An Introduction to Three-Dimensional, Rigid Body Dynamics

James W. Kamman, PhD

Volume I: Kinematics

Unit 10

Introduction to Modeling Mechanical System Kinematics using MATLAB® Scripts, Simulink® and SimMechanics®

Summary

This unit provides and introduction to ***modeling*** mechanical system kinematics using ***MATLAB scripts***, Simulink ***models***, and ***SimMechanics models***. 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. SimMechanics 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 SimMechanics models.

Developing models using MATLAB scripts alone requires an analyst to ***write*** or ***computationally develop*** all the necessary equations to model the system. Many solution algorithms built into MATLAB and its toolboxes can be used to ***solve*** the equations. This is a ***detailed process*** and may be too time-intensive for the average analyst to master. Alternatively, Simulink can be used to ***speed-up model development*** by allowing the solution process to be programmed in the form of block diagrams. Like MATLAB, Simulink has many built-in algorithms to lighten the analyst's load. Because these two approaches give the analyst ***total control of all modeling and solution details***, they provide the ***most flexible*** modeling environment. However, the amount of time and effort required to develop models using these approaches ***increases*** rapidly as the system complexity increases.

SimMechanics models are Simulink models that contain ***special blocks*** for modeling ***multibody dynamics***. These blocks eliminate the need for analysts to develop a set of equations of motion of their own. The analyst builds a block diagram model of the system, and SimMechanics ***develops and solves*** the equations of motion. These models require ***less time and knowledge*** to develop, but they are also less flexible than those discussed above in that the analyst ***does not have access*** to the detailed equations or the solution process.

Page Count	Examples	Suggested Exercises
51	5	7

MATLAB Scripts

Example 1: Conversions from Orientation Angles to Euler Parameters

This example provides a MATLAB script for computing a set of ***four Euler parameters*** associated with a 1-2-3 body-fixed sequence of ***orientation angles***. The script consists of four separate routines, a ***main module*** and three supporting ***functions***. See Units 5 and 6 for a description of body-fixed orientation angle sequences and Euler parameters.

The main module first ***sets*** the values of the three orientation angles and ***converts*** the angles to radians. To ***find*** the four Euler parameters associated with these angles, it ***calls a function*** to compute the ***transformation matrix*** associated with the angles, and using that transformation matrix, it then ***calls a second function*** to calculate the four Euler parameters associated with that transformation matrix.

Main module:

```
%   This script calculates the values of the 4 Euler parameters associated
%   with a 1-2-3 body-fixed sequence of orientation angles

    degtoRad = pi/180.0;

%   Set the three angles of the 1-2-3 orientation angle sequence
    theta1Deg = 180; theta1 = theta1Deg*degtoRad;
    theta2Deg = 30;  theta2 = theta2Deg*degtoRad;
    theta3Deg = 20;  theta3 = theta3Deg*degtoRad;

%   get the coordinate transformation matrix associated with this angle sequence
    transformationMatrix = ...
        calculateTransformationMatrixFromAngles123(theta1,theta2,theta3);

%   get the Euler parameters associated with the transformation matrix
    eulerParameter = ...
      calculateEulerParametersFromTransformationMatrix(transformationMatrix);

%   get the transformation matrix associated with the Euler parameters
    transformationMatrixE = ...
      calculateTransformationMatrixFromEulerParameters(eulerParameter);

%   Check calculate [Ra][Re’] to see how close to identity matrix
    identityCheck = transformationMatrix*(transformationMatrixE');

%   display the results
    disp('The Angles:')
    disp(theta1Deg); disp(theta2Deg); disp(theta3Deg);
    disp('')
    disp('The Euler Parameters:')
    disp(eulerParameter)
    disp('The Transformation Matrix (based on the angles):')
    disp(transformationMatrix)
    disp('The Transformation Matrix (based on the Euler parameters):')
    disp(transformationMatrixE)
    disp('Identity matrix?:')
    disp(identityCheck)
```

The values of the Euler parameters are ***checked*** in a two-step process. First, a function is called to compute the transformation matrix associated with the newly-computed parameters. Then, it computes the matrix product of the original transformation matrix with the transpose of the transformation matrix associated with the Euler parameters. If the four Euler parameters are accurate, the result should be the identity matrix. Recall that these transformation matrices are ***orthogonal matrices*** so that their ***inverses*** are equal to their ***transposes***. The results are then ***sent*** to the MATLAB command window. The angles, Euler parameters, transformation matrices and identity matrix check are all displayed. A set of ***sample output*** is shown below.

Command Window Output:

```
>> angles123toEulerParameters
Largest Euler parameter is epsilon_1
The Angles:

   180
    30
    20

The Euler Parameters:

   0.951251242564198
  -0.167731259496521
   0.254887002244179
  -0.044943455527548

The Transformation Matrix (based on the angles):

   0.813797681349374  -0.342020143325669   0.469846310392954
  -0.296198132726024  -0.939692620785908  -0.171010071662834
   0.500000000000000  -0.000000000000000  -0.866025403784439

The Transformation Matrix (based on the Euler parameters):

   0.813797681349374  -0.342020143325669   0.469846310392954
  -0.296198132726024  -0.939692620785909  -0.171010071662834
   0.500000000000000  -0.000000000000000  -0.866025403784439

Identity matrix?:

   1.000000000000000  -0.000000000000000   0.000000000000000
   0.000000000000000   1.000000000000000                   0
  -0.000000000000000   0.000000000000000   1.000000000000000
```

Supporting Functions:

Brief descriptions of the operation of each of the three supporting functions used by the main module are given below. A detailed listing of each function follows the descriptions. To aid in understanding the calculations in each function, the following equation provides the form of the coordinate transformation matrices in terms of the 1-2-3 orientation angle sequence and in terms of the four Euler parameters as presented in Units 5 and 6.

$$[R]=\begin{bmatrix} C_2C_3 & C_1S_3+S_1S_2C_3 & S_1S_3-C_1S_2C_3 \\ -C_2S_3 & C_1C_3-S_1S_2S_3 & S_1C_3+C_1S_2S_3 \\ S_2 & -S_1C_2 & C_1C_2 \end{bmatrix}=\begin{bmatrix} (\varepsilon_1^2-\varepsilon_2^2-\varepsilon_3^2+\varepsilon_4^2) & 2(\varepsilon_1\varepsilon_2+\varepsilon_3\varepsilon_4) & 2(\varepsilon_1\varepsilon_3-\varepsilon_2\varepsilon_4) \\ 2(\varepsilon_1\varepsilon_2-\varepsilon_3\varepsilon_4) & (-\varepsilon_1^2+\varepsilon_2^2-\varepsilon_3^2+\varepsilon_4^2) & 2(\varepsilon_2\varepsilon_3+\varepsilon_1\varepsilon_4) \\ 2(\varepsilon_1\varepsilon_3+\varepsilon_2\varepsilon_4) & 2(\varepsilon_2\varepsilon_3-\varepsilon_1\varepsilon_4) & (-\varepsilon_1^2-\varepsilon_2^2+\varepsilon_3^2+\varepsilon_4^2) \end{bmatrix}$$

1. `calculateTransformationMatrixFromAngles123` – accepts the values of three orientation angles and computes the elements of the 3×3 transformation matrix $[R]$ discussed in Unit 5.
2. `calculateEulerParametersFromTransformationMatrix` – accepts the elements of a 3×3 transformation matrix and computes the four Euler parameters. The function uses an algorithm recommended by Baruh in reference 1 and presented in Unit 6. First, the squares of the four Euler parameters are calculated. The parameter with the largest value is computed from these results and assumed to be positive. The other three parameters are computed with equations that enable their values and algebraic signs to be determined. See Unit 6 for more details.
3. `calculateTransformationMatrixFromEulerParameters` – accepts the values of four Euler parameters and computes the elements of the corresponding 3×3 transformation matrix $[R]$.

Function: `calculateTransformationMatrixFromAngles123`

```
function [transformationMatrix] = ...
            calculateTransformationMatrixFromAngles123(theta1,theta2,theta3)
%
%   This function computes the coordinate transformation matrix associated
%   with a 1-2-3 orientation angle sequence.
%
%   Input:
%   theta1, theta2, and theta3 are the three angles in radians (in order)
%
%   Output:
%   transformationMatrix

%   Compute the transformation matrix
    transformationMatrix = zeros(3);

%   first row
    transformationMatrix(1,1) = cos(theta2)*cos(theta3);
    transformationMatrix(1,2) = (cos(theta1)*sin(theta3)) ...
                                  + (sin(theta1)*sin(theta2)*cos(theta3));
    transformationMatrix(1,3) = (sin(theta1)*sin(theta3)) ...
                                  - (cos(theta1)*sin(theta2)*cos(theta3));
%   second row
    transformationMatrix(2,1) = -cos(theta2)*sin(theta3);
    transformationMatrix(2,2) = (cos(theta1)*cos(theta3)) ...
                                  - (sin(theta1)*sin(theta2)*sin(theta3));
    transformationMatrix(2,3) = (sin(theta1)*cos(theta3)) ...
                                  + (cos(theta1)*sin(theta2)*sin(theta3));
%   third row
    transformationMatrix(3,1) = sin(theta2);
    transformationMatrix(3,2) = -sin(theta1)*cos(theta2);
    transformationMatrix(3,3) = cos(theta1)*cos(theta2);
```

Function: *calculateEulerParametersFromTransformationMatrix*

```
function [eulerParameter] = ...
         calculateEulerParametersFromTransformationMatrix(transformationMatrix)
%
%   This function calculates the Euler parameters associated with a
%   specific transformation matrix
%   Ref: H. Baruh, Analytical Dynamics, WCB/McGraw-Hill, 1999.
%
%   Input:
%   3x3 transformationMatrix
%
%   Output:
%   4 Euler paramters
%
%   Initialize the Euler parameter squares array, Euler parameter array,
%   and tolerance
    eulerParametersq = zeros(4,1); eulerParameter = zeros(4,1);

%   calculate the squares of the Euler parameters
    eulerParametersq(1) = (transformationMatrix(1,1)-transformationMatrix(2,2) ...
                          -transformationMatrix(3,3)+1.0)/4.0;
    eulerParametersq(2) = (-transformationMatrix(1,1)+transformationMatrix(2,2) ...
                          -transformationMatrix(3,3)+1.0)/4.0;
    eulerParametersq(3) = (-transformationMatrix(1,1)-transformationMatrix(2,2) ...
                          +transformationMatrix(3,3)+1.0)/4.0;
    eulerParametersq(4) = (transformationMatrix(1,1)+transformationMatrix(2,2) ...
                          +transformationMatrix(3,3)+1.0)/4.0;

%   Determine which is the largest Euler parameter
    eulerMax = eulerParameter(1); iPoint = 1;
    for i = 2:4
        if (eulerParameter(i) > eulerMax)
            eulerMax = eulerParameter(i);
            iPoint = i;
        end
    end
%   Find the other Euler parameters based on the largest parameter
    if (iPoint == 1)
        eulerParameter(1) = sqrt(abs(eulerParametersq(1)));
        eulerParameter(2) = (transformationMatrix(1,2) + ...
                             transformationMatrix(2,1))/(4.0*eulerParameter(1));
        eulerParameter(3) = (transformationMatrix(1,3) + ...
                             transformationMatrix(3,1))/(4.0*eulerParameter(1));
        eulerParameter(4) = (transformationMatrix(2,3) - ...
                             transformationMatrix(3,2))/(4.0*eulerParameter(1));
    end
    if (iPoint == 2)
        eulerParameter(2) = sqrt(abs(eulerParametersq(2)));
        eulerParameter(1) = (transformationMatrix(1,2) + ...
                             transformationMatrix(2,1))/(4.0*eulerParameter(2));
        eulerParameter(3) = (transformationMatrix(2,3) + ...
                             transformationMatrix(3,2))/(4.0*eulerParameter(2));
        eulerParameter(4) = (transformationMatrix(3,1) - ...
                             transformationMatrix(1,3))/(4.0*eulerParameter(2));
    end
                                    ⋮
```

Function: *calculateEulerParametersFromTransformationMatrix* (continued)

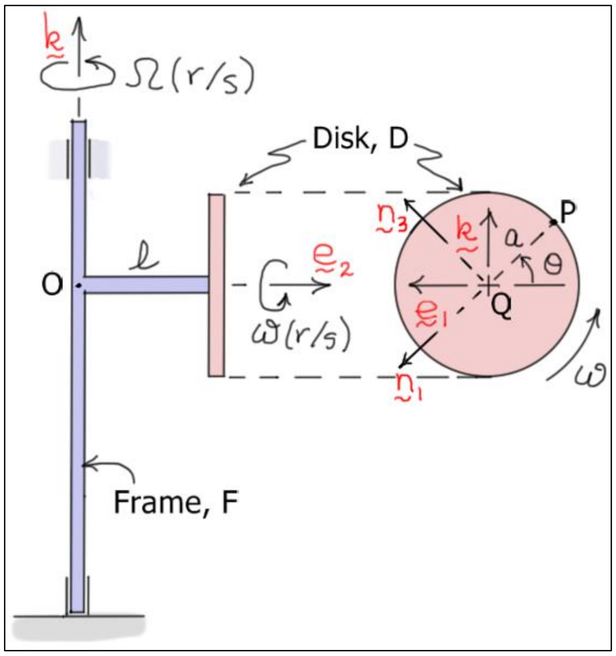
```
                                        ⋮
    if (iPoint == 3)
        eulerParameter(3) = sqrt(abs(eulerParametersq(3)));
        eulerParameter(1) = (transformationMatrix(1,3) + ...
                             transformationMatrix(3,1))/(4.0*eulerParameter(3));
        eulerParameter(2) = (transformationMatrix(2,3) + ...
                             transformationMatrix(3,2))/(4.0*eulerParameter(3));
        eulerParameter(4) = (transformationMatrix(1,2) - ...
                             transformationMatrix(2,1))/(4.0*eulerParameter(3));
    end
    if (iPoint == 4)
        eulerParameter(4) = sqrt(abs(eulerParametersq(4)));
        eulerParameter(1) = (transformationMatrix(2,3) - ...
                             transformationMatrix(3,2))/(4.0*eulerParameter(4));
        eulerParameter(2) = (transformationMatrix(3,1) - ...
                             transformationMatrix(1,3))/(4.0*eulerParameter(4));
        eulerParameter(3) = (transformationMatrix(1,2) - ...
                             transformationMatrix(2,1))/(4.0*eulerParameter(4));
    end
```

Function: *calculateTransformationMatrixFromEulerParameters*

```
function [transformationMatrix] = ...
    calculateTransformationMatrixFromEulerParameters(eulerParameter)
%
%   This function calculates the coordinate transformation matrix
%   associated with a set of 4 Euler parameters
%
%   Input:
%   4x1 Euler Parameter array (epsilon1,epsilon2,epsilon3,epsilon4)
%
%   Output:
%   3x3 coordinate transformation matrix

%   Initialize the transformation matrix
    transformationMatrix=zeros(3);

%   calculate the transformation matrix associated with the Euler parameters

%   first row
    transformationMatrix(1,1) = (eulerParameter(1)^2) - (eulerParameter(2)^2) ...
                              - (eulerParameter(3)^2) + (eulerParameter(4)^2);
    transformationMatrix(1,2) = 2.0*((eulerParameter(1)*eulerParameter(2)) + ...
                                     (eulerParameter(3)*eulerParameter(4)));
    transformationMatrix(1,3) = 2.0*((eulerParameter(1)*eulerParameter(3)) - ...
                                     (eulerParameter(2)*eulerParameter(4)));
%   second row
    transformationMatrix(2,1) = 2.0*((eulerParameter(1)*eulerParameter(2)) - ...
                                     (eulerParameter(3)*eulerParameter(4)));
    transformationMatrix(2,2) = -(eulerParameter(1)^2) + (eulerParameter(2)^2) ...
                                - (eulerParameter(3)^2) + (eulerParameter(4)^2);
    transformationMatrix(2,3) = 2.0*((eulerParameter(2)*eulerParameter(3)) + ...
                                     (eulerParameter(1)*eulerParameter(4)));
                                        ⋮
```

```
                                    ⋮
%   third row
    transformationMatrix(3,1) = 2.0*((eulerParameter(1)*eulerParameter(3)) + ...
                                     (eulerParameter(2)*eulerParameter(4)));
    transformationMatrix(3,2) = 2.0*((eulerParameter(2)*eulerParameter(3)) - ...
                                     (eulerParameter(1)*eulerParameter(4)));
    transformationMatrix(3,3) = -(eulerParameter(1)^2) - (eulerParameter(2)^2) ...
                                + (eulerParameter(3)^2) + (eulerParameter(4)^2);
```

Example 2:



The system shown consists of two connected bodies – the frame *F* and the disk *D*. Frame *F* rotates at a rate of Ω (rad/s) about the fixed vertical direction annotated by the unit vector $\underset{\sim}{k}$. Disk *D* is affixed to and rotates relative to *F* at a rate of ω (rad/s) about the horizontal arm of *F* which is annotated by the rotating unit vector $\underset{\sim}{e}_2$.

Equations were developed in each of Units 2, 3, and 4 for the velocity and acceleration of point *P* on the periphery of *D* as shown. The following kinematic results were generated and expressed in frame *F*

$$\boxed{{}^R\underset{\sim}{\omega}_D = \omega\,\underset{\sim}{e}_2 + \Omega\,\underset{\sim}{k}} \quad \boxed{{}^R\underset{\sim}{\alpha}_D = -\omega\,\Omega\,\underset{\sim}{e}_1 + \dot{\omega}\,\underset{\sim}{e}_2 + \dot{\Omega}\,\underset{\sim}{k}}$$

$$\boxed{{}^R\underset{\sim}{v}_P = \left(a\,\omega\,S_\theta - \ell\,\Omega\right)\underset{\sim}{e}_1 - \left(a\,\Omega\,C_\theta\right)\underset{\sim}{e}_2 + \left(a\,\omega\,C_\theta\right)\underset{\sim}{k}}$$

$$\boxed{{}^R\underset{\sim}{a}_P = \left[a\,\dot{\omega}\,S_\theta - \ell\,\dot{\Omega} + a\,C_\theta\left(\omega^2+\Omega^2\right)\right]\underset{\sim}{e}_1 + \left[-a\,\dot{\Omega}\,C_\theta + 2\,a\,\omega\,\Omega\,S_\theta - \ell\,\Omega^2\right]\underset{\sim}{e}_2 + \left[a\,\dot{\omega}\,C_\theta - a\,\omega^2 S_\theta\right]\underset{\sim}{k}}$$

These results can also easily be expressed in the frame $D:(\underset{\sim}{n}_1,\underset{\sim}{e}_2,\underset{\sim}{n}_3)$ by noting that $\underset{\sim}{k} = -S_\theta\,\underset{\sim}{n}_1 + C_\theta\,\underset{\sim}{n}_3$ and $\underset{\sim}{e}_1 = C_\theta\,\underset{\sim}{n}_1 + S_\theta\,\underset{\sim}{n}_3$. Making these substitutions and collecting terms gives the representations of these vectors in the frame *D*.

$$\boxed{{}^R\underset{\sim}{\omega}_D = -\Omega\,S_\theta\,\underset{\sim}{n}_1 + \omega\,\underset{\sim}{e}_2 + \Omega\,C_\theta\,\underset{\sim}{n}_3} \quad \boxed{{}^R\underset{\sim}{\alpha}_D = -\left(\omega\,\Omega\,C_\theta + \dot{\Omega}\,S_\theta\right)\underset{\sim}{n}_1 + \dot{\omega}\,\underset{\sim}{e}_2 + \left(\dot{\Omega}\,C_\theta - \omega\,\Omega\,S_\theta\right)\underset{\sim}{n}_3}$$

$$\boxed{{}^R\underset{\sim}{v}_P = \left(-\ell\,\Omega\,C_\theta\right)\underset{\sim}{n}_1 - \left(a\,\Omega\,C_\theta\right)\underset{\sim}{e}_2 + \left(a\,\omega - \ell\,\Omega\,S_\theta\right)\underset{\sim}{n}_3}$$

$$\boxed{\begin{aligned}{}^R\underset{\sim}{a}_P = &\left[-\ell\,\dot{\Omega}C_\theta + a\,C_\theta^2\left(\omega^2+\Omega^2\right) + a\,\omega^2 S_\theta^2\right]\underset{\sim}{n}_1 + \left[-a\,\dot{\Omega}\,C_\theta + 2\,a\,\omega\,\Omega\,S_\theta - \ell\,\Omega^2\right]\underset{\sim}{e}_2 + \\ &\left[a\,\dot{\omega} - \ell\,\dot{\Omega}S_\theta + a\,C_\theta S_\theta \Omega^2\right]\underset{\sim}{n}_3\end{aligned}}$$

These equations can be used in a MATLAB script to find components of the angular velocity and angular acceleration of disk *D* and the velocity and acceleration of point *P* at a series of times throughout the motion of the system. To illustrate this process, the following data is used

Variable	Value	Variable	Value
ℓ	0.5 (m)	$\dot{\omega}=\ddot{\theta}$	2 (rad/s^2) …constant
a	0.25 (m)	$\dot{\Omega}=\ddot{\phi}$	3 (rad/s^2) …constant

Note that the relative angular accelerations ($\dot{\omega}$ and $\dot{\Omega}$) are ***constant***, and the ***initial values*** of the corresponding relative angular velocities (ω and Ω) and relative angles (θ and ϕ) are taken to be ***zero***.

