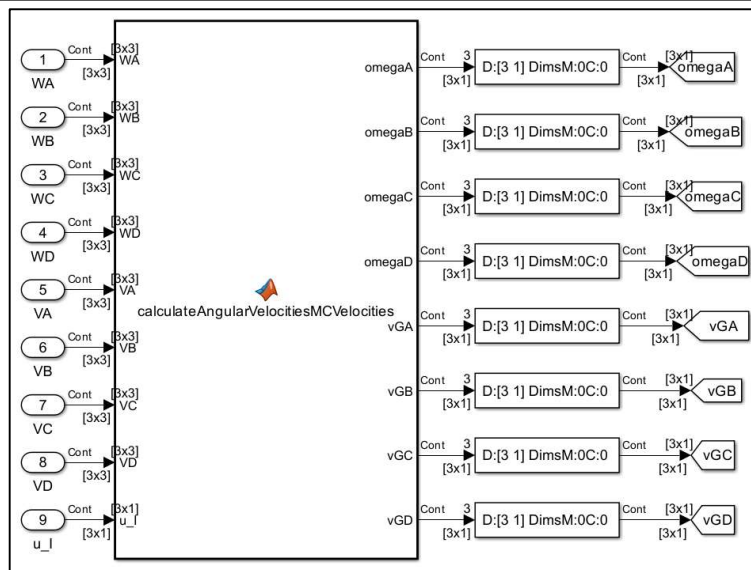
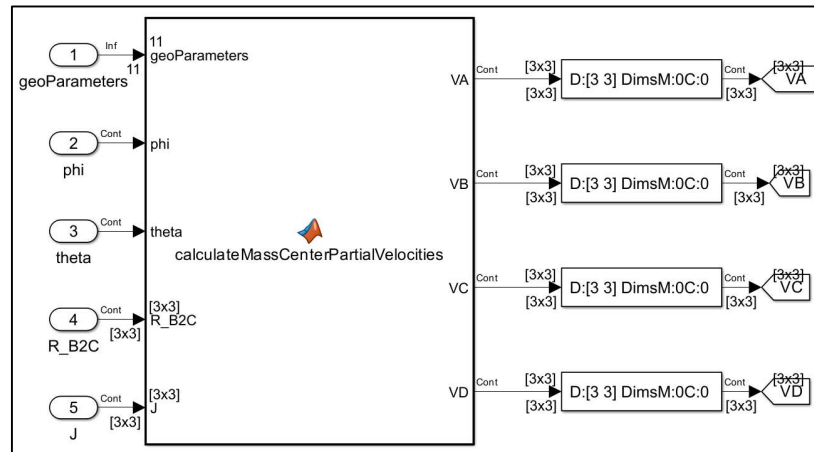
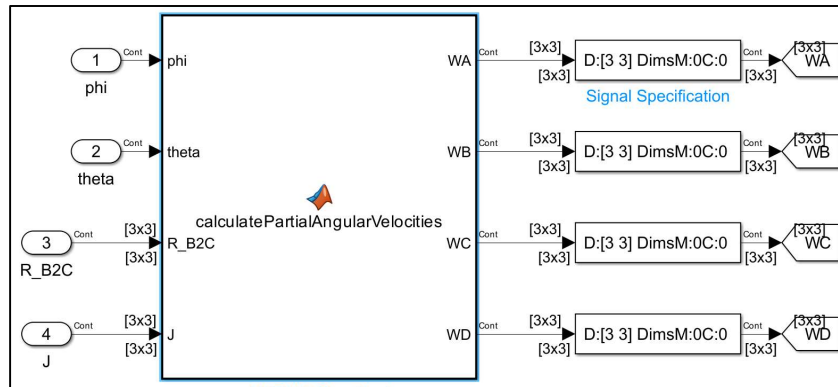
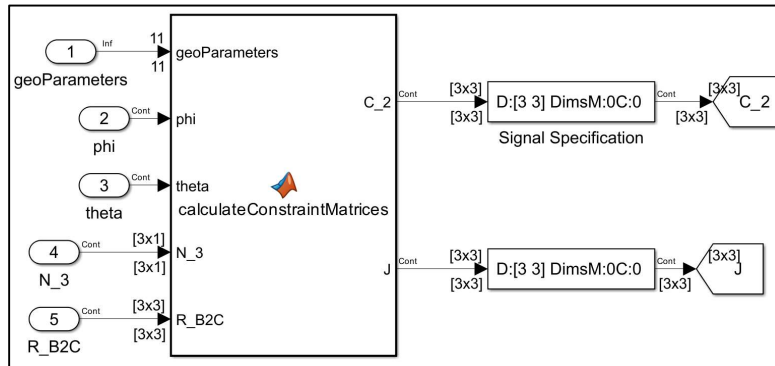
Simulink Model 3 Panel 1: (main subsystem with ten additional subsystems to calculate the state vector derivative)