Using these values, the values of ω, Ω, θ, and ϕ can be calculated at any time as follows.

$$\omega(t)=2t \text{ (rad/s)} \qquad \Omega(t)=3t \text{ (rad/s)} \qquad \theta(t)=t^2 \text{ (rad)} \qquad \phi(t)=\tfrac{3}{2}t^2 \text{ (rad)}$$

Specifically, at $t=2 \text{ (sec)}$:

$$\boxed{\begin{aligned} {}^R\underset{\sim}{\omega}_D &= -\Omega S_\theta \underset{\sim}{n}_1+\omega\underset{\sim}{e}_2+\Omega C_\theta\underset{\sim}{n}_3 = -(3t)\sin(t^2)\underset{\sim}{n}_1+(2t)\underset{\sim}{e}_2+(3t)\cos(t^2)\underset{\sim}{n}_3\\ &=4.541\underset{\sim}{n}_1+4\underset{\sim}{e}_2-3.922\underset{\sim}{n}_3 \text{ (rad/s)}\end{aligned}}$$

$$\boxed{\begin{aligned} {}^R\underset{\sim}{\alpha}_D &= -\left(\omega\Omega C_\theta+\dot{\Omega}S_\theta\right)\underset{\sim}{n}_1+\dot{\omega}\underset{\sim}{e}_2+\left(\dot{\Omega}C_\theta-\omega\Omega S_\theta\right)\underset{\sim}{n}_3\\ &=-\left(6t^2\cos(t^2)+3\sin(t^2)\right)\underset{\sim}{n}_1+2\underset{\sim}{e}_2+\left(3\cos(t^2)-6t^2\sin(t^2)\right)\underset{\sim}{n}_3\\ &=17.96\underset{\sim}{n}_1+2\underset{\sim}{e}_2+16.20\underset{\sim}{n}_3 \text{ (rad/s}^2\text{)}\end{aligned}}$$

$$\boxed{\begin{aligned} {}^R\underset{\sim}{v}_P &= \left(-\ell\Omega C_\theta\right)\underset{\sim}{n}_1-\left(a\Omega C_\theta\right)\underset{\sim}{e}_2+\left(a\omega-\ell\Omega S_\theta\right)\underset{\sim}{n}_3\\ &=\left(-\tfrac{3}{2}t\cos(t^2)\right)\underset{\sim}{n}_1-\left(\tfrac{3}{4}t\cos(t^2)\right)\underset{\sim}{e}_2+\left(\tfrac{1}{2}t-\tfrac{3}{2}t\sin(t^2)\right)\underset{\sim}{n}_3\\ &=1.961\underset{\sim}{n}_1+0.9805\underset{\sim}{e}_2+3.270\underset{\sim}{n}_3 \text{ (m/s)}\end{aligned}}$$

$$\boxed{\begin{aligned} {}^R\underset{\sim}{a}_P &= \left[-\ell\dot{\Omega}C_\theta+aC_\theta^2\left(\omega^2+\Omega^2\right)+a\omega^2S_\theta^2\right]\underset{\sim}{n}_1+\left[-a\dot{\Omega}C_\theta+2a\omega\Omega S_\theta-\ell\Omega^2\right]\underset{\sim}{e}_2+\\ &\qquad\left[a\dot{\omega}-\ell\dot{\Omega}S_\theta+aC_\theta S_\theta\Omega^2\right]\underset{\sim}{n}_3\\ &=\left[-\tfrac{3}{2}\cos(t^2)+\tfrac{1}{4}\cos^2(t^2)\left(4t^2+9t^2\right)+t^2\sin^2(t^2)\right]\underset{\sim}{n}_1+\left[-\tfrac{3}{4}\cos(t^2)+3t^2\sin(t^2)-\tfrac{9}{2}t^2\right]\underset{\sim}{e}_2\\ &\quad+\left[\tfrac{1}{2}-\tfrac{3}{2}\sin(t^2)+\tfrac{9}{4}t^2\sin(t^2)\cos(t^2)\right]\underset{\sim}{n}_3\\ &=8.826\underset{\sim}{n}_1-26.59\underset{\sim}{e}_2+6.087\underset{\sim}{n}_3 \text{ (m/s}^2\text{)}\end{aligned}}$$

These results can be generated over an interval of time using a MATLAB script. An example script is shown below in a series of ***three panels***. The ***first panel*** contains the ***first two sections*** of the script. The ***first section*** describes what the script calculates and plots. The ***second section*** initializes the variables and arrays necessary for the calculations that follow. The physical data and input motions are as described above.

<u>Script – Panel 1</u>: (functional description and initialization of variables and arrays)

```
%% Unit 10 Example 2: Relative velocity and acceleration

%  This script calculates the velocity and acceleration of point P fixed
%  on the edge of disk D relative to ground (R). Results are expressed
%  in frame D.
%
%  The following results are plotted over the time vector:
%   Figure 1: Angular Velocity of D in R - Components in Frame D
%   Figure 2: Angular Acceleration of D in R - Components in Frame D
%   Figure 3: Velocity Components of Point P in R expressed in Frame D
%   Figure 4: Acceleration Components of Point P in R expressed in Frame D

%% Initialize variables
   armLength   = 0.5;       % length of arm F in meters
   diskRadius  = 0.25;      % radius of disk D in meters

   omegaDot    = 2.0;       % angular acceleration of D in F in rad/s^2
   capOmegaDot = 3.0;       % angular acceleration of F in R in rad/s^2

   omegaInitial    = 0.0;   % initial angular velocity of D in F in rad/s
   capOmegaInitial = 0.0;   % initial angular velocity of F in R in rad/s

   thetaInitial    = 0.0;   % initial angle of D in radians
   phiInitial      = 0.0;   % initial angle of F in radians

   time = 0:0.01:4.0;  % time vector
   numberoftimes = length(time); % length of time vector

%  array initializations
   capOmega  = zeros(numberoftimes,1); omega     = zeros(numberoftimes,1);
   phi       = zeros(numberoftimes,1); theta     = zeros(numberoftimes,1);
   omegaDinR = zeros(numberoftimes,3); alphaDinR = zeros(numberoftimes,3);
   vn        = zeros(numberoftimes,3); an        = zeros(numberoftimes,3);
                                            ⋮
```

The ***second panel*** contains the ***third section*** of the script that ***calculates*** the ***disk-fixed components*** of the ***angular velocity*** and ***angular acceleration*** of the disk *D* and the ***velocity*** and ***acceleration*** of point *P*. The results are calculated on a time window from $0 \rightarrow 4\ (\text{sec})$ in steps of $0.01\ (\text{sec})$ using a "for-loop". The angular velocity and angular acceleration are expressed directly in the disk frame (*D*), whereas, the velocity and acceleration of *P* are first expressed in the frame *F* and then transformed into the frame *D*. (Note that these latter calculations could have been done directly in frame *D*.) The results are stored in arrays whose contents are plotted in the fourth (and final) section.

The ***third panel*** contains the ***fourth section*** of the script. It contains the statements necessary to create four figures, each having three subplots. The values of the $\underset{\sim}{n}_1$, $\underset{\sim}{n}_2$, and $\underset{\sim}{n}_3$ components of each vector are presented in the three subplots of that figure. Each figure has a title and each subplot has a grid and a label on each axis.

Script – Panel 2: (calculation of results using a "for-loop")

```
                                    ⋮
%% Calculate the velocity and acceleration of point P (see diagram in Unit 10)

for i = 1:numberoftimes

%  Assuming constant relative angular accelerations
   capOmega(i)  = capOmegaInitial + (capOmegaDot*time(i));
   phi(i)       = phiInitial + (capOmegaInitial*time(i)) + ...
                    (0.5*capOmegaDot*(time(i)^2));
   omega(i)  = omegaInitial + (omegaDot*time(i));
   theta(i)  = thetaInitial + (omegaInitial*time(i)) + ...
                (0.5*omegaDot*(time(i)^2));

   cosTheta = cos(theta(i)); sinTheta = sin(theta(i));

%  angular velocity of D in R - components in frame D
   omegaDinR(i,1)  = -capOmega(i)*sinTheta;
   omegaDinR(i,2)  = omega(i);
   omegaDinR(i,3)  =  capOmega(i)*cosTheta;

%  angular acceleration of D in R - components in frame D
   alphaDinR(i,1)  = (-omega(i)*capOmega(i)*cosTheta) - (capOmegaDot*sinTheta);
   alphaDinR(i,2)  =  omegaDot;
   alphaDinR(i,3)  = (-omega(i)*capOmega(i)*sinTheta) + (capOmegaDot*cosTheta);

%  velocity components in frame F
   ve1 = (diskRadius*omega(i)*sinTheta) - (armLength*capOmega(i));
   ve2 = -diskRadius*capOmega(i)*cosTheta;
   ve3 =  diskRadius*omega(i)*cosTheta;

%  velocity components in disk frame D
   vn(i,1)  = (ve1*cosTheta) - (ve3*sinTheta);
   vn(i,2)  =  ve2;
   vn(i,3)  = (ve1*sinTheta) + (ve3*cosTheta);

%  acceleration components in frame F
   ae1 = (diskRadius*omegaDot*sinTheta) - (armLength*capOmegaDot) + ...
                (diskRadius*cosTheta*((omega(i)^2) + (capOmega(i)^2)));
   ae2 = (-diskRadius*capOmegaDot*cosTheta) - (armLength*(capOmega(i)^2)) + ...
                (2.0*diskRadius*omega(i)*capOmega(i)*sinTheta);
   ae3 = diskRadius*((omegaDot*cosTheta) - ((omega(i)^2)*sinTheta));

%  acceleration components in disk frame D
   an(i,1)  = (ae1*cosTheta) - (ae3*sinTheta);
   an(i,2)  =  ae2;
   an(i,3)  = (ae1*sinTheta) + (ae3*cosTheta);

end
                                    ⋮
```

Script – Panel 3: (Plotting of results in four figure windows)

```
                                        ⋮
%% Plot the results

figure(1); clf;
subplot(3,1,1); plot(time,omegaDinR(:,1),'b-'); grid on;
xlabel('time(sec)'), ylabel('omegaDinR_1 (rad/s)');
title('Angular Velocity of D in R - Components in Frame D');

subplot(3,1,2); plot(time,omegaDinR(:,2),'b-'); grid on;
xlabel('time(sec)'), ylabel('omegaDinR_2 (rad/s)');

subplot(3,1,3); plot(time,omegaDinR(:,3),'b-'); grid on;
xlabel('time(sec)'), ylabel('omegaDinR_3 (rad/s)');

figure(2); clf;
subplot(3,1,1); plot(time,alphaDinR(:,1),'b-'); grid on;
xlabel('time(sec)'), ylabel('alphaDinR_1 (rad/s^2)');
title('Angular Acceleration of D in R - Components in Frame D');

subplot(3,1,2); plot(time,alphaDinR(:,2),'b-'); grid on;
xlabel('time(sec)'), ylabel('alphaDinR_2 (rad/s^2)');

subplot(3,1,3); plot(time,alphaDinR(:,3),'b-'); grid on;
xlabel('time(sec)'), ylabel('alphaDinR_3 (rad/s^2)');

figure(3); clf;
subplot(3,1,1); plot(time,vn(:,1),'b-'); grid on;
xlabel('time(sec)'), ylabel('v_1 (m/s)');
title('Velocity Components of Point P in R expressed in Frame D');

subplot(3,1,2); plot(time,vn(:,2),'b-'); grid on;
xlabel('time(sec)'), ylabel('v_2 (m/s)');

subplot(3,1,3); plot(time,vn(:,3),'b-'); grid on;
xlabel('time(sec)'), ylabel('v_3 (m/s)');

figure(4); clf;
subplot(3,1,1); plot(time,an(:,1),'b-'); grid on;
xlabel('time(sec)'), ylabel('a_1 (m/s^2)');
title('Acceleration Components of Point P in R expressed in Frame D');

subplot(3,1,2); plot(time,an(:,2),'b-'); grid on;
xlabel('time(sec)'), ylabel('a_2 (m/s^2)');

subplot(3,1,3); plot(time,an(:,3),'b-'); grid on;
xlabel('time(sec)'), ylabel('a_3 (m/s^2)');
```

The ***results*** generated by the script are shown in the following four figures. The results at $t = 2$ (sec) are ***highlighted*** using the "data cursor" or "datatip" feature available in the ***figure window*** generated by MATLAB. Note that the highlighted results are the same as those presented above.

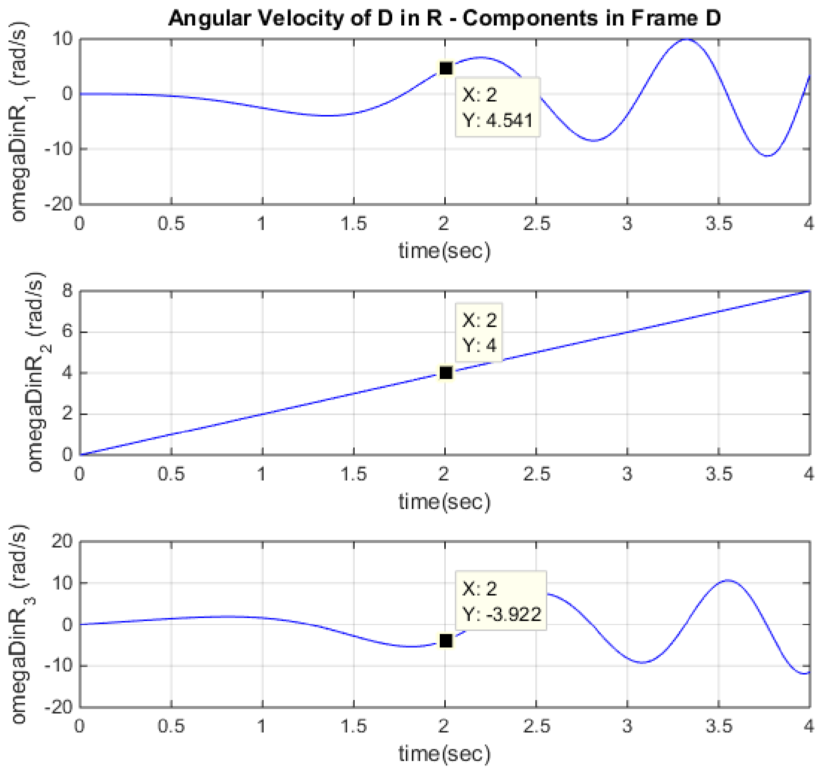


Figure 1. Angular Velocity of D in R - Components in Frame D

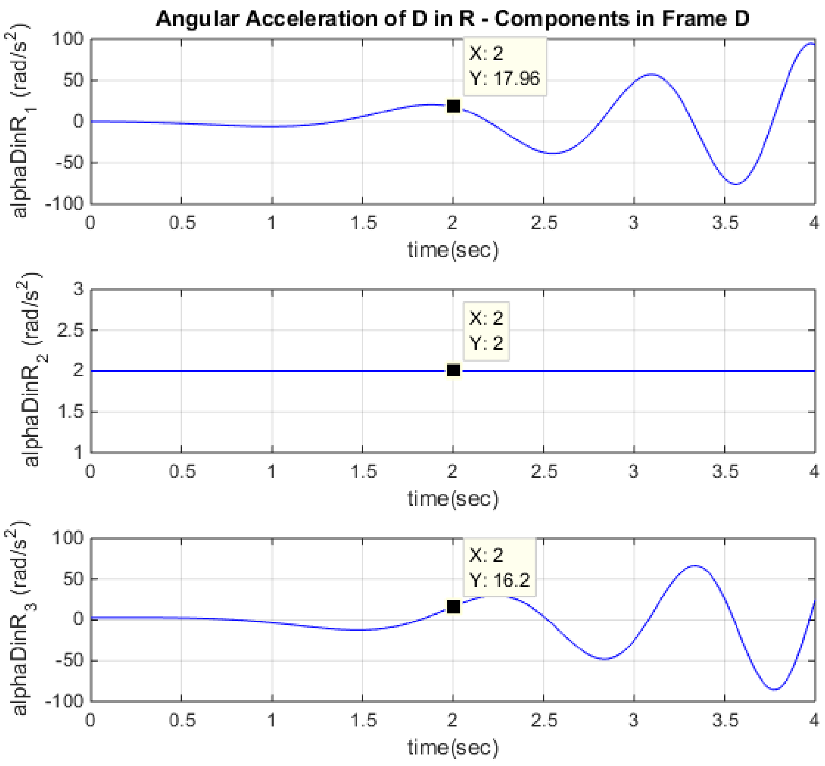


Figure 2. Angular Acceleration of D in R - Components in Frame D

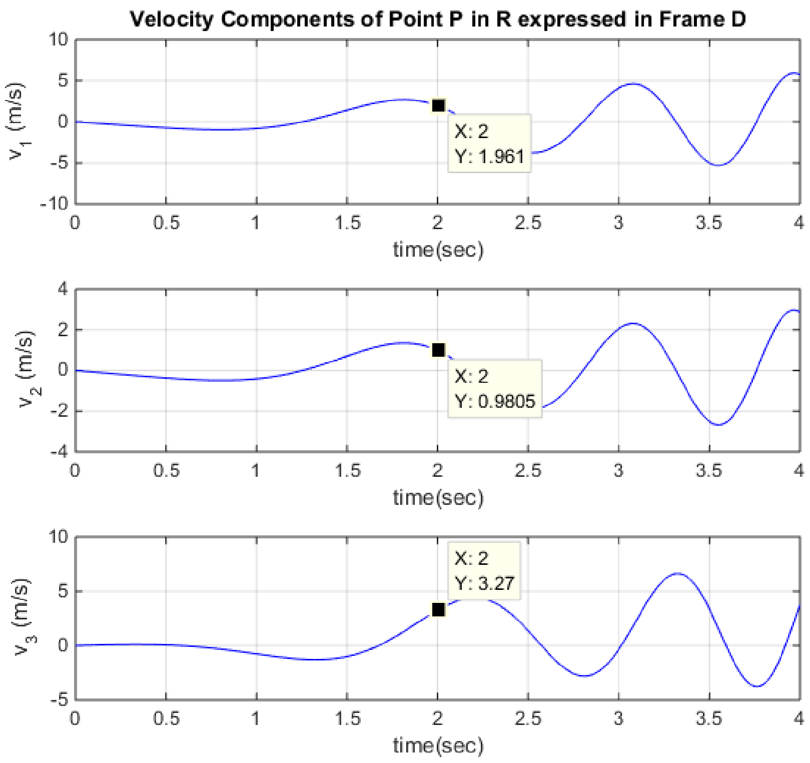


Figure 3. Velocity Components of Point P in R expressed in Frame D

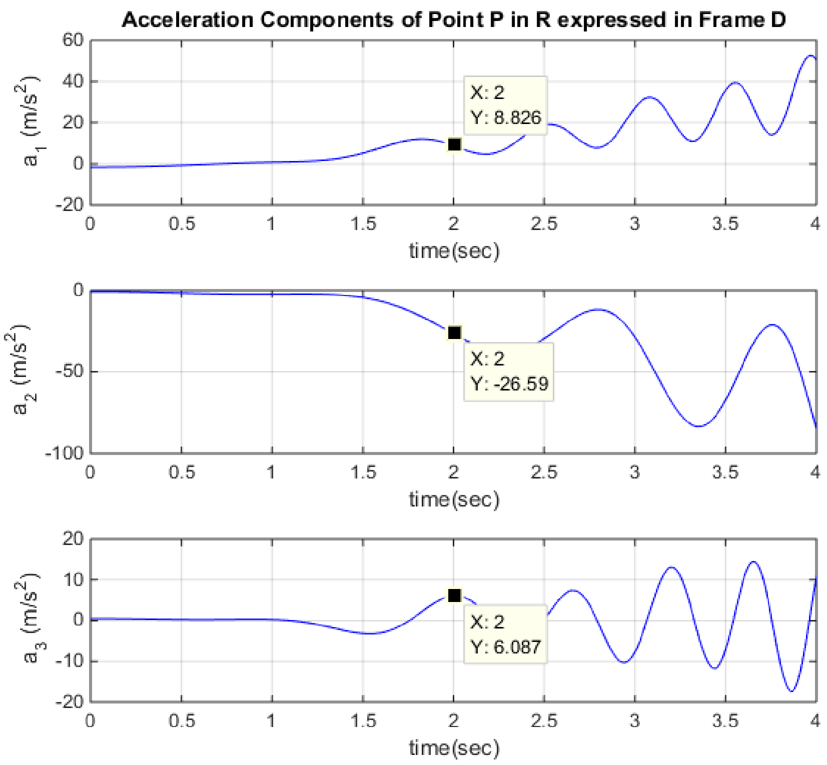
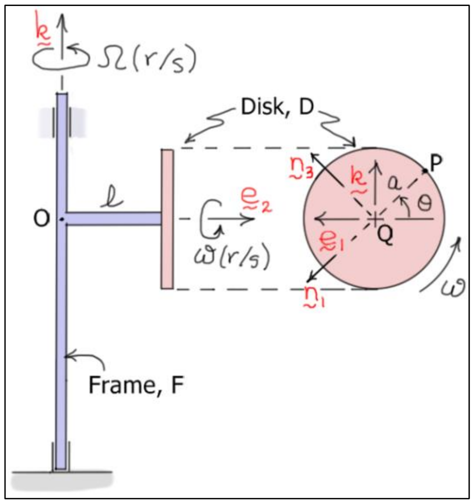


Figure 4. Acceleration Components of Point P in R expressed in Frame D

Simulink Modeling

Example 3:

In this example, the system of Example 2 is modeled using a ***MATLAB script*** in conjunction with a ***Simulink model***. As before, the column (frame, *F*) is driven relative to the ground and the disk (*D*) is driven relative to the column at ***constant angular accelerations***. Values of the lengths ℓ and a and the relative angular accelerations $\dot{\Omega}$ and $\dot{\omega}$ are as ***provided*** in Example 2.



The MATLAB script has ***four sections*** and is shown below in ***two panels***. The ***first panel*** shows the ***first three sections*** of the script. As in Example 2, the first section of the script provides a ***functional description*** of the model, and the second section ***initializes variables*** used by the model. In this case, however, the calculations are performed in Simulink rather than in a MATLAB script. The third section of the script uses the "sim" statement to execute the Simulink model.

Script – Panel 1: (functional description, initialization of variables, execution of Simulink model)

```
%% Script for the Simulink model:
%     Unit10TwoBodyRelativeVelocityAccelerationExample03
%
%  This script sets values for the Simulink model of Unit 10, Example 3 a two-body
%  system with applied motions and then executes the model.
%
%  The model calculates the angular velocity and angular acceleration of the disk D
%  and the velocity and acceleration of point P. The components are resolved in the
%  disk frame.

%% Give values to the required variables

   armLength  = 0.5;    % (m)
   diskRadius = 0.25;   % (m)

   omegaDot     = 2.0;  % (r/s^2) ... constant
   omegaInitial = 0.0;  % (r/s)
   thetaInitial = 0.0;  % (rad)

   capOmegaDot     = 3.0;  % (r/s^2) ... constant
   capOmegaInitial = 0.0;  % (r/s)
   phiInitial      = 0.0;  % (rad)

%% Execute the Simulink model
   sim('Unit10TwoBodyKinematicsExample03');
                                       ⋮
```

The final section of the script simply plots the results generated by the Simulink code. The output for each of the vectors, ${}^{R}\underset{\sim}{\omega}_{D}$, ${}^{R}\underset{\sim}{\alpha}_{D}$, ${}^{R}\underset{\sim}{v}_{P}$, and ${}^{R}\underset{\sim}{a}_{P}$ are stored in the MATLAB workspace as $N \times 4$ arrays. The first column of each array holds the time values, and the last three columns hold the values of the $\underset{\sim}{n}_1$, $\underset{\sim}{e}_2$, and $\underset{\sim}{n}_3$ components of the vectors. Here, N represents the number of time values in the simulation. As with the script for Example 3, the results are presented in four figures, each containing plots of the three components of the vector over time.

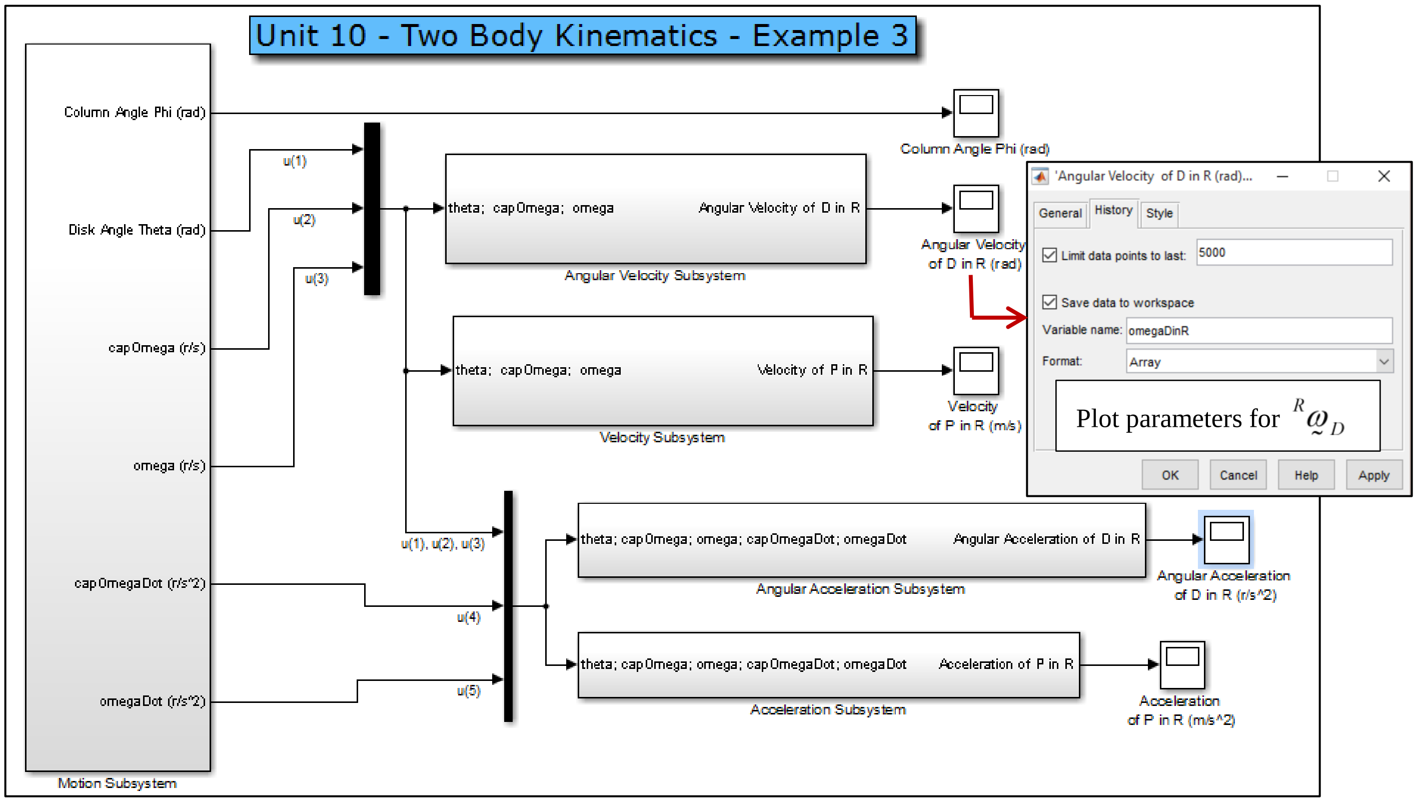
Script – Panel 2: (Plotting of results in four figure windows)

```
                                    ⋮
%% Plot the results

figure(1); clf;
subplot(3,1,1); plot(omegaDinR(:,1),omegaDinR(:,2),'b-'); grid on;
xlabel('time(sec)'), ylabel('omegaDinR_1 (rad/s)');
title('Angular Velocity of D in R - Components in Frame D');

subplot(3,1,2); plot(omegaDinR(:,1),omegaDinR(:,3),'b-'); grid on;
xlabel('time(sec)'), ylabel('omegaDinR_2 (rad/s)');

subplot(3,1,3); plot(omegaDinR(:,1),omegaDinR(:,4),'b-'); grid on;
xlabel('time(sec)'), ylabel('omegaDinR_3 (rad/s)');

figure(2); clf;
subplot(3,1,1); plot(alphaDinR(:,1),alphaDinR(:,2),'b-'); grid on;
xlabel('time(sec)'), ylabel('alphaDinR_1 (rad/s^2)');
title('Angular Acceleration of D in R - Components in Frame D');

subplot(3,1,2); plot(alphaDinR(:,1),alphaDinR(:,3),'b-'); grid on;
xlabel('time(sec)'), ylabel('alphaDinR_2 (rad/s^2)');

subplot(3,1,3); plot(alphaDinR(:,1),alphaDinR(:,4),'b-'); grid on;
xlabel('time(sec)'), ylabel('alphaDinR_3 (rad/s^2)');

figure(3); clf;
subplot(3,1,1); plot(vn(:,1),vn(:,2),'b-'); grid on;
xlabel('time(sec)'), ylabel('v_1 (m/s)');
title('Velocity Components of Point P in R expressed in Frame D');

subplot(3,1,2); plot(vn(:,1),vn(:,3),'b-'); grid on;
xlabel('time(sec)'), ylabel('v_2 (m/s)');

subplot(3,1,3); plot(vn(:,1),vn(:,4),'b-'); grid on;
xlabel('time(sec)'), ylabel('v_3 (m/s)');

figure(4); clf;
subplot(3,1,1); plot(an(:,1),an(:,2),'b-'); grid on;
xlabel('time(sec)'), ylabel('a_1 (m/s^2)');
title('Acceleration Components of Point P in R expressed in Frame D');

subplot(3,1,2); plot(an(:,1),an(:,3),'b-'); grid on;
xlabel('time(sec)'), ylabel('a_2 (m/s^2)');

subplot(3,1,3); plot(an(:,1),an(:,4),'b-'); grid on;
xlabel('time(sec)'), ylabel('a_3 (m/s^2)');
```

Simulink Model – Top Layer:

The ***top layer*** of the Simulink model shows the ***overall operation*** of the model and is shown in the panel below. The subsystem on the left side of the diagram generates values for the ***motion variables*** – ϕ, $\Omega\left(=\dot{\phi}\right)$, $\dot{\Omega}$, θ, $\omega\left(=\dot{\theta}\right)$, and $\dot{\omega}$. Some or all of these variables are then passed to the subsystems that calculate the values of the ***disk-fixed components*** of the vectors ${}^R\underset{\sim}{\omega}_D$, ${}^R\underset{\sim}{v}_P$, ${}^R\underset{\sim}{\alpha}_D$, and ${}^R\underset{\sim}{a}_P$. The values of these variables are plotted and then stored in the MATLAB workspace using Simulink scopes. The History tab of the Parameters menu of each scope determines the variable name and format to be used. The window shown below for the angular velocity scope indicates that an array with the name "omegaDinR" will be generated. This array will be an $N\times 4$ array with N indicating the number of time values (maximum number of times is set to 5000 – the last 5000 calculated). As noted earlier, the first column of this array holds the time values, and the last three columns hold the values of the vector components.

The ***second panel*** below shows the "model configuration parameters" found on the "simulation menu" of the model. The parameters shown are for the Simulink solver and indicate the simulation will run from $0\rightarrow 4\ (\text{sec})$ in fixed steps of $0.01\ (\text{sec})$. When ***numerical integration*** is required, a ***fourth-order Runge-Kutta method*** is used. Note that any numerical values shown here could have been set using variables from the MATLAB script. Given these values, the arrays generated in the MATLAB workspace will be 401×4.


Unit 10 - Two Body Kinematics - Example 3
Column Angle Phi (rad)
Disk Angle Theta (rad)
capOmega (r/s)
omega (r/s)
capOmegaDot (r/s^2)
omegaDot (r/s^2)
Motion Subsystem
u(1)
u(2)
u(3)
theta; capOmega; omega
Angular Velocity of D in R
Angular Velocity Subsystem
theta; capOmega; omega
Velocity of P in R
Velocity Subsystem
u(1), u(2), u(3)
u(4)
u(5)
theta; capOmega; omega; capOmegaDot; omegaDot
Angular Acceleration of D in R
Angular Acceleration Subsystem
theta; capOmega; omega; capOmegaDot; omegaDot
Acceleration of P in R
Acceleration Subsystem
Column Angle Phi (rad)
Angular Velocity of D in R (rad)
Velocity of P in R (m/s)
Angular Acceleration of D in R (r/s^2)
Acceleration of P in R (m/s^2)
'Angular Velocity of D in R (rad)...
General
History
Style
OK
Cancel
Help
Apply

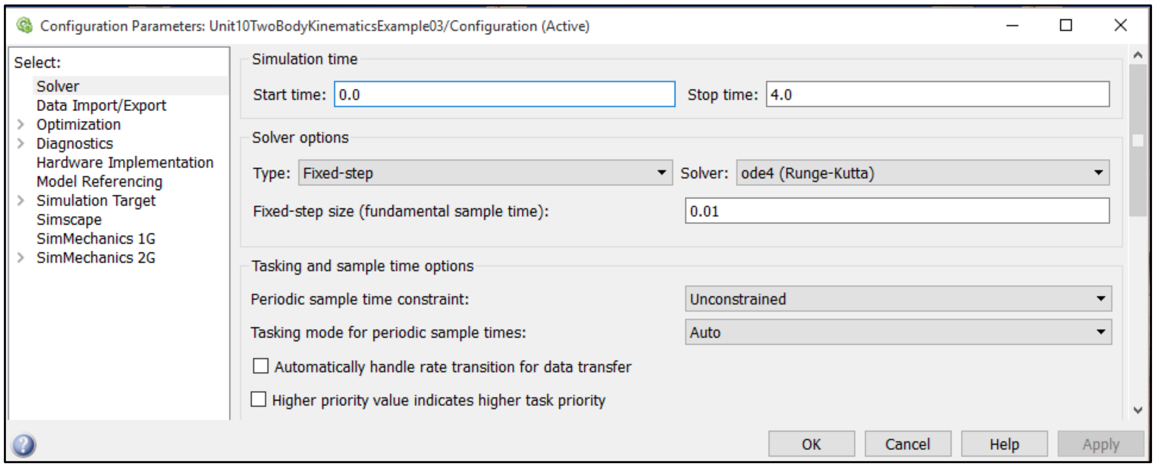
[✓] Limit data points to last: 5000

[✓] Save data to workspace

Variable name: omegaDinR

Format: Array

Plot parameters for ${}^R\underset{\sim}{\omega}_D$



Configuration Parameters: Unit10TwoBodyKinematicsExample03/Configuration (Active)

Select:
- Solver
- Data Import/Export
- Optimization
- Diagnostics
- Hardware Implementation
- Model Referencing
- Simulation Target
- Simscape
- SimMechanics 1G
- SimMechanics 2G

Simulation time

Start time: 0.0 Stop time: 4.0

Solver options

Type: Fixed-step Solver: ode4 (Runge-Kutta)

Fixed-step size (fundamental sample time): 0.01

Tasking and sample time options

Periodic sample time constraint: Unconstrained

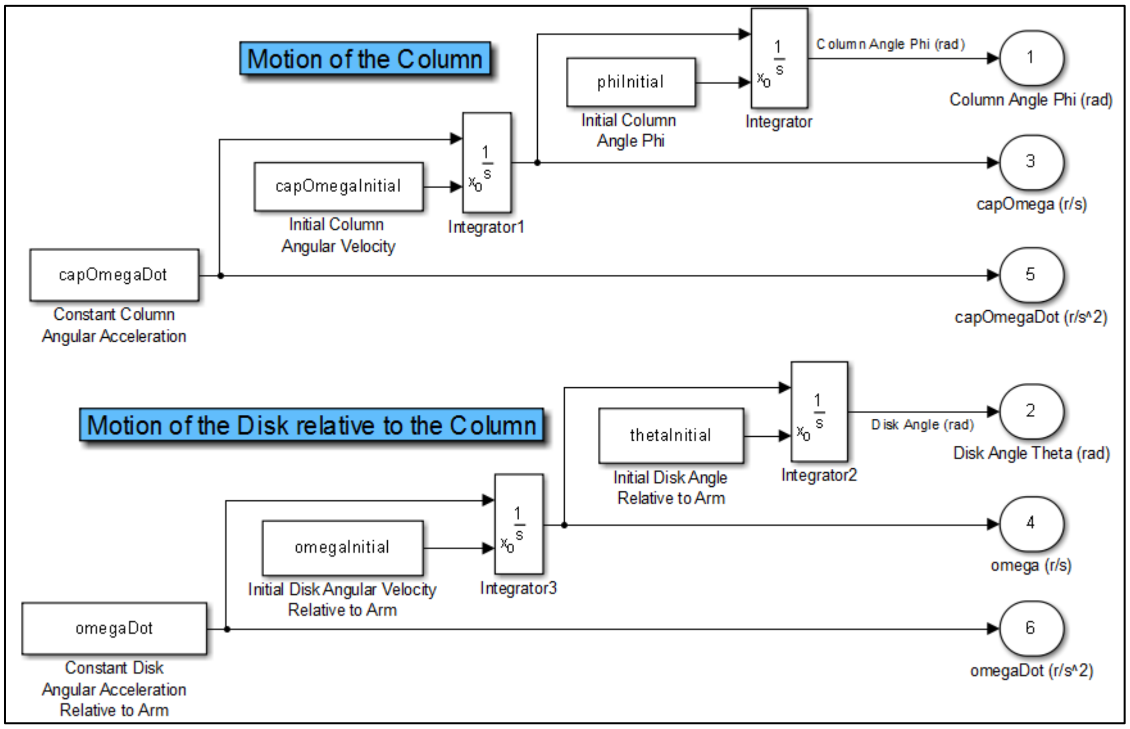
Tasking mode for periodic sample times: Auto