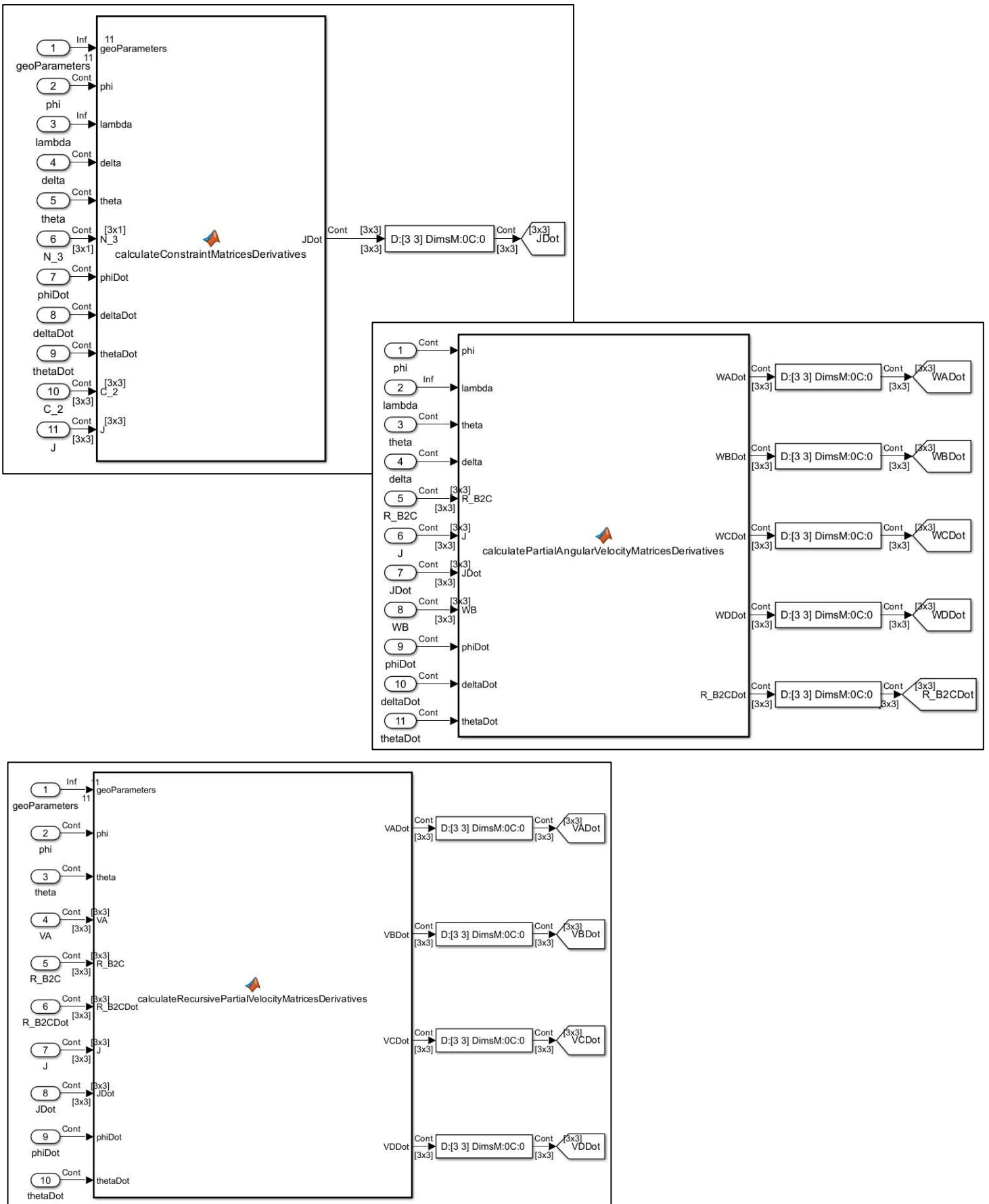
Simulink Model 3 Panel 2: (subsystems 1 and 2)