☐ Automatically handle rate transition for data transfer

☐ Higher priority value indicates higher task priority

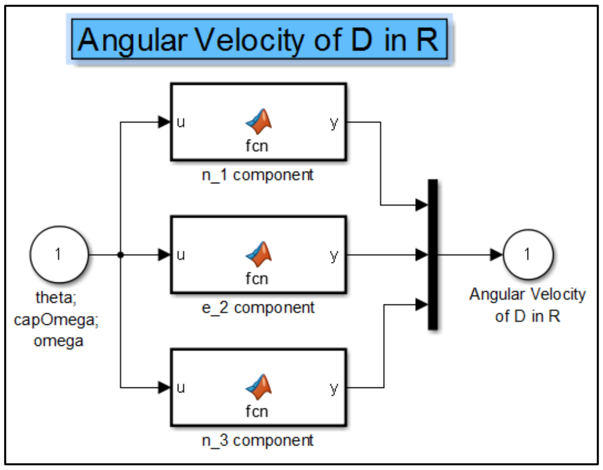
Simulink Model – Motion Subsystem:

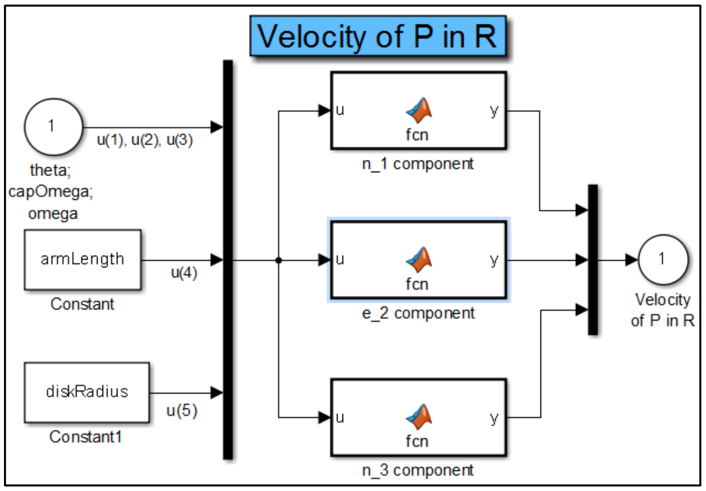
The ***motion subsystem*** calculates the ***relative angular motions*** of the system ***assuming*** the relative angular accelerations $(\dot{\Omega}, \dot{\omega})$ are ***constant***. These values are ***integrated*** once using the given ***initial relative angular velocities*** (variables "capOmegaInitial" and "omegaInitial" from the script) to calculate the ***relative angular velocities***, and then ***integrated again*** using the given ***initial relative angles*** (variables "phiInitial" and "thetaInitial" from the script) to calculate the ***angles***. The integrators are labeled by Simulink using the symbol "$\frac{1}{s}$" (in reference to integrations in the Laplace domain), and the ports for the initial values are indicated by the symbol "x_0". See the diagram below. Clearly, ***more complicated motion profiles*** could be generated using this process by simply expressing the relative angular accelerations themselves as functions of time.

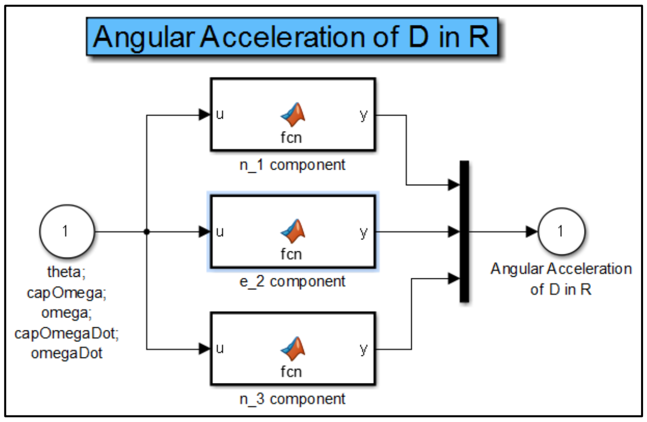


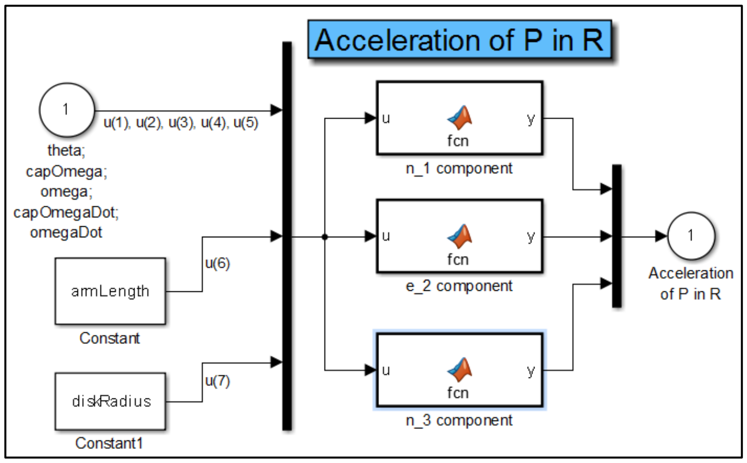
Simulink Model – Vector Component Subsystems:

The motion subsystem generates vector inputs for each of the vector component subsystems. The angular velocity and velocity subsystems each receive a 3×1 vector containing values for θ, Ω, and ω, and the angular acceleration and acceleration subsystems each receive a 5×1 vector containing values for θ, Ω, ω, $\dot{\Omega}$, and $\dot{\omega}$ as indicated in the diagram. These values alone are sufficient to calculate the vector components of the angular velocity and angular acceleration vectors. However, values of the arm length ℓ and the disk radius a are also needed to calculate the velocity and acceleration vectors. Values for these variables are set using the MATLAB script variables "armLength" and "diskRadius". These values are appended to the vector received from the motion subsystem before the vector components are calculated. See the diagrams below. In each subsystem, the vector components are returned as 3×1 vectors.






Angular Acceleration of D in R
1
theta; capOmega; omega; capOmegaDot; omegaDot
u fcn y
n_1 component
e_2 component
n_3 component
1
Angular Acceleration of D in R



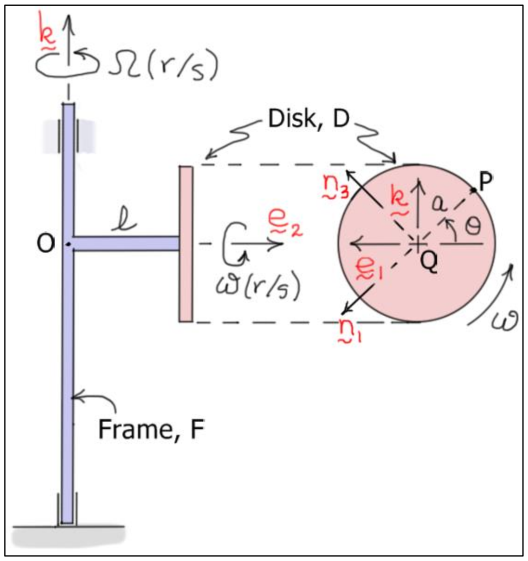
Each of the vector components are computed using MATLAB functions. The contents of the three functions used to calculate the acceleration vector components are shown in the next panel. Based on the block diagrams, each acceleration function receives a 7×1 vector "u" containing values of θ, Ω, ω, $\dot{\Omega}$, $\dot{\omega}$, ℓ, and a. The functions are MATLAB scripts with input vector "u" and output vector "y".

```
function y = fcn(u)
%#codegen
% n_1 component of the acceleration of P in R
%
y = (-u(6)*u(4)*cos(u(1))) + (u(7)*(u(3)^2)) + (u(7)*((u(2)*cos(u(1)))^2));

---------------------------------------------------------------------------
function y = fcn(u)
%#codegen
% e_2 component of the acceleration of P in R
%
y = (-u(7)*u(4)*cos(u(1))) + (2*u(7)*u(2)*u(3)*sin(u(1))) - (u(6)*(u(2)^2));

----------------------------------------------------------------------------
function y = fcn(u)
%#codegen
% n_3 component of the acceleration of P in R
%
y = (u(7)*u(5)) - (u(6)*u(4)*sin(u(1))) + (u(7)*(u(2)^2)*sin(u(1))*cos(u(1)));
```

The ***results*** generated by this Simulink model are ***identical*** to those generated by the MATLAB script of Example 2. As with the script, the programmer has full control over the equations used to model the system behavior. Simulink does, however, provide an ***alternative*** to script programming that some may find ***more intuitive***. Even though this is a ***very simple*** model, it is clear from this example that Simulink provides a more ***visual representation*** of the ***functionality*** of a model than a MATLAB script. ***Data generation*** and ***flow*** within a model is ***made clear*** by the ***block diagram***. In this regard, it is easier for someone viewing the model for the first time to ***understand the process*** the model represents.

Example 4: SimMechanics Modeling

In this example, the system of Example 2 is modeled again, this time using a ***MATLAB script*** in conjunction with a ***SimMechanics model***. As before, the column (frame, *F*) is driven relative to the ground and the disk *D* is driven relative to the column at ***constant angular accelerations***. Values of the lengths ℓ and a and the relative angular accelerations $\dot{\Omega}$ and $\dot{\omega}$ are as ***provided*** in Example 2.



The MATLAB script has ***four sections*** and is shown below in ***three panels***. The ***first panel*** shows the ***first section*** of the script which provides a ***functional description*** of the model and initializes ***variables*** used by the model.

In this example, the model is ***more detailed*** than in the previous two examples. In addition to the ***motion data***, it also includes ***geometric***, ***mass***, and ***inertia data*** for the bodies, and a ***stopping time*** and ***time***

increment. Also, the frame F is treated as two bodies ***welded together*** (to form a single body) – a ***vertical column*** and a ***horizontal arm***. For the purpose of calculating inertia matrices, the column and arm are assumed to be ***right circular cylinders*** and the disk is assumed to be ***thin***.

Script – Panel 1: (functional description and initialization of variables)

```
%%  Unit 10 - Example 4: Script for the SimMechanics model:
%                Unit10TwoBodyKinematicsExample04
%
%  This script sets values for the SimMechanics model of Unit 10, Example 4 a two-
%  body system with applied motions and then executes the model.
%
%  The model calculates the angular velocity and angular acceleration of the disk D
%  and the velocity and acceleration of point P. The components are resolved in the
%  disk frame. All results are plotted and some are displayed in the MATLAB
%  command window.

   timeMax       = 4.0; %sec
   timeIncrement = 0.01; %sec

%  Column Data
   columnMass   = 1.0;    %kg
   columnLength = 1.5;    %meters
   columnRadius = 0.05;   %meters

   Ixx = columnMass*((3*(columnRadius^2))+(columnLength^2))/12; Iyy = Ixx;
   Izz = 0.5*columnMass*(columnRadius^2);
   columnInertiaMatrix = [Ixx,0,0; 0,Iyy,0; 0,0,Izz];  %kg-m^2

%  Arm Data
   armMass   = 1.0;    %kg
   armLength = 0.5;    %meters
   armRadius = 0.05;   %meters

   Ixx = armMass*((3*(armRadius^2))+(armLength^2))/12; Izz = Ixx;
   Iyy = 0.05*armMass*(armRadius^2);
   armInertiaMatrix = [Ixx,0,0; 0,Iyy,0; 0,0,Izz]; %kg-m^2

%  Disk Data
   diskMass   = 1.0;     %kg
   diskRadius = 0.25;    %meters
   theta  = 30*pi/180;   %radians

   Ixx = 0.25*diskMass*(diskRadius^2); Izz = Ixx;
   Iyy = 0.50*diskMass*(diskRadius^2);
   diskInertiaMatrix = [Ixx,0,0; 0,Iyy,0; 0,0,Izz];  %kg-m^2

%  Motion Data
   omegaDot    = 2.0;      % rad/s^2 (constant)
   capOmegaDot = 3.0;      % rad/s^2 (constant)

   omegaInitial    = 0.0;  % rad/s
   capOmegaInitial = 0.0;  % rad/s
   thetaInitial    = 0.0;  % rad (thetaDot = omega)
   phiInitial      = 0.0;  % rad (phiDot = capOmega)
                                  ⋮
```

Note that values of mass and inertia are ***not necessary*** for ***kinematic calculations*** associated with the specified motions of this example; however, they are ***necessary*** if the model is ***extended*** to include calculations of the constraint and driving ***forces*** and ***torques*** associated with these motions. These types of calculations will be considered in Volume II of this text.

The next two panels show the last three sections of the script. The first section ***executes*** the SimMechanics model, the second ***plots*** results for ${}^{R}\underset{\sim}{\omega}_{D}$, ${}^{R}\underset{\sim}{\alpha}_{D}$, ${}^{R}\underset{\sim}{v}_{P}$, and ${}^{R}\underset{\sim}{a}_{P}$, and the last section ***displays*** the ***initial values*** of these vectors in the MATLAB workspace. Note the output from the SimMechanics model are ***structured arrays*** which include time signals. For example, the time signal for ${}^{R}\underset{\sim}{\omega}_{D}$ is named "`diskAngularVelocity.time`", and the signals for the *x*, *y*, and *z* components of ${}^{R}\underset{\sim}{\omega}_{D}$ are named "`diskAngularVelocity.signals(1).values`", "`diskAngularVelocity.signals(2).values`", and "`diskAngularVelocity.signals(3).values`". A similar naming convention is used for the other vectors. See the description of the SimMechanics model for more details on how these structures are generated.

Script – Panel 2: (execution of SimMechanics model with some results plotted)

```
                                 ⋮
%% Run SimMechanics Model
   sim('Unit10TwoBodyKinematicsExample04.slx');

%% Plot Output
%  Angular Velocity Components of Disk, D (rad/s)
   figure(1); clf; subplot(3,1,1);
   plot(diskAngularVelocity.time,diskAngularVelocity.signals(1).values);
   grid; title('Body-Fixed Angular Velocity Components of Disk, D (rad/s)');
   xlabel('Time (sec)'); ylabel('X-Component (rad/s)')

   subplot(3,1,2);
   plot(diskAngularVelocity.time,diskAngularVelocity.signals(2).values);
   grid; xlabel('Time (sec)'); ylabel('Y-Component (rad/s)')

   subplot(3,1,3);
   plot(diskAngularVelocity.time,diskAngularVelocity.signals(3).values);
   grid; xlabel('Time (sec)'); ylabel('Z-Component (rad/s)')

%  Angular Acceleration Components of Disk, D (rad/s)
    figure(2); clf; subplot(3,1,1);
    plot(diskAngularAcceleration.time,diskAngularAcceleration.signals(1).values);
    grid; title('Body-Fixed Angular Acceleration Components of Disk, D (rad/s^2)');
    xlabel('Time (sec)'); ylabel('X-Component (rad/s^2)')

    subplot(3,1,2);
    plot(diskAngularAcceleration.time,diskAngularAcceleration.signals(2).values);
    grid; xlabel('Time (sec)'); ylabel('Y-Component (rad/s^2)')

    subplot(3,1,3);
    plot(diskAngularAcceleration.time,diskAngularAcceleration.signals(3).values);
    grid; xlabel('Time (sec)'); ylabel('Z-Component (rad/s^2)')
```

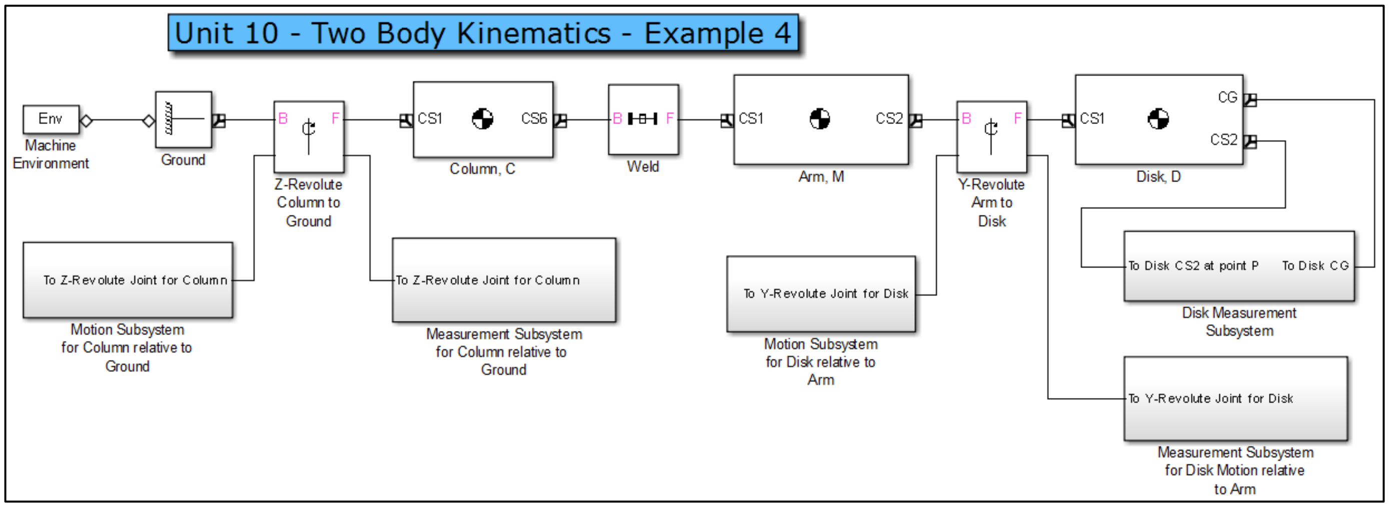
Script – Panel 3: (more results plotted and initial values of results displayed in command window)

```
                                    ⋮
%  Velocity Components of Point P (m/s)
   figure(3); clf;
   subplot(3,1,1); plot(velocityPointP.time,velocityPointP.signals(1).values);
   grid; title('Body-Fixed Components of the Velocity of Point P (m/s)');
   xlabel('Time (sec)'); ylabel('X-Component (m/s)')

   subplot(3,1,2); plot(velocityPointP.time,velocityPointP.signals(2).values);
   grid; xlabel('Time (sec)'); ylabel('Y-Component (m/s)')

   subplot(3,1,3); plot(velocityPointP.time,velocityPointP.signals(3).values);
   grid; xlabel('Time (sec)'); ylabel('Z-Component (m/s)')

%  Acceleration Components of Point P (m/s^2)
   figure(4); clf; subplot(3,1,1);
   plot(accelerationPointP.time,accelerationPointP.signals(1).values);
   grid; title('Body-Fixed Components of the Acceleration of Point P (m/s^2)');
   xlabel('Time (sec)'); ylabel('X-Component (m/s^2)')

   subplot(3,1,2);
   plot(accelerationPointP.time,accelerationPointP.signals(2).values);
   grid; xlabel('Time (sec)'); ylabel('Y-Component (m/s^2)')

   subplot(3,1,3);
   plot(accelerationPointP.time,accelerationPointP.signals(3).values);
   grid; xlabel('Time (sec)'); ylabel('Z-Component (m/s^2)')

%% Initial angular velocity and angular acceleration of Disk D
%  and Initial velocity and acceleration of point P

   disp('Body-Fixed Components of the Initial Angular Velocity of Disk D (r/s)')
   disp('X-Component'),disp(diskAngularVelocity.signals(1).values(1))
   disp('Y-Component'),disp(diskAngularVelocity.signals(2).values(1))
   disp('Z-Component'),disp(diskAngularVelocity.signals(3).values(1))

   disp('Body-Fixed Components of the Initial Angular Acceleration of Disk D (r/s^2)')
   disp('X-Component'),disp(diskAngularAcceleration.signals(1).values(1))
   disp('Y-Component'),disp(diskAngularAcceleration.signals(2).values(1))
   disp('Z-Component'),disp(diskAngularAcceleration.signals(3).values(1))

   disp('Body-Fixed Components of the Initial Velocity of Point P (m/s)')
   disp('X-Component'),disp(velocityPointP.signals(1).values(1))
   disp('Y-Component'),disp(velocityPointP.signals(2).values(1))
   disp('Z-Component'),disp(velocityPointP.signals(3).values(1))

   disp('Body-Fixed Components of the Initial Acceleration of Point P (m/s^2)')
   disp('X-Component'),disp(accelerationPointP.signals(1).values(1))
   disp('Y-Component'),disp(accelerationPointP.signals(2).values(1))
   disp('Z-Component'),disp(accelerationPointP.signals(3).values(1))
```