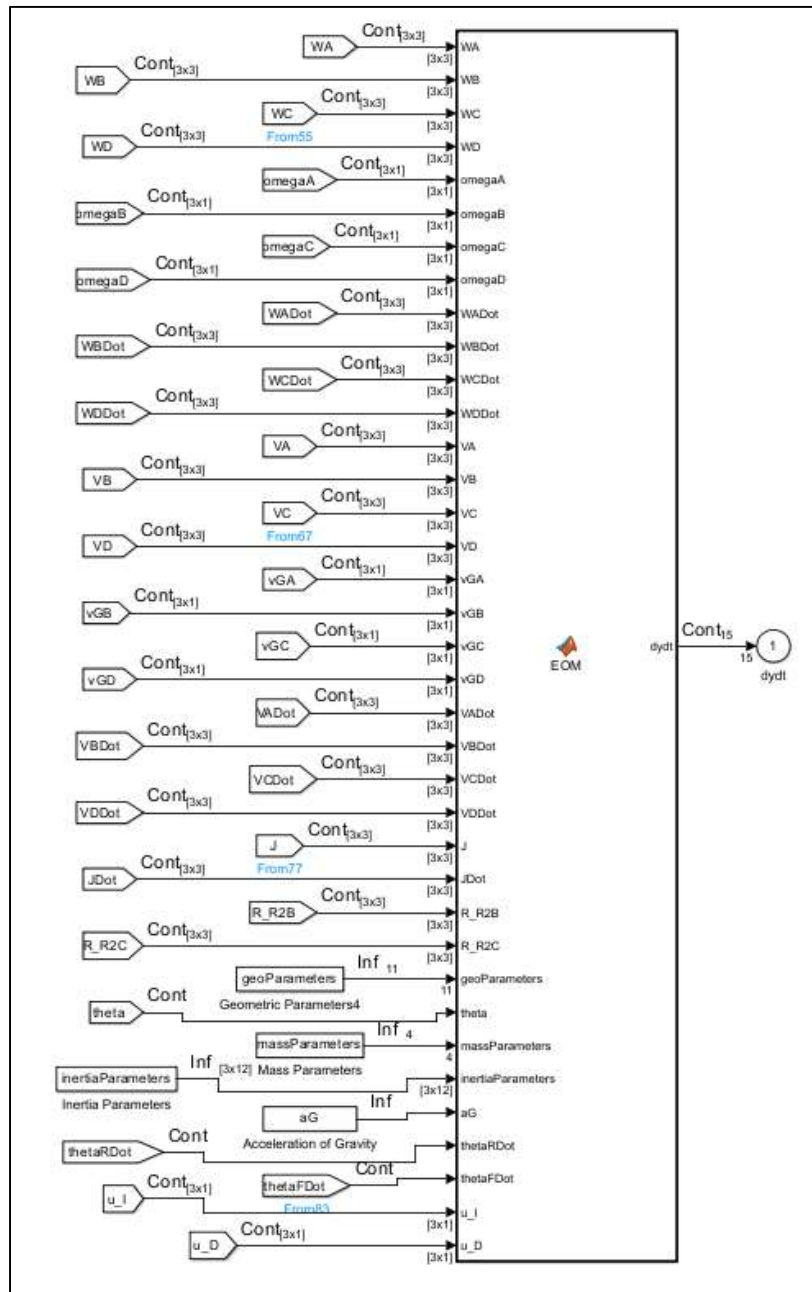
Simulink Model 3 Panel 3: (subsystems 3 through 6)



Simulink Model 3 Panel 4: (subsystems 7 through 9)



Simulink Model 3 Panel 5: (subsystem 10)

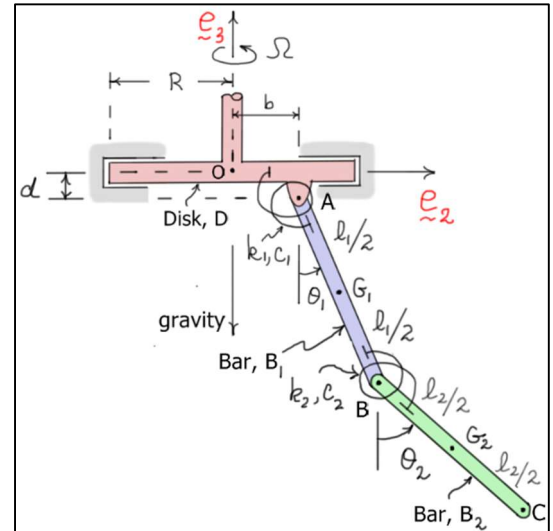


Model 3 Simulation Results:

The simulation results for this model are *identical* to those presented above for Model 1.

Exercises:

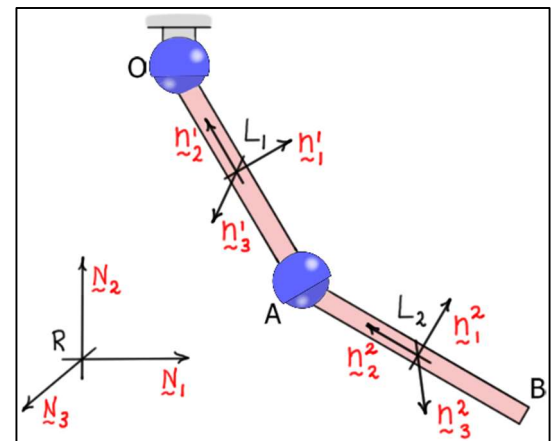
7.1 A three-body mechanical system is shown in the diagram. Disk D is connected to the ground with a revolute joint allowing rotation only in the fixed \underline{e}_3 direction. Bar B_1 is connected to the disk and bar B_2 is connected to bar B_1 with revolute joints allowing relative rotations only in the $\underline{e}_1 = \underline{e}_2 \times \underline{e}_3$ direction. The disk rotates at a **constant** rate of $\Omega = 2\pi$ (rad/s), while the motions of the bars are free. **Torsional springs** and **dampers** restrain the motion of the bar. Develop a SimMechanics model of the system using the following data.