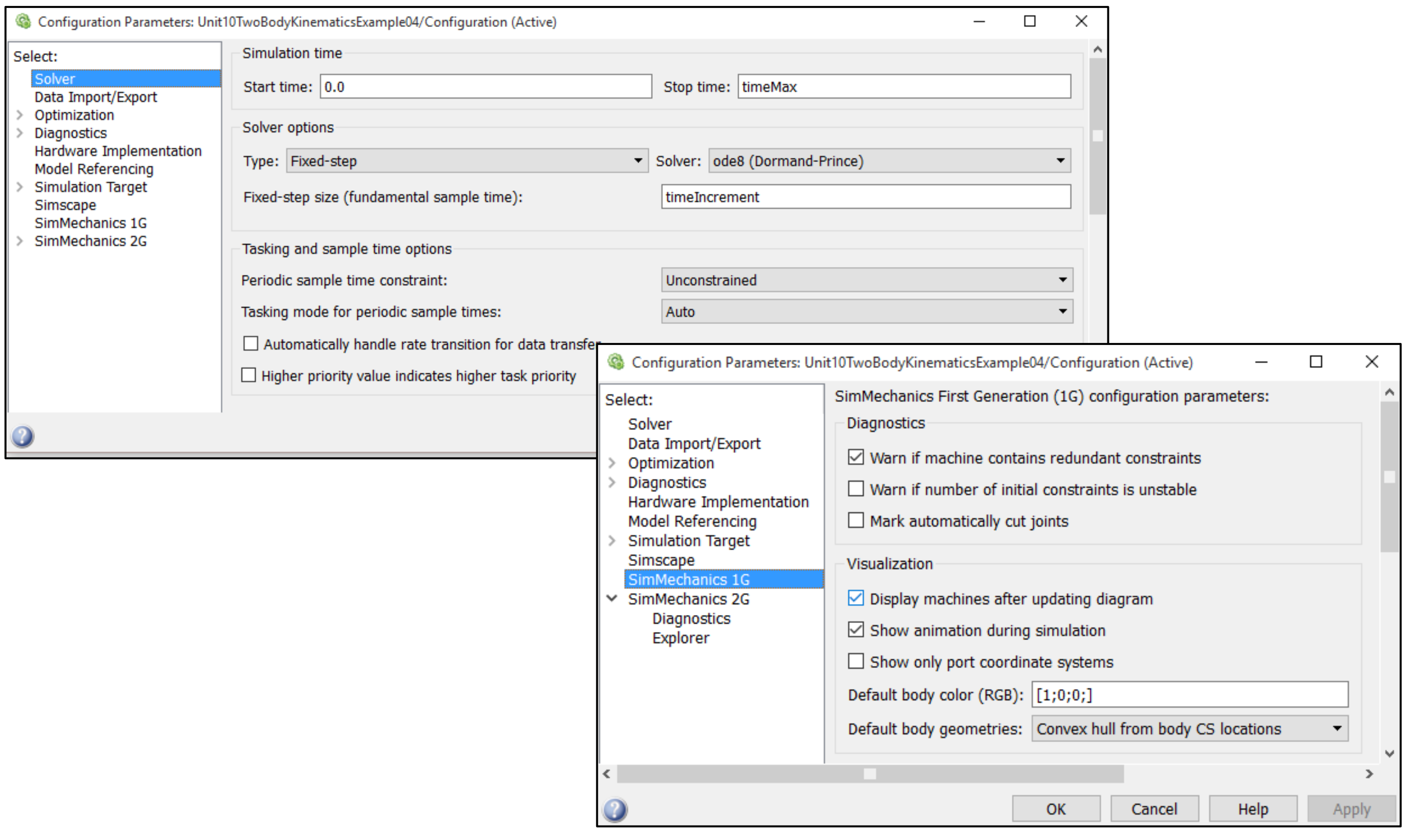
SimMechanics Model – Top Layer:

The ***Simulink model*** presented in Example 3 focused on ***calculations*** based on a set of previously derived model equations. The ***block diagram*** represents the ***process*** needed to solve the model equations. In contrast, a ***SimMechanics block diagram model*** focuses on the ***physical system***. SimMechanics then ***formulates*** and ***solves*** the model equations. Data can be moved seamlessly between SimMechanics and Simulink using ***actuators*** and ***sensors***. ***Simulink signals*** can be used to drive a system using ***actuator blocks***, and ***SimMechanics signals*** can measured (and sent to Simulink) using ***sensor blocks***.

The top layer of the SimMechanics model of the system of this example is shown in the panel below. The model has a ***Machine Environment*** block, a ***Ground*** block, three ***Body*** blocks, three ***Joint*** blocks, two ***Motion*** subsystems, and three ***Measurement*** subsystems. The diagram generally indicates how the bodies are connected and where the applied motions and measurements occur. More specific details of each block within the model are presented below.


Unit 10 - Two Body Kinematics - Example 4
Env
Machine Environment
Ground
B
F
Z-Revolute Column to Ground
CS1
CS6
Column, C
Weld
CS1
CS2
Arm, M
Y-Revolute Arm to Disk
CS1
CG
CS2
Disk, D
To Z-Revolute Joint for Column
Motion Subsystem for Column relative to Ground
To Z-Revolute Joint for Column
Measurement Subsystem for Column relative to Ground
To Y-Revolute Joint for Disk
Motion Subsystem for Disk relative to Arm
To Disk CS2 at point P
To Disk CG
Disk Measurement Subsystem
To Y-Revolute Joint for Disk
Measurement Subsystem for Disk Motion relative to Arm

The ***two panels*** below show the "model configuration parameters" found on the "simulation menu" of the model. Parameters are shown for the Simulink solver and for the SimMechanics first generation (1G) model. The Simulink solver parameters indicate the simulation will start at time $t = 0$ and stop at the time indicated by the variable "timeMax". ***Numerical integration*** of the model equations will occur in ***fixed time steps*** indicated by the variable "timeIncrement" and be done using an ***8th order, Dormand-Prince algorithm***. Values for the variables "timeMax" and "timeIncrement" are set in the MATLAB script. The configuration parameters for the SimMechanics model instruct SimMechanics to provide ***warnings*** to the user for any ***redundant constraints***, ***display*** the system in an animation window after the user ***updates*** the ***block diagram***, and to ***display an animation*** of the motion of the system ***during*** the simulation.


Configuration Parameters: Unit10TwoBodyKinematicsExample04/Configuration (Active)

Select: Solver, Data Import/Export, Optimization, Diagnostics, Hardware Implementation, Model Referencing, Simulation Target, Simscape, SimMechanics 1G, SimMechanics 2G

Simulation time

Start time: 0.0 Stop time: timeMax

Solver options

Type: Fixed-step Solver: ode8 (Dormand-Prince)

Fixed-step size (fundamental sample time): timeIncrement

Tasking and sample time options

Periodic sample time constraint: Unconstrained

Tasking mode for periodic sample times: Auto

☐ Automatically handle rate transition for data transfe

☐ Higher priority value indicates higher task priority

Configuration Parameters: Unit10TwoBodyKinematicsExample04/Configuration (Active)

Select: Solver, Data Import/Export, Optimization, Diagnostics, Hardware Implementation, Model Referencing, Simulation Target, Simscape, SimMechanics 1G, SimMechanics 2G (Diagnostics, Explorer)

SimMechanics First Generation (1G) configuration parameters:

Diagnostics

☑ Warn if machine contains redundant constraints

☐ Warn if number of initial constraints is unstable

☐ Mark automatically cut joints

Visualization

☑ Display machines after updating diagram

☑ Show animation during simulation

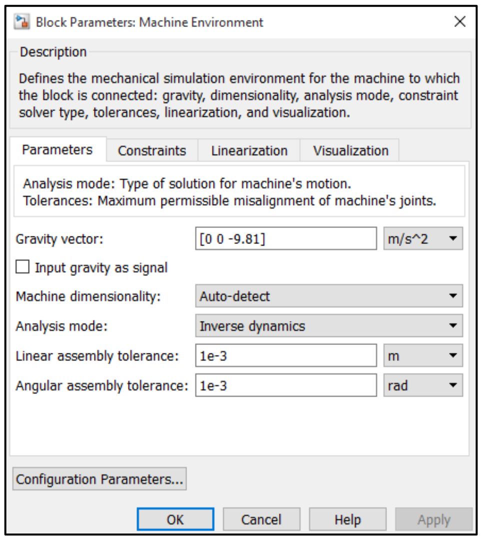
☐ Show only port coordinate systems

Default body color (RGB): [1;0;0;]

Default body geometries: Convex hull from body CS locations

The following sections provide details on the ***content*** and ***function*** of each of the blocks within the model. Double clicking on a block will open its ***dialog box***.

<u>Machine Environment Block</u>:


Block Parameters: Machine Environment

Description

Defines the mechanical simulation environment for the machine to which the block is connected: gravity, dimensionality, analysis mode, constraint solver type, tolerances, linearization, and visualization.

Analysis mode: Type of solution for machine's motion.
Tolerances: Maximum permissible misalignment of machine's joints.

Gravity vector: [0 0 -9.81] m/s^2

☐ Input gravity as signal

Machine dimensionality: Auto-detect

Analysis mode: Inverse dynamics

Linear assembly tolerance: 1e-3 m

Angular assembly tolerance: 1e-3 rad

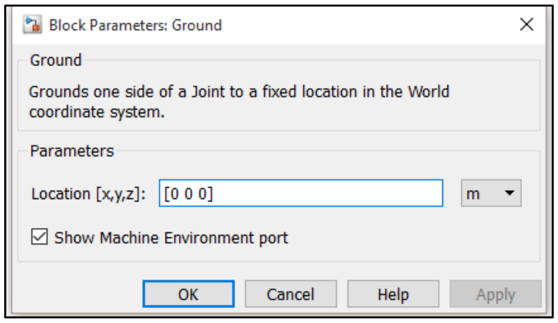
The dialog box for the Machine Environment block is displayed in the panel to the right. This box allows the user to set the "simulation environment" for the system being modeled. In this example, ***default values*** are used for entries on the Constraints, Linearization, and Visualization tabs. Under the Parameters tab, the ***Machine Dimensionality*** has been set to "Auto-detect", and the ***Analysis mode*** has been set to "Inverse Dynamics", and (although it is not needed in this example) a ***gravity vector*** has been defined to be in the ***negative Z direction*** of the global coordinate system (in metric units).

"Auto-detect" indicates that SimMechanics will ***choose*** the dimensionality of the system based on the data it receives. The other choices are "2D Only" and "3D Only". As the system is ***three dimensional***, the Machine dimensionality could also be set to "3D Only." The Analysis mode of "Inverse Dynamics" indicates that all ***fundamental motions*** are ***specified*** so all other results may be derived directly from the specified motions and the physical data provided. This example focuses on kinematic results only; however, the forces and torques required to produce the motions can also be computed. The model equations in this case are algebraic.

The other analysis options are "Forward Dynamics", "Kinematics", and "Trimming". ***Forward Dynamic*** analyses generally apply to systems for which ***all fundamental motions*** are ***not specified*** so that some parts of the system will ***respond freely*** to the specified motions and external forces (e.g. gravity) acting on the system. That is, the system has some ***unspecified degrees of freedom***. Even though the system of this example has no unspecified degrees of freedom, a forward dynamic analysis may also be used. A "Kinematics" analysis is designed for use only with ***closed-loop mechanisms***. The system of this example is an ***open-tree system*** with no closed loops, so a "Kinematics" analysis may ***not*** be used. A "Trimming" analysis is used to find equilibrium (or steady state) configurations about which the model equations can be ***linearized*** and does not apply to this example.

Ground block:

The dialog box for the Ground block is shown in the panel at the right. In this example, the Ground block serves ***two purposes***. It ***specifies*** the ***location*** in the world (global) coordinate system where the first revolute joint is located – in this case, the revolute joint is located at the ***origin*** of the world system. The ***check box*** at the bottom of the dialog box allows the user to show a ***Machine Environment port*** to which a Machine Environment block can be attached.


Block Parameters: Ground

Ground

Grounds one side of a Joint to a fixed location in the World coordinate system.

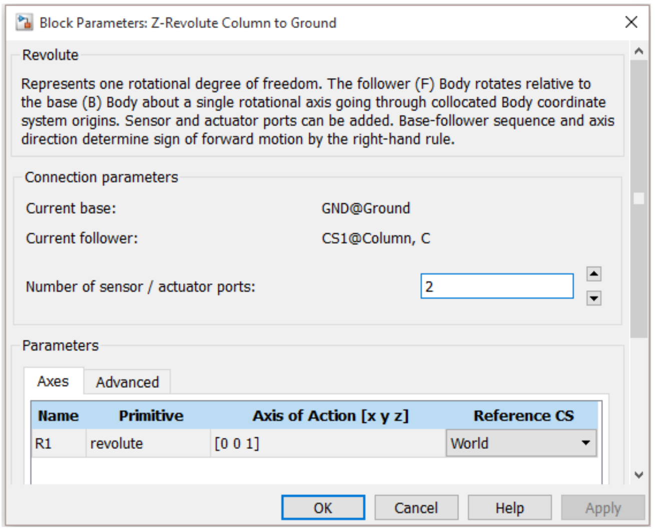
Parameters

Location [x,y,z]: [0 0 0] m

☑ Show Machine Environment port

Z-Revolute: Column to Ground

The dialog box for the Z-revolute joint connecting the column to the ground is shown in the panel to the right. The "Axes" tab indicates the joint is an "R1" or ***single degree-of-freedom revolute joint*** about the direction "[0 0 1]" (or *z*-axis) of the "World" coordinate system (CS). The middle portion of the box indicates the "base" port of the joint is connected to the "Ground" block, the "follower" port is connected to the ***coordinate system*** CS1 of the column and that ***two sensor/actuator ports*** will be shown on the block.


Block Parameters: Z-Revolute Column to Ground

Revolute

Represents one rotational degree of freedom. The follower (F) Body rotates relative to the base (B) Body about a single rotational axis going through collocated Body coordinate system origins. Sensor and actuator ports can be added. Base-follower sequence and axis direction determine sign of forward motion by the right-hand rule.

Connection parameters

Current base: GND@Ground

Current follower: CS1@Column, C

Number of sensor / actuator ports: 2

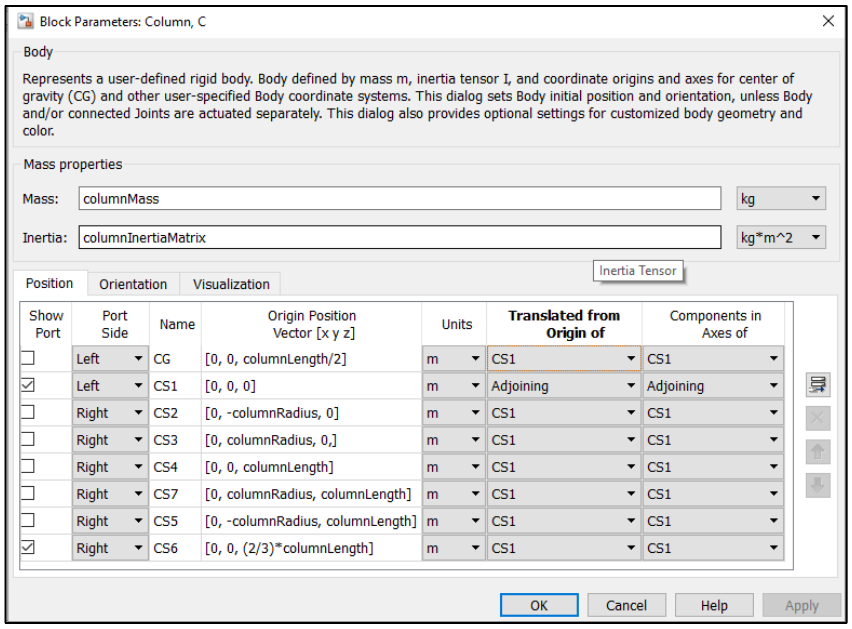
Parameters

Name	Primitive	Axis of Action [x y z]	Reference CS
R1	revolute	[0 0 1]	World

Note from the diagram of the ***top layer*** of the model that ***each joint block*** has “base” and “follower” ports. The base port is indicated in the block diagram with a ***red*** “B”, and the follower port is indicated with a ***red*** “F”. Note also that SimMechanics requires the ***points on either side*** of the joint block to be ***collocated***. In this case, it means that the ***coordinate system*** CS1 of the column must be located at the ***origin*** of the world coordinate system to ***match*** the location of the ground block. If this condition is not met, SimMechanics will stop with an error message.

Body block: Column, C

The dialog box for the body block associated with the column is shown in the panel below. As shown, values of the ***mass*** and the 3×3 ***inertia matrix*** of the column are given by the variables “columnMass” and “columnInertiaMatrix”. The values for these variables are set in the MATLAB script.


Block Parameters: Column, C

Body

Represents a user-defined rigid body. Body defined by mass m, inertia tensor I, and coordinate origins and axes for center of gravity (CG) and other user-specified Body coordinate systems. This dialog sets Body initial position and orientation, unless Body and/or connected Joints are actuated separately. This dialog also provides optional settings for customized body geometry and color.

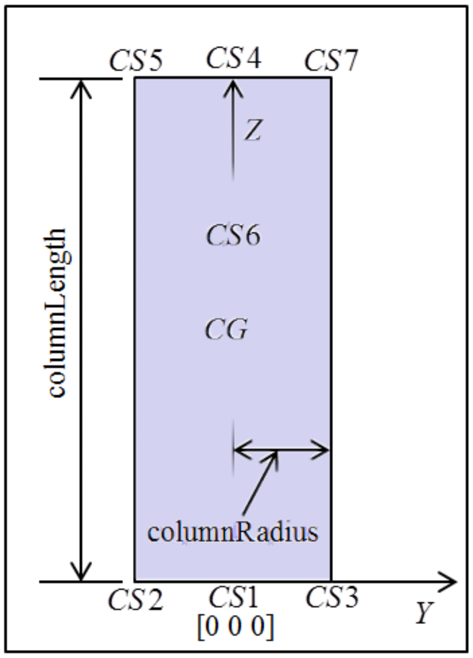
Mass properties

Mass:	columnMass	kg
Inertia:	columnInertiaMatrix	kg*m^2

Inertia Tensor

Position | Orientation | Visualization

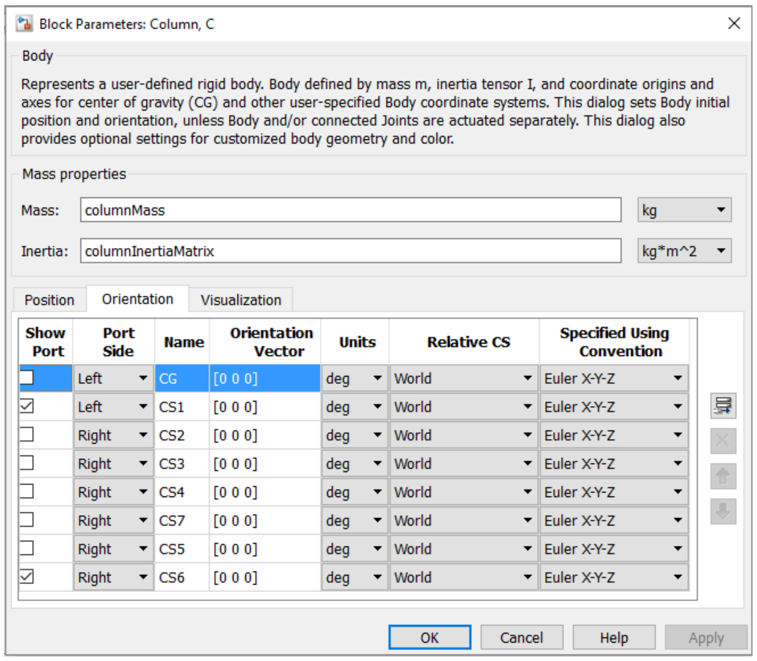
Show Port	Port Side	Name	Origin Position Vector [x y z]	Units	Translated from Origin of	Components in Axes of
☐	Left	CG	[0, 0, columnLength/2]	m	CS1	CS1
✓	Left	CS1	[0, 0, 0]	m	Adjoining	Adjoining
☐	Right	CS2	[0, -columnRadius, 0]	m	CS1	CS1
☐	Right	CS3	[0, columnRadius, 0,]	m	CS1	CS1
☐	Right	CS4	[0, 0, columnLength]	m	CS1	CS1
☐	Right	CS7	[0, columnRadius, columnLength]	m	CS1	CS1
☐	Right	CS5	[0, -columnRadius, columnLength]	m	CS1	CS1
✓	Right	CS6	[0, 0, (2/3)*columnLength]	m	CS1	CS1



The “Position” tab shows the locations of all the ***coordinate systems*** associated with the ***column***, and it shows which coordinate systems are shown as ports on the body block. The relative location of the coordinate systems is shown in the figure to the right. ***CS1*** has a port on the left side of the block and is located with the coordinates [0 0 0] relative to the ***adjoining*** Z-revolute joint, that is, it is ***collocated*** with that joint. All other coordinate systems of the column are located relative to CS1. ***CS6*** is located ***two-thirds*** of the way up the column, and it has a port on the right side of the block. Note from the ***top layer*** of the model that this is the ***location*** at which the horizontal arm is ***welded*** to the column. The coordinate system located at the mass center of the column (labeled CG) is located half-way up the column. The other systems outline the overall projected shape of the column. The ***column length*** and ***column radius*** are specified by the variables “columnLength” and “columnRadius” whose values are supplied by the MATLAB script.

The “Orientation” tab of the dialog box (shown below) indicates how each body coordinate system is oriented relative to other coordinate systems. In this case, each of the body coordinate systems is ***initially aligned*** with the “World” coordinate system. Generally, a coordinate system may be oriented relative to the “World” or “Adjoining” coordinate systems or ***any other coordinate system*** of that body. The relative orientation can be specified using a variety of ***orientation angle sequences***, ***orientation parameters***, or a 3×3 ***transformation matrix***.

The “Visualization” tab (shown below) describes how bodies will be ***viewed*** in the ***animation window***. In this example, the column will be “blue” and its shape determined using an “Equivalent ellipsoid from the mass (inertia) properties”. The other body geometry choices are: “Use machine default body geometry”, “Convex hull from body CS locations”, and “External graphics file”. As shown, the body will appear as an ellipsoid, but if the body geometry is based on the CS locations given, it will appear as a rectangle. More realistic shapes can be generated using an external graphics file.



Block Parameters: Column, C

Body

Represents a user-defined rigid body. Body defined by mass m, inertia tensor I, and coordinate origins and axes for center of gravity (CG) and other user-specified Body coordinate systems. This dialog sets Body initial position and orientation, unless Body and/or connected Joints are actuated separately. This dialog also provides optional settings for customized body geometry and color.

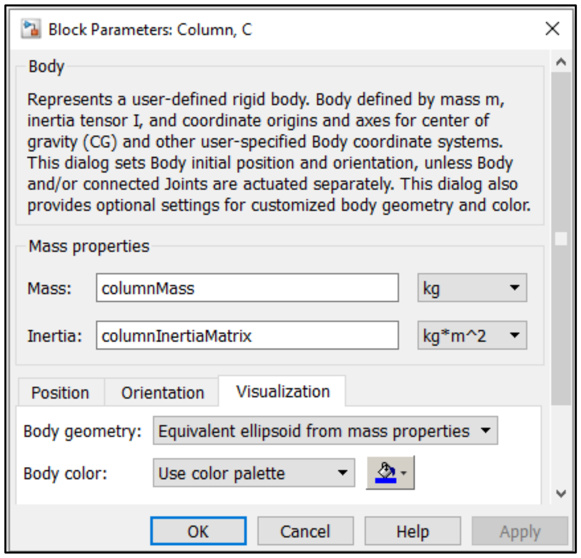
Mass properties

Mass: columnMass — kg

Inertia: columnInertiaMatrix — kg*m^2

Position | Orientation | Visualization

Show Port	Port Side	Name	Orientation Vector	Units	Relative CS	Specified Using Convention
☐	Left	CG	[0 0 0]	deg	World	Euler X-Y-Z
☑	Left	CS1	[0 0 0]	deg	World	Euler X-Y-Z
☐	Right	CS2	[0 0 0]	deg	World	Euler X-Y-Z
☐	Right	CS3	[0 0 0]	deg	World	Euler X-Y-Z
☐	Right	CS4	[0 0 0]	deg	World	Euler X-Y-Z
☐	Right	CS7	[0 0 0]	deg	World	Euler X-Y-Z
☐	Right	CS5	[0 0 0]	deg	World	Euler X-Y-Z
☑	Right	CS6	[0 0 0]	deg	World	Euler X-Y-Z



Block Parameters: Column, C

Body

Represents a user-defined rigid body. Body defined by mass m, inertia tensor I, and coordinate origins and axes for center of gravity (CG) and other user-specified Body coordinate systems. This dialog sets Body initial position and orientation, unless Body and/or connected Joints are actuated separately. This dialog also provides optional settings for customized body geometry and color.

Mass properties

Mass: columnMass — kg

Inertia: columnInertiaMatrix — kg*m^2

Position | Orientation | Visualization

Body geometry: Equivalent ellipsoid from mass properties

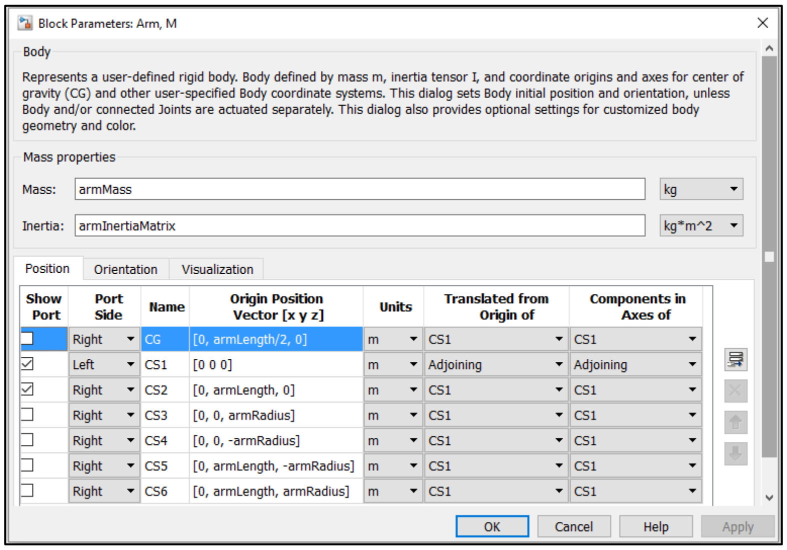
Body color: Use color palette

<u>Weld Joint</u>:

The weld joint connects coordinate system ***CS6*** of the ***column*** to coordinate system ***CS1*** of the ***arm***. This joint requires that the column and arm move as a ***single body***.

<u>Body block</u>: Arm, *M*

The dialog box for the body block associated with the horizontal arm is shown below. As shown, values of the ***mass*** and the 3×3 ***inertia matrix*** of the column are given by the variables “armMass” and “armInertiaMatrix”. The values for these variables are set in the MATLAB script. Arm coordinate system ***CS1*** is collocated with the adjoining coordinate system ***CS6*** of the ***column***. All other arm coordinate systems are located ***relative to*** CS1 and define the overall projected shape of the body. Arm coordinate system ***CS2*** is located at the ***other end of the arm*** where the disk is attached with a revolute joint, and ***CG*** is located at the midpoint. The ***orientation angles*** of all the arm coordinate systems are ***zero***, so all the arm coordinate systems are ***initially aligned*** with the ***world system***. The visualization parameters (not shown here) are the same as those for the column. Hence, the arm will be ***blue*** and will appear as an ***ellipsoid***.


Block Parameters: Arm, M

Body

Represents a user-defined rigid body. Body defined by mass m, inertia tensor I, and coordinate origins and axes for center of gravity (CG) and other user-specified Body coordinate systems. This dialog sets Body initial position and orientation, unless Body and/or connected Joints are actuated separately. This dialog also provides optional settings for customized body geometry and color.

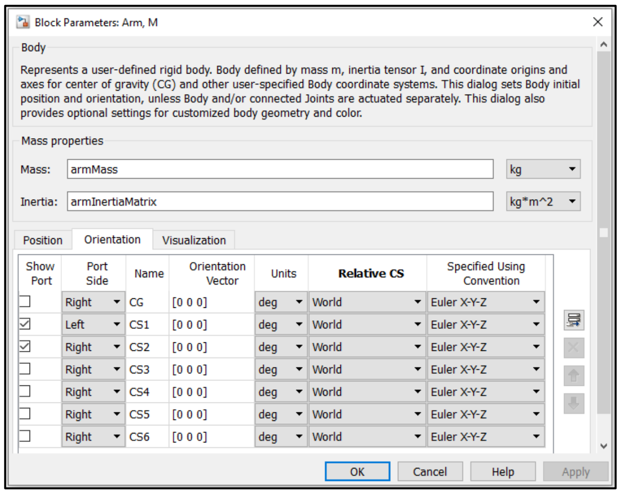
Mass properties

Mass: armMass kg

Inertia: armInertiaMatrix kg*m^2

Position | Orientation | Visualization

Show Port	Port Side	Name	Origin Position Vector [x y z]	Units	Translated from Origin of	Components in Axes of
☐	Right	CG	[0, armLength/2, 0]	m	CS1	CS1
✓	Left	CS1	[0 0 0]	m	Adjoining	Adjoining
✓	Right	CS2	[0, armLength, 0]	m	CS1	CS1
☐	Right	CS3	[0, 0, armRadius]	m	CS1	CS1
☐	Right	CS4	[0, 0, -armRadius]	m	CS1	CS1
☐	Right	CS5	[0, armLength, -armRadius]	m	CS1	CS1
☐	Right	CS6	[0, armLength, armRadius]	m	CS1	CS1


Block Parameters: Arm, M

Body

Represents a user-defined rigid body. Body defined by mass m, inertia tensor I, and coordinate origins and axes for center of gravity (CG) and other user-specified Body coordinate systems. This dialog sets Body initial position and orientation, unless Body and/or connected Joints are actuated separately. This dialog also provides optional settings for customized body geometry and color.

Mass properties

Mass: armMass kg

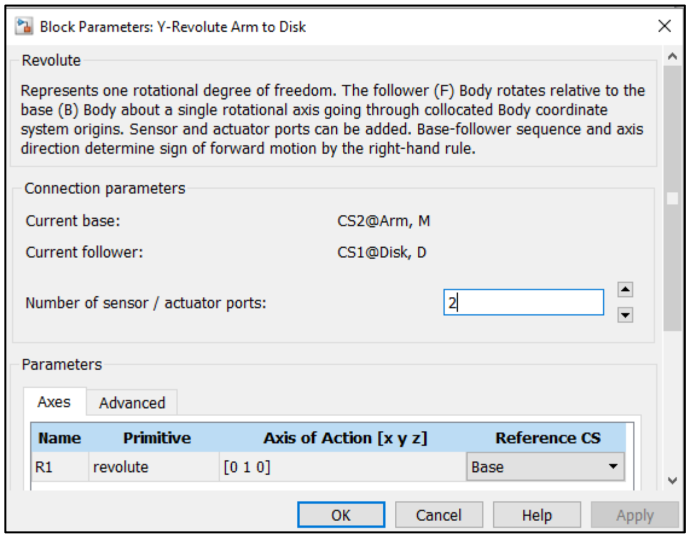
Inertia: armInertiaMatrix kg*m^2

Position | Orientation | Visualization

Show Port	Port Side	Name	Orientation Vector	Units	Relative CS	Specified Using Convention
☐	Right	CG	[0 0 0]	deg	World	Euler X-Y-Z
✓	Left	CS1	[0 0 0]	deg	World	Euler X-Y-Z
✓	Right	CS2	[0 0 0]	deg	World	Euler X-Y-Z
☐	Right	CS3	[0 0 0]	deg	World	Euler X-Y-Z
☐	Right	CS4	[0 0 0]	deg	World	Euler X-Y-Z
☐	Right	CS5	[0 0 0]	deg	World	Euler X-Y-Z
☐	Right	CS6	[0 0 0]	deg	World	Euler X-Y-Z

Y-Revolute: Disk to Column

The dialog box for the Y-revolute joint connecting the ***disk*** to the ***column*** is shown in the panel to the right. The "Axes" tab indicates the joint is an "R1" or ***single degree-of-freedom revolute joint*** about the direction "[0 1 0]" (or *y*-axis) of the "Base" coordinate system. As connected in the top layer (and as indicated in this dialog box) the arm coordinate system ***CS2*** is the reference coordinate system for this joint. This ensures the axis of the revolute joint will always be directed along the horizontal arm. ***Two sensor/actuator ports*** are shown on the block.


Block Parameters: Y-Revolute Arm to Disk

Revolute

Represents one rotational degree of freedom. The follower (F) Body rotates relative to the base (B) Body about a single rotational axis going through collocated Body coordinate system origins. Sensor and actuator ports can be added. Base-follower sequence and axis direction determine sign of forward motion by the right-hand rule.

Connection parameters

Current base: CS2@Arm, M

Current follower: CS1@Disk, D

Number of sensor / actuator ports: 2

Parameters

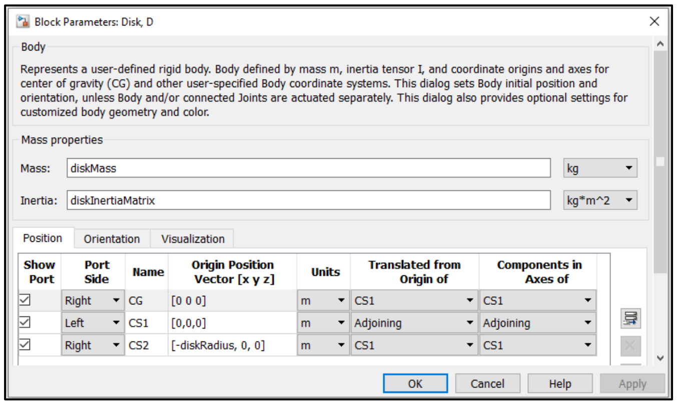
Axes | Advanced

Name	Primitive	Axis of Action [x y z]	Reference CS
R1	revolute	[0 1 0]	Base

Body block: Disk, *D*

The dialog box for the body block associated with the disk is shown below. As shown, values of the ***mass*** and the 3×3 ***inertia matrix*** of the disk are given by the variables "diskMass" and "diskInertiaMatrix". The values for these variables are set in the MATLAB script. Disk coordinate system ***CS1*** is collocated with the adjoining coordinate system ***CS2*** of the ***arm***. All other arm coordinate systems are located ***relative to***

CS1. The disk mass center coordinate system ***CG*** is collocated with ***CS1*** at the center of the disk, and coordinate system ***CS2*** is located ***on the edge of the disk***. The ***orientation angles*** of all the disk coordinate systems are ***zero***, so all the disk coordinate systems are ***initially aligned*** with the ***world system***. The "Body geometry" on the "Visualization" tab (not shown here) is the ***same*** as for the column and arm, but the ***color*** has been changed to ***pink*** so it is distinguishable from the arm. The ***equivalent ellipsoidal shape*** associated with the inertia matrix specified in the script for the disk is a circle.


Block Parameters: Disk, D

Body

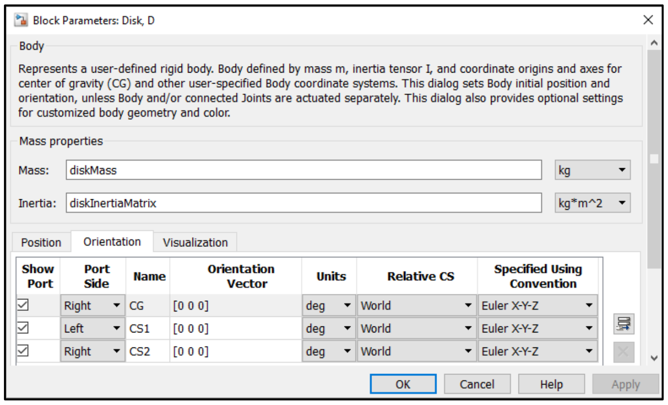
Represents a user-defined rigid body. Body defined by mass m, inertia tensor I, and coordinate origins and axes for center of gravity (CG) and other user-specified Body coordinate systems. This dialog sets Body initial position and orientation, unless Body and/or connected Joints are actuated separately. This dialog also provides optional settings for customized body geometry and color.

Mass properties

Mass:	diskMass	kg
Inertia:	diskInertiaMatrix	kg*m^2

Position | Orientation | Visualization

Show Port	Port Side	Name	Origin Position Vector [x y z]	Units	Translated from Origin of	Components in Axes of
✓	Right	CG	[0 0 0]	m	CS1	CS1
✓	Left	CS1	[0,0,0]	m	Adjoining	Adjoining
✓	Right	CS2	[-diskRadius, 0, 0]	m	CS1	CS1


Block Parameters: Disk, D

Body

Represents a user-defined rigid body. Body defined by mass m, inertia tensor I, and coordinate origins and axes for center of gravity (CG) and other user-specified Body coordinate systems. This dialog sets Body initial position and orientation, unless Body and/or connected Joints are actuated separately. This dialog also provides optional settings for customized body geometry and color.