- Disk mass: $m = 0.2$ (slug)
- Disk physical dimensions: $R = 1$ (ft), $b = 0.5$ (ft), $d = 0.2$ (ft)
- Disk inertia: disk is assumed to be a thin circular disk
- Bars' masses: $m = 0.1$ (slug)
- Bars' physical dimensions: lengths $\ell = 2$ (ft) and radii $r = 0.1$ (ft)
- Bars' inertias: bars are assumed to have cylindrical shape with the given length and radius
- Bar B_1 spring and dampers: $k_1 = 50$ (ft-lb/rad) and $c_1 = 4$ (ft-lb-s/rad)
- Bar B_2 spring and dampers: $k_2 = 5$ (ft-lb/rad) and $c_2 = 0.4$ (ft-lb-s/rad)

The bars initially both hang vertically downward, that is $\theta_1 = \theta_2 = 0$. Using your model identify the **final values** of the angles of the two bars. Without a set of explicit equations of motion, how can you **validate** the model (at least qualitatively)?

7.2 The system shown is a **three-dimensional double pendulum** or **arm**. The first link is connected to ground and the second link is connected to the first with **ball and socket** joints at O and A. The **orientation** of each link is defined relative to the ground using a 3-1-3 **body-fixed** rotation sequence. The links are **identical** with mass m and length ℓ . The links are assumed to be **slender bars** with **circular cross sections** (diameter, d) and mass centers at their **midpoints**. Using the physical data listed below and the equations of motion developed in Unit 5 of this volume, develop a MATLAB script to simulate the motion of the system under the action of gravity. Use the MATLAB function ODE45.



Physical data:

$$\ell = 0.5 \text{ (m)}, m = 2.8 \text{ (kg)}, d = 3 \text{ (cm)}$$

- 7.3** Model the system of Exercise 7.2 using a MATLAB script and a Simulink model. The MATLAB script should initialize all necessary variables to integrate the equations of motion using Simulink. Compare the results with the model of Exercise 7.2.
- 7.4** Model the system of Exercise 7.2 using a MATLAB script and a SimMechanics model. The MATLAB script should initialize all necessary variables to support the SimMechanics model. Compare the results with the previous two models.

References:

1. L. Meirovitch, *Methods of Analytical Dynamics*, McGraw-Hill, 1970.
2. T.R. Kane, P.W. Likins, and D.A. Levinson, *Spacecraft Dynamics*, McGraw-Hill, 1983
3. T.R. Kane and D.A. Levinson, *Dynamics: Theory and Application*, McGraw-Hill, 1985
4. R.L. Huston, *Multibody Dynamics*, Butterworth-Heinemann, 1990
5. H. Baruh, *Analytical Dynamics*, McGraw-Hill, 1999
6. H. Josephs and R.L. Huston, *Dynamics of Mechanical Systems*, CRC Press, 2002
7. R.C. Hibbeler, *Engineering Mechanics: Dynamics*, 13th Ed., Pearson Prentice Hall, 2013
8. J.L. Meriam and L.G. Craig, *Engineering Mechanics: Dynamics*, 3rd Ed, 1992
9. F.P. Beer and E.R. Johnston, Jr. *Vector Mechanics for Engineers: Dynamics*, 4th Ed, 1984
10. M.L. Boas, *Mathematical Methods in the Physical Sciences*, John Wiley & Sons, Inc., 1966
11. R. Bronson, *Matrix Methods – An Introduction*, Academic Press, 1970
12. Bicycle references:
 - a. B.W. Kostich, “Development of Empirical and Virtual Tools for the Study of Bicycle Safety”, MS Thesis, Western Michigan University, 2017
 - b. J.K. Moore, “Human Control of a Bicycle”, Ph.D. Dissertation, University of California-Davis, 2012
 - c. J.P. Meijaard, J.M. Papadopoulos, A. Ruina, and A.L. Schwab, “Linearized Dynamics Equations for the Balance and Steer of a Bicycle: a Benchmark and Review”, *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 463(2084), 1955–1982, 2007
 - d. P. Basu-Mandal, A. Chatterjee, and J.M. Papadopoulos, “Hands-free Circular Motions of a Benchmark Bicycle. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 463(March), 1983–2003, 2007