Mass properties

Mass:	diskMass	kg
Inertia:	diskInertiaMatrix	kg*m^2

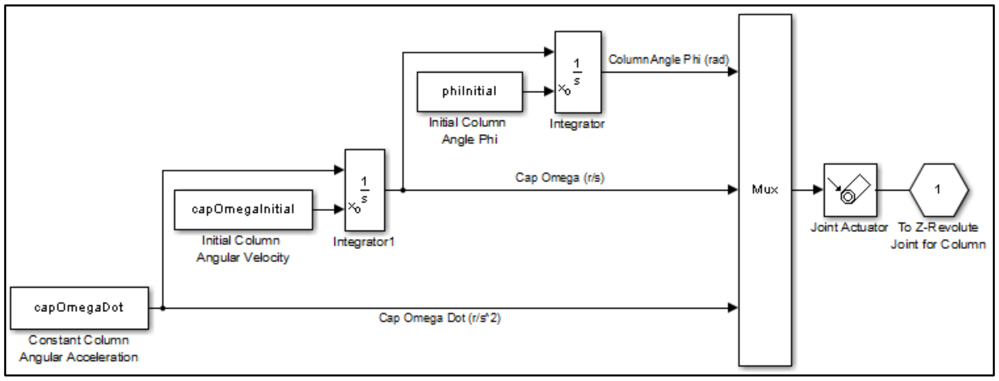
Position | Orientation | Visualization

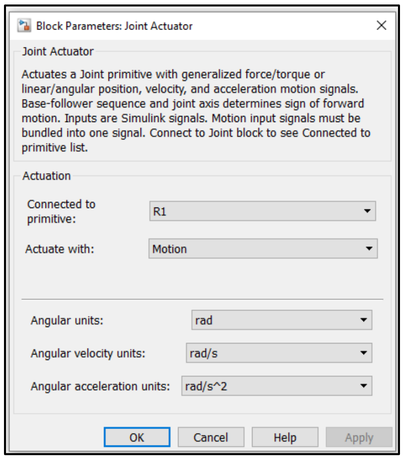
Show Port	Port Side	Name	Orientation Vector	Units	Relative CS	Specified Using Convention
✓	Right	CG	[0 0 0]	deg	World	Euler X-Y-Z
✓	Left	CS1	[0 0 0]	deg	World	Euler X-Y-Z
✓	Right	CS2	[0 0 0]	deg	World	Euler X-Y-Z

<u>Motion Subsystem</u>: Column relative to Ground

The subsystem that ***specifies*** the ***motion*** of the ***column*** relative to the ground is shown below. This subsystem is very similar to the subsystem presented in the Simulink model. The only ***difference*** is that the combined motion signals are sent to SimMechanics using the "Joint Actuator" block of the Z-revolute joint that connects the column to the ground. As in the previous example, the specified motion is ***constant angular acceleration*** "capOmegaDot" with ***initial angular velocity*** "capOmegaInitial" and ***initial angle*** "phiInitial". Values of the three variables are specified in the MATLAB script.

The dialog box for the ***joint actuator block*** is also shown below. It indicates that motion is specified for the attached single degree-of-freedom revolute joint and that the angle, angular velocity, and angular acceleration signals will be provided in radians, rad/sec, and rad/sec^2. Note here that the actuator block moves signals from Simulink to SimMechanics.




Block Parameters: Joint Actuator

Joint Actuator

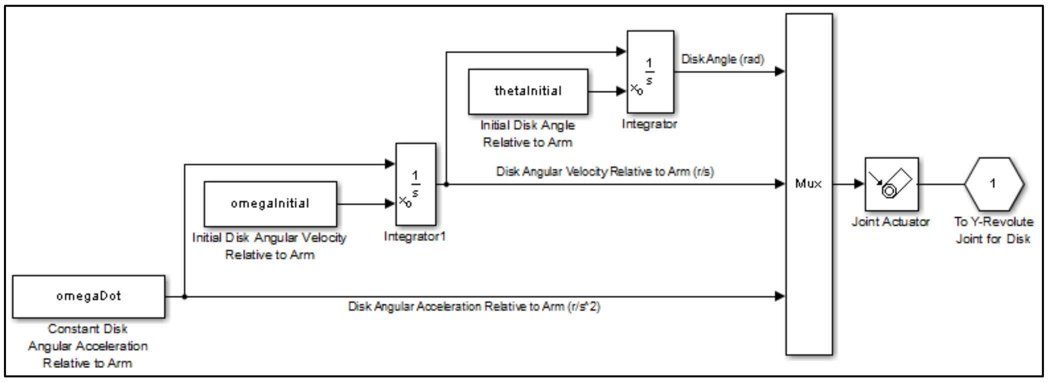
Actuates a Joint primitive with generalized force/torque or linear/angular position, velocity, and acceleration motion signals. Base-follower sequence and joint axis determines sign of forward motion. Inputs are Simulink signals. Motion input signals must be bundled into one signal. Connect to Joint block to see Connected to primitive list.

Actuation

Connected to primitive:	R1
Actuate with:	Motion
Angular units:	rad
Angular velocity units:	rad/s
Angular acceleration units:	rad/s^2

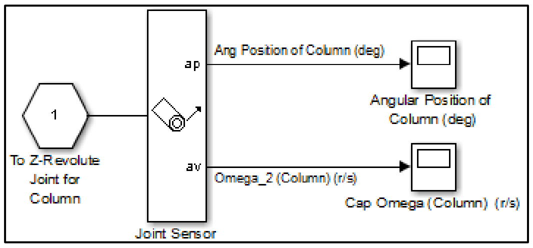
Motion Subsystem: Disk relative to Arm

This subsystem is also very similar to the subsystem presented in the Simulink model. In this case, the only ***difference*** is that the combined motion signals are sent to SimMechanics using the "Joint Actuator" block of the Y-revolute joint that connects the disk to the arm. This ***specifies*** the motion at this joint. As in the previous example, the specified motion is ***constant angular acceleration*** "omegaDot" with ***initial angular velocity*** "omegaInitial" and ***initial angle*** "thetaInitial". Values of the three variables are specified in the MATLAB script. The joint actuator block in this subsystem functions the same as the one described above for the column motion.

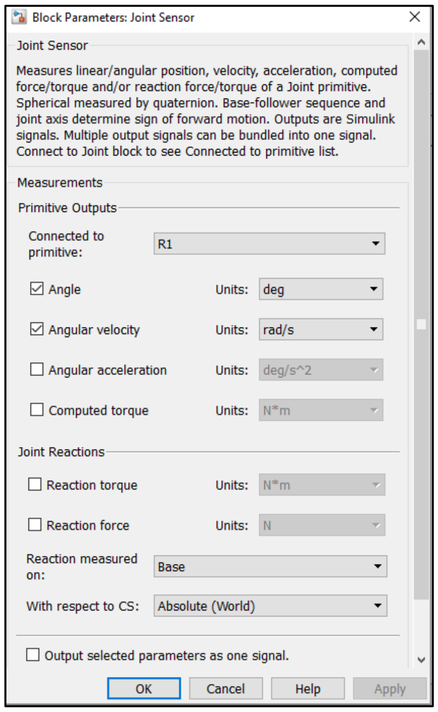

thetaInitial
Initial Disk Angle Relative to Arm
1/s
Integrator
DiskAngle (rad)
omegaInitial
Initial Disk Angular Velocity Relative to Arm
Integrator1
Disk Angular Velocity Relative to Arm (r/s)
Mux
Joint Actuator
1
To Y-Revolute Joint for Disk
omegaDot
Constant Disk Angular Acceleration Relative to Arm
Disk Angular Acceleration Relative to Arm (r/s^2)

Joint Measurement Subsystem: Column to Ground

The subsystem that measures the ***motion*** of the ***column relative to the ground*** is shown in the panel to the right. The joint sensor is connected directly to the ***Z-revolute joint*** that connects the column to the ground. The sensor sends the signal from SimMechanics to Simulink. In this case, the angular position (ap) and angular velocity (av) are measured and plotted on two scopes.


1
To Z-Revolute Joint for Column
ap
av
Joint Sensor
Ang Position of Column (deg)
Angular Position of Column (deg)
Omega_2 (Column) (r/s)
Cap Omega (Column) (r/s)

The ***dialog box*** for the ***joint sensor*** (shown in the panel to the right) allows the user to determine what is being measured. The dialog box indicates the sensor is connected to a ***single degree-of-freedom revolute joint*** and that "Primitive Outputs" and "Joint Reactions" can be measured. In this case, the ***angle*** is measured in degrees and the ***angular velocity*** is measured in rad/sec. Measurements of forces and torques associated with the joint will be discussed in Volume II of this text.


Block Parameters: Joint Sensor

Joint Sensor

Measures linear/angular position, velocity, acceleration, computed force/torque and/or reaction force/torque of a Joint primitive. Spherical measured by quaternion. Base-follower sequence and joint axis determine sign of forward motion. Outputs are Simulink signals. Multiple output signals can be bundled into one signal. Connect to Joint block to see Connected to primitive list.

Measurements

Primitive Outputs

Connected to primitive: R1

- [x] Angle — Units: deg
- [x] Angular velocity — Units: rad/s
- [ ] Angular acceleration — Units: deg/s^2
- [ ] Computed torque — Units: N*m

Joint Reactions

- [ ] Reaction torque — Units: N*m
- [ ] Reaction force — Units: N

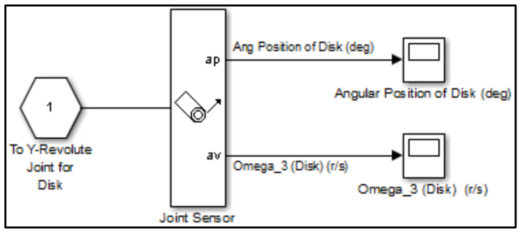
Reaction measured on: Base

With respect to CS: Absolute (World)

- [ ] Output selected parameters as one signal.

Joint Measurement Subsystem: Disk to Arm

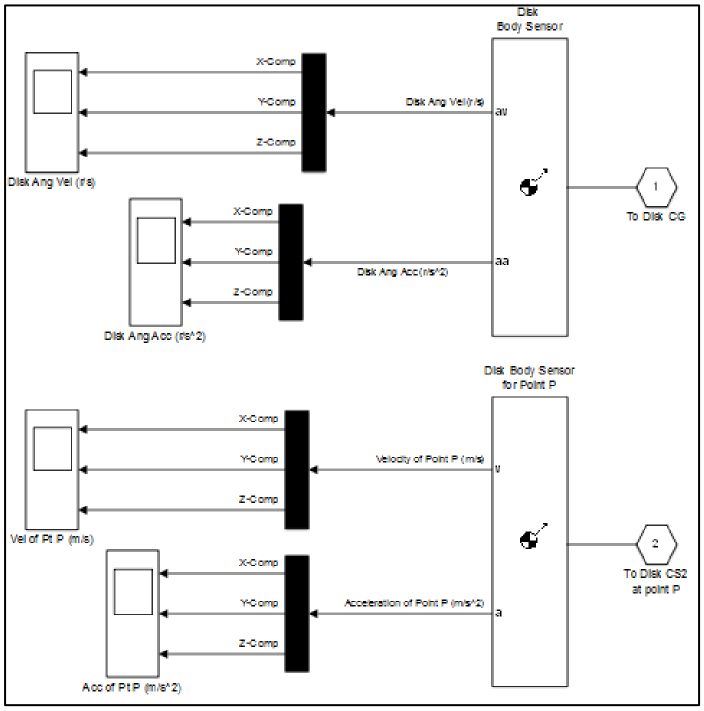
The subsystem that measures the ***motion*** of the ***disk relative to the arm*** is shown in the panel to the right. In this case, the sensor is ***connected*** to the ***Y-revolute joint*** that connects the disk and arm, and it ***measures*** the relative angular position (ap) and angular velocity (av). The settings in the joint sensor dialog box are the same as those in the joint sensor for the Z-revolute joint discussed above.



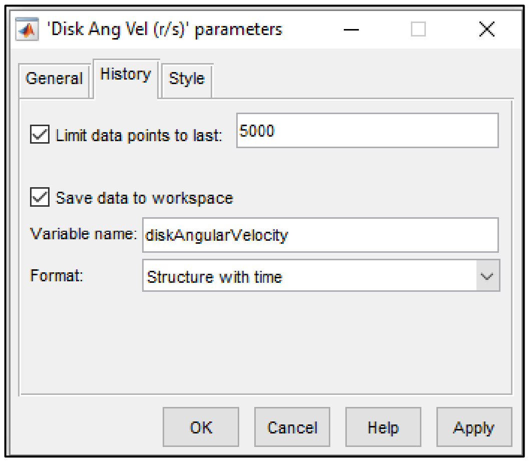
1
To Y-Revolute Joint for Disk
ap
av
Joint Sensor
Ang Position of Disk (deg)
Angular Position of Disk (deg)
Omega_3 (Disk) (r/s)
Omega_3 (Disk) (r/s)

Body Measurement Subsystem: Disk

The subsystem that ***measures*** the ***angular velocity*** (av) and ***angular acceleration*** (aa) of the ***disk*** and the ***velocity*** (v) and ***acceleration*** (a) of point ***P*** on the disk is shown in the panel to the right. Each of these ***vector quantities*** are ***separated*** (using a Demux (demultiplexor)) into ***components*** that are plotted on separate axes on a single scope. The sensor that measures the angular velocity and angular acceleration of the disk are connected to ***CG*** the coordinate system at the center of gravity, and the sensor that measures the velocity and acceleration of *P* is connected to coordinate system ***CS2*** on the edge of the disk. ***CS2*** tracks the angle theta of the disk starting at $\theta = 0$.



Each of the ***four scopes*** are used to plot the data as well as send the data to the ***MATLAB workspace*** for later plotting by the MATLAB script. The dialog box for the "Parameters" menu of the scope for the angular velocity vector is shown in the panel to the right. The "History" tab shows that the ***data will be saved*** to the workspace as a "Structure with time" using the ***variable name*** "diskAngularVelocity". The data will be limited to the last 5000 data points of the signal. The other three scopes create structures named "diskAngularAcceleration", "velocityPointP", and "accelerationPointP".



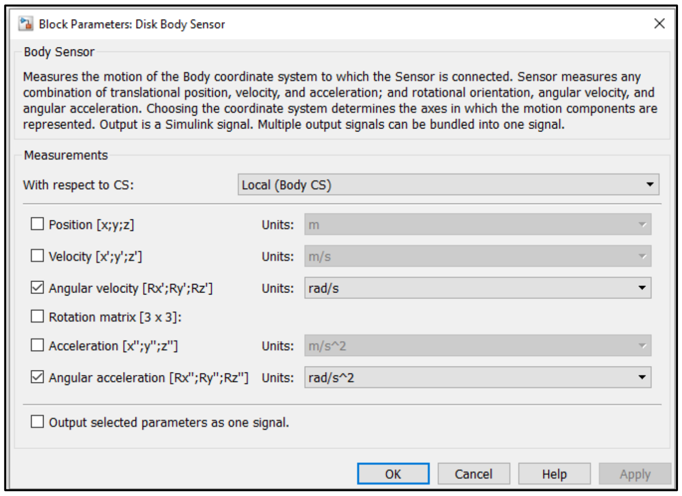
☑ Limit data points to last: 5000

☑ Save data to workspace

Variable name: diskAngularVelocity

Format: Structure with time

The dialog boxes for the ***two body sensors*** are shown in the panels below. The panel on the ***left*** indicates that the angular velocity vector in rad/sec and the angular acceleration vector in rad/sec^2 will be measured and the components will be expressed in the ***local body coordinate system***. These are the ***body-fixed components*** of the vectors. The panel on the right indicates that the body-fixed components of the velocity vector in meters/sec and acceleration vector in meters/sec^2 of point P (located by the coordinate system ***CS2***) will also be measured.



Block Parameters: Disk Body Sensor

Body Sensor

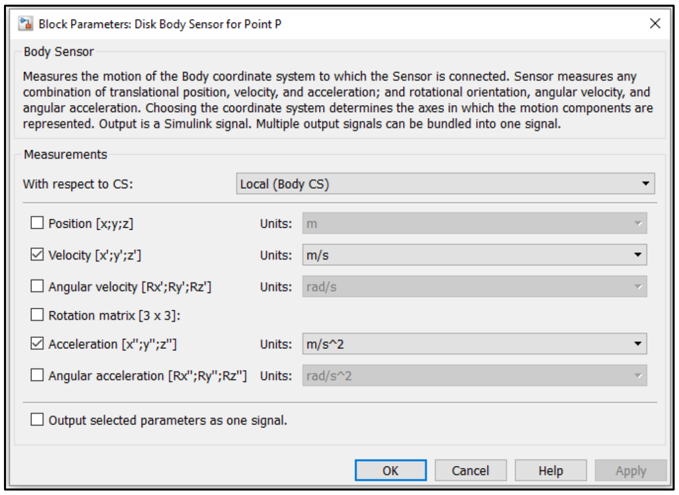
Measures the motion of the Body coordinate system to which the Sensor is connected. Sensor measures any combination of translational position, velocity, and acceleration; and rotational orientation, angular velocity, and angular acceleration. Choosing the coordinate system determines the axes in which the motion components are represented. Output is a Simulink signal. Multiple output signals can be bundled into one signal.

Measurements

With respect to CS: Local (Body CS)

☐ Position [x;y;z]	Units:	m
☐ Velocity [x';y';z']	Units:	m/s
☑ Angular velocity [Rx';Ry';Rz']	Units:	rad/s
☐ Rotation matrix [3 x 3]:		
☐ Acceleration [x'';y'';z'']	Units:	m/s^2
☑ Angular acceleration [Rx'';Ry'';Rz'']	Units:	rad/s^2

☐ Output selected parameters as one signal.



Block Parameters: Disk Body Sensor for Point P

Body Sensor

Measures the motion of the Body coordinate system to which the Sensor is connected. Sensor measures any combination of translational position, velocity, and acceleration; and rotational orientation, angular velocity, and angular acceleration. Choosing the coordinate system determines the axes in which the motion components are represented. Output is a Simulink signal. Multiple output signals can be bundled into one signal.

Measurements

With respect to CS: Local (Body CS)

☐ Position [x;y;z]	Units:	m
☑ Velocity [x';y';z']	Units:	m/s
☐ Angular velocity [Rx';Ry';Rz']	Units:	rad/s
☐ Rotation matrix [3 x 3]:		
☑ Acceleration [x'';y'';z'']	Units:	m/s^2
☐ Angular acceleration [Rx'';Ry'';Rz'']	Units:	rad/s^2

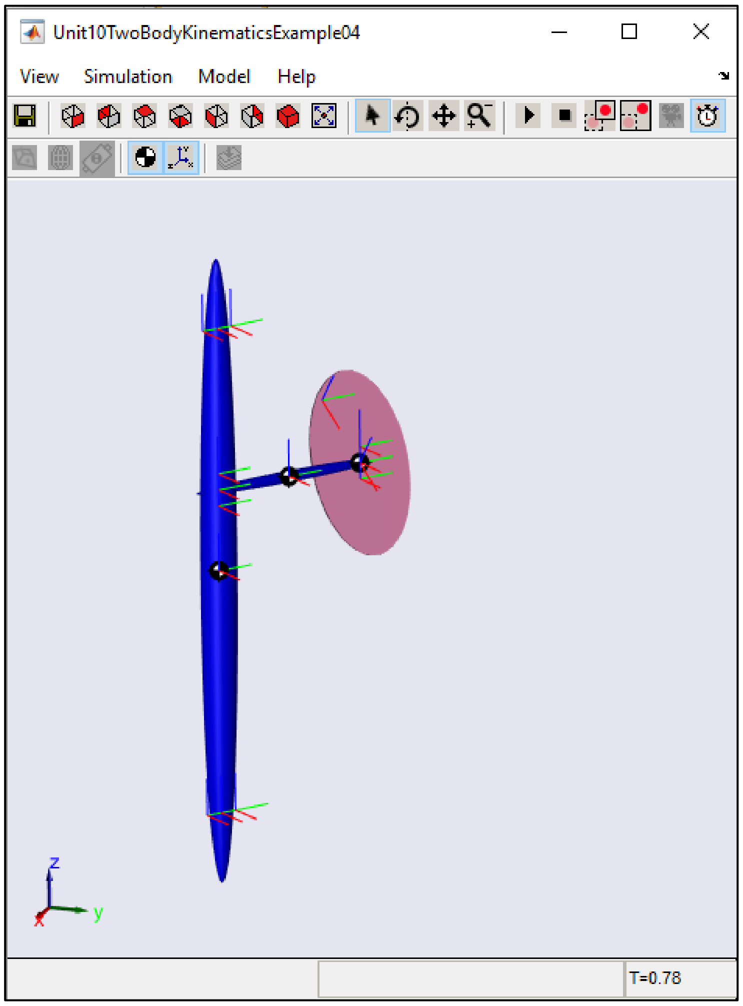
☐ Output selected parameters as one signal.

<ins>Model Execution</ins>:

The model is ***executed*** by running the MATLAB script. As described earlier, the script ***defines*** all necessary ***variables***, ***runs*** the SimMechanics model, ***plots*** the results in four figure windows, and ***displays*** the initial values of results in the command window. As the SimMechanics model executes, an ***animation window*** (see description below) opens to display the motion of the system. This model produces identical results to those of the previous models (presented in Figures 1-4 above).

Animation window:

The ***animation window*** shown in the panel to the right shows the ***position*** of the system at time $t = 0.78$ (sec). The window shows the ***world*** (global) ***coordinate system*** (lower left corner of the window), and it shows the ***mass center*** and all the ***coordinate systems*** associated with ***each body***. All X-axes are ***red***, all Y-axes are ***green***, and all Z-axes are ***blue***. At time $t = 0$ all the coordinate systems were ***aligned*** with the world system. As time progresses, however, all coordinate systems must ***maintain their orientation relative to the body*** on which they are defined. Note the column and arm coordinate systems are ***rotated*** relative to the world system, but they all remain aligned (with each other) as they move as a single body. The disk coordinate systems are ***rotated*** relative to the arm due to the disk rotation.



Command Window Output:

The command window output of the vector component values at $t = 0$ are shown in the panel below. Again, these results are ***identical*** to those generated by the previous models.

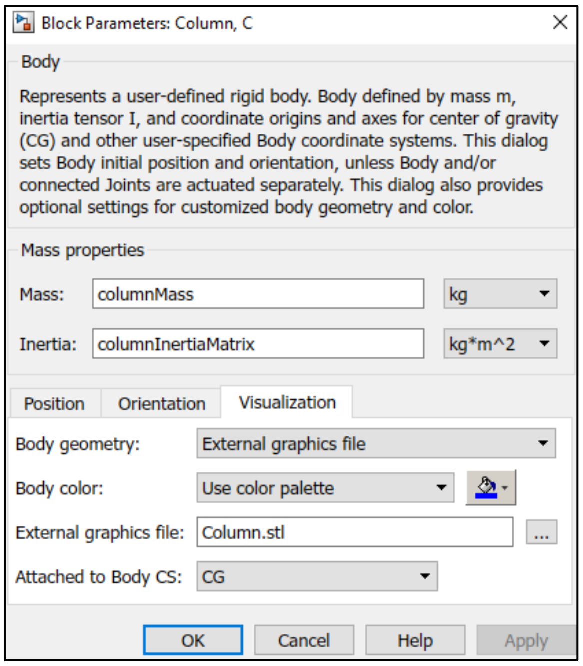
```
Body-Fixed Components of the Initial Angular Velocity of Disk D (r/s)
X-Component      Y-Component      Z-Component
     0                0                0

Body-Fixed Components of the Initial Angular Acceleration of Disk D (r/s^2)
X-Component      Y-Component      Z-Component
     0                2                3

Body-Fixed Components of the Initial Velocity of Point P (m/s)
X-Component      Y-Component      Z-Component
     0                0                0

Body-Fixed Components of the Initial Acceleration of Point P (m/s^2)
    X-Component           Y-Component           Z-Component
-1.500000000000000     -0.750000000000000    0.500000000000000
```

Increasing the Quality of the Animation:

As mentioned above, the body geometries of the column, arm, and disk (as shown above) were determined using "Equivalent ellipsoids from the mass properties" of the bodies as indicated on the "Visualization" tab of the body block dialog boxes. The ***quality*** of the animation can be easily improved using ***external stereolithographic files*** to define the geometries of the bodies. There are many applications that can be used to generate these files. Windows 10, for example, has a free application called 3D Builder that is quite easy to use.

The graphics files are indicated in the body block dialog box by first indicating the "Body geometry" is found in an "External graphics file" and then indicating the ***name*** of the file and the ***body coordinate system*** to which the geometry is attached. In this example, the geometry of the column is contained in a file named "Column.stl" located in the default directory and is attached to the ***center of gravity*** of the column. The geometries of the arm and disk are defined in two other files. The dialog boxes of the body blocks of the arm and disk are not shown here.

The resulting animation window shown here is a snapshot of the position of the system at $t = 0.85\text{ (sec)}$. The arm and column are now clearly indicated as having ***circular cross sections*** with the specified diameters, and the disk is shown with its specified diameter and thickness. For visualization purposes, a supporting cylindrical structure (shown at the base of the column) is included in the graphics file as part of the geometry of the column.


Block Parameters: Column, C

Body

Represents a user-defined rigid body. Body defined by mass m, inertia tensor I, and coordinate origins and axes for center of gravity (CG) and other user-specified Body coordinate systems. This dialog sets Body initial position and orientation, unless Body and/or connected Joints are actuated separately. This dialog also provides optional settings for customized body geometry and color.

Mass properties

Mass: columnMass kg

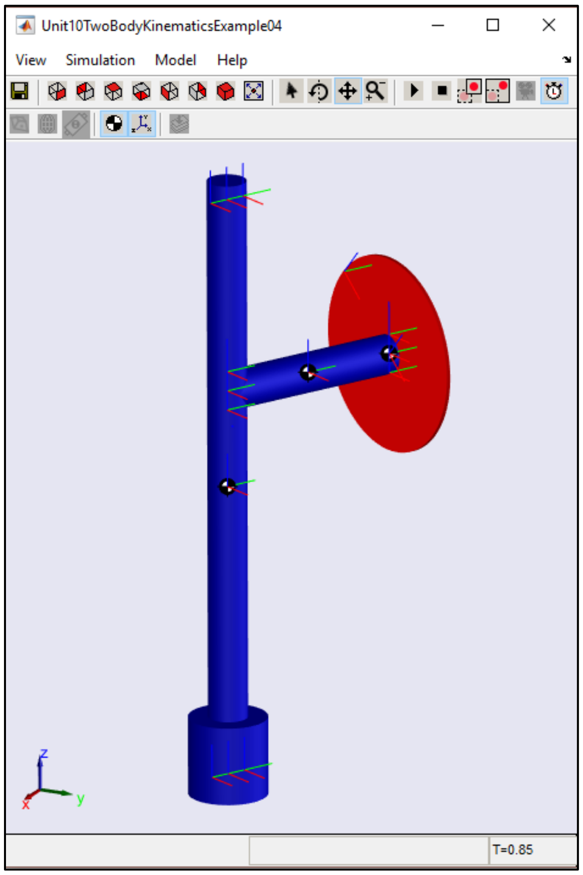
Inertia: columnInertiaMatrix kg*m^2

Body geometry: External graphics file

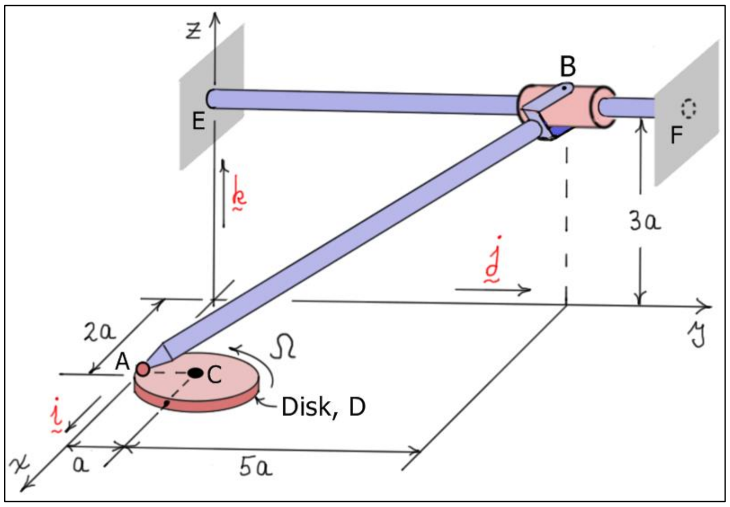
Body color: Use color palette

External graphics file: Column.stl

Attached to Body CS: CG



Example 5: SimMechanics Modeling of a Closed Loop Mechanism



At the instant shown $(\phi = 0)$, *A* has coordinates $(2a,\ 0,\ 0)$ and *B* has coordinates $(0,\ 6a,\ 3a)$.

The figure shows the three-dimensional slider-crank mechanism that was analyzed in Unit 8. Disk *D* rotates about its center *C* with angular velocity ${}^{R}\underset{\sim}{\omega}_{D} = \dot{\phi}\,\underset{\sim}{k} = \Omega\,\underset{\sim}{k}$. The center of the disk is in the *xy* plane at the point $(2a,\ a,\ 0)$. Bar *AB* is attached to the disk with a ball and socket joint at *A* and is attached to the fixed bar *EF* with a ***three degree of freedom joint*** at *B*. The collar can ***translate along*** and ***rotate about*** the fixed bar *EF* (in the *y* direction), and bar *AB* can ***rotate*** relative to the collar about an axis which is ***normal to the plane*** *AEB*. The ***size*** of the collar at *B* is assumed to be ***negligible***.

As in Unit 8, the ***motion*** of ***disk D*** is ***specified***, and the ***motion*** of ***bar AB*** and the ***collar*** at *B* is ***calculated***. In this case, a ***MATLAB script*** is used to ***execute*** a SimMechanics model that ***computes*** the ***motions*** of the ***bar*** and the ***collar***, and then it uses the ***analytical equations*** from Unit 8 to compute analytical results. The results of the two methods are shown to be ***identical***.

The ***first four sections*** of the MATLAB script are shown in Panel 1 below. The ***first section*** provides a ***functional description*** of the script and starts the "pause" option, the ***second defines variables*** used by the SimMechanics model, the ***third executes*** the SimMechanics model, and the ***fourth plots*** the ***specified motions*** applied to the disk. In this example, disk *D* has a ***constant angular acceleration*** given by the value of variable "`crankInitialAngularAcceleration`", an ***initial angular velocity*** given by the variable "`crankInitialAngularVelocity`", and an ***initial angle*** of ***zero***. As shown in the diagram above, the ***initial geometry*** of the system is defined in terms of the parameter "a". The ***radius*** of *D* ("`diskRadius`") is "a", and the ***length*** of the bar ("`barLength`") is "7a".

The ***last section*** shown in this panel ***plots*** the angular motion of the crank (i.e. the disk) in Figures 1-3. The angle, angular velocity, and angular acceleration of the disk are assumed to be stored in structures with the associated time values. The names of the structures are given in the plot commands.

Script – Panel 1: (functional description, initialization of variables, model execution, crank motion plots)

```
%% Unit 10 - Example 5 - Three-dimensional slider crank model
%
%  This script sets values for the SimMechanics model of Unit 10, Example
%  5, a three dimensional slider crank mechanism with applied motion,
%  executes the SimMechanics model, and plots the results.
%
%   This script computes results using both a SimMechanics model and
%   an analytical model so the results can be compared.
%
    pause on;
%
%%  model input

    crankInitialAngularVelocity        = 10; % (rad/s)
    crankInitialAngularAcceleration    = 5;  % (rad/s^2)

    a          = 0.1;  % (meters)
    diskRadius = a;    % (meters)
    barLength  = 7*a;  % (meters)

%%  run the SimMechanics model

    sim('Unit10ThreeDimensionalSliderCrankExample05');

%%  plot the crank motion

    figure(1); clf;
    plot(crankAngle.time,crankAngle.signals.values);
    grid; xlabel('Time (sec)'); ylabel('Angle (deg)');
    title('Angle of Crank');

    figure(2); clf;
    plot(crankAngularVelocity.time,crankAngularVelocity.signals.values);
    grid; xlabel('Time (sec)'); ylabel('Angular Velocity (rad/s)');
    title('Angular Velocity of Crank');

    figure(3); clf;
    plot(crankAngularAcceleration.time,crankAngularAcceleration.signals.values);
    grid; xlabel('Time (sec)'); ylabel('Angular Acceleration (rad/s^2)');
    title('Angular Acceleration of Crank');
                                    ⋮
```

The ***next two sections*** of the script are shown in ***Panel 2*** below. The ***first*** section ***plots*** the components of the ***angular velocity*** and ***angular acceleration*** of the ***bar*** resolved in the world coordinate system in Figures 4-5. After each plot, MATLAB ***pauses*** to allow the user to ***edit*** the plot (for example, to highlight data values using the data cursor). After pausing, the script will ***continue execution*** when the user ***presses any key*** on the keyboard. The ***second*** section ***plots*** the ***velocity*** and ***acceleration*** of the slider in Figures 6-7. Again, MATLAB ***pauses*** after each plot to allow the user to ***edit*** the plot. The data is assumed to be stored in structures with the associated time values; the names of the structures are given in the plot commands.

Script – Panel 2: (plot bar and slider motion and pause to allow the user to edit the plots)

```
%%  plot the bar motion
    figure(4); clf;
    subplot(3,1,1)
    plot(barAngularVelocity.time,barAngularVelocity.signals(1).values)
    title('Angular Velocity of the Bar - World Components');
    ylabel('X Component (r/s)'); grid on;
    subplot(3,1,2)
    plot(barAngularVelocity.time,barAngularVelocity.signals(2).values)
    ylabel('Y Component (r/s)'); grid on;
    subplot(3,1,3)
    plot(barAngularVelocity.time,barAngularVelocity.signals(3).values)
    ylabel('Z Component (r/s)'); grid on; xlabel('Time (sec)');

    pause;

    figure(5); clf;
    subplot(3,1,1)
    plot(barAngularAcceleration.time,barAngularAcceleration.signals(1).values)
    title('Angular Acceleration of the Bar - World Components');
    ylabel('X Component (r/s)'); grid on;
    subplot(3,1,2)
    plot(barAngularAcceleration.time,barAngularAcceleration.signals(2).values)
    ylabel('Y Component (r/s)'); grid on;
    subplot(3,1,3)
    plot(barAngularAcceleration.time,barAngularAcceleration.signals(3).values)
    ylabel('Z Component (r/s)'); grid on; xlabel('Time (sec)');

    pause;

%%  plot the motion of the slider
    figure(6); clf;
    plot(sliderVelocity.time,sliderVelocity.signals.values);
    grid; xlabel('Time (sec)'); ylabel('Velocity (m/s)');
    title('Velocity of Slider');

    pause;

    figure(7); clf;
    plot(sliderAcceleration.time,sliderAcceleration.signals.values);
    grid; xlabel('Time (sec)'); ylabel('Acceleration (m/s^2)');
    title('Acceleration of Slider');

    pause;
                                        ⋮
```

The ***next section*** of the script is shown in ***Panel 3*** below. This section ***calculates analytical results*** at the ***initial position*** $(t = 0)$ and ***displays*** the results in the MATLAB command window. See Unit 8 for ***derivations*** of these equations. To ***format*** the results in the command window, advantage is taken of the MATLAB function "`num2str`" which ***converts*** a ***numerical value*** into a ***string*** for use in the display command.

Script – Panel 3: (calculate and display analytical results at the initial position, $t = 0$)

```
%%  Calculate analytical results for omega_AB, v_B, alpha_AB, a_B at t=0
%   The following results are calculated for the initial position
%
    omegaX0 =  3*crankInitialAngularVelocity/39;
    omegaY0 = -9*crankInitialAngularVelocity/39;
    omegaZ0 =  2*crankInitialAngularVelocity/39;
    v_B0    = -a*crankInitialAngularVelocity/3;

   alphaCoefficientMatrix0 = [0 3 -6; 6 2 0; -2 0 3];
   alphaRHS0 = zeros(3,1);
   alphaRHS0(1,1) = -(crankInitialAngularAcceleration) - ...
                      omegaY0*((6*omegaX0)+(2*omegaY0)) - ...
                      omegaZ0*((2*omegaZ0)+(3*omegaX0));
   alphaRHS0(2,1) =  omegaX0*((2*omegaZ0)+(3*omegaX0)) + ...
                     omegaY0*((3*omegaY0)-(6*omegaZ0));
   alphaRHS0(3,1) = crankInitialAngularVelocity*omegaX0;

   alpha0 = alphaCoefficientMatrix0\alphaRHS0; % solve matrix equation for alpha

   a_B0 = -(3*a*alpha0(1,1)) - (2*a*alpha0(3,1)) + ...
           (a*crankInitialAngularVelocity*crankInitialAngularVelocity) +...
            omegaZ0*((3*a*omegaY0)-(6*a*omegaZ0)) -...
            omegaX0*((6*a*omegaX0)+(2*a*omegaY0));

   disp('   ');
   disp('Analytical Results at the Initial Position (phi = 0)');
   disp('-----------------------------------------------------');
   disp('   ');
   disp('Initial Angular Velocity Components (r/s)');
   printString = ['   X component: ' num2str(omegaX0)]; disp(printString);
   printString = ['   Y component: ' num2str(omegaY0)]; disp(printString);
   printString = ['   Z component: ' num2str(omegaZ0)]; disp(printString);
   disp('   ');
   disp('Initial Angular Acceleration Components (r/s^2)');
   printString = ['   X component: ' num2str(alpha0(1,1))]; disp(printString);
   printString = ['   Y component: ' num2str(alpha0(2,1))]; disp(printString);
   printString = ['   Z component: ' num2str(alpha0(3,1))]; disp(printString);
   disp('   ');
   disp('Initial Collar Velocity (v_B0) (m/s)');
   printString = ['   v_B0: ' num2str(v_B0)]; disp(printString);
   disp('   ');
   disp('Initial Collar Acceleration (a_B0) (m/s^2)');
   printString = ['   a_B0: ' num2str(a_B0)]; disp(printString);
                                                ⋮
```

The ***next section*** of the script is shown in ***Panel 4*** below. This section ***calculates analytical results*** at all times $t \geq 0$ and ***stores*** the results for later plotting.

Script – Panel 4: (calculate analytical results at all positions, $t \geq 0$)

```
%% Calculate analytical results for arrays omega_AB, v_B, alpha_AB, a_B (t >= 0)
%  The following results are calculated for the entire motion

   nTimes   = length(crankAngle.time);  yBHat   = zeros(nTimes,1);
   v_B      = zeros(nTimes,1);          a_B     = zeros(nTimes,1);
   omegaAB  = zeros(nTimes,3);          alphaAB = zeros(nTimes,3);

   coefficientMatrix = zeros(3,3); rightHandSideV = zeros(3,1);
   rightHandSideA    = zeros(3,1);

   coefficientMatrix(1,1) = 0.0; coefficientMatrix(1,2) = 3.0;
   coefficientMatrix(2,3) = 0.0; coefficientMatrix(3,2) = 0.0;
   coefficientMatrix(3,3) = 3.0;
   rightHandSideV(2,1)    = 0.0; rightHandSideV(3,1)    = 0.0;

   for i = 1:nTimes
      crankAngleRad = (pi/180)*crankAngle.signals.values(i);
      cosPhi = cos(crankAngleRad);    sinPhi = sin(crankAngleRad);
      crankAngularVelocityValue     = crankAngularVelocity.signals.values(i);
      crankAngularAccelerationValue = crankAngularAcceleration.signals.values(i);

      yBHat(i)               = 1 - cosPhi + sqrt(40 - (2+sinPhi)^2);
      coefficientMatrix(1,3) = 1 - cosPhi - yBHat(i);
      coefficientMatrix(2,1) = -coefficientMatrix(1,3);
      coefficientMatrix(2,2) = 2.0 + sinPhi;
      coefficientMatrix(3,1) = -coefficientMatrix(2,2);
      rightHandSideV(1,1)    = -crankAngularVelocityValue*cosPhi;

      omegaAB(i,:) = coefficientMatrix\rightHandSideV;  % solve matrix equation
      v_B(i,1)     = (a*crankAngularVelocityValue*sinPhi) -...
                     (3*a*omegaAB(i,1)) - (a*(2+sinPhi)*omegaAB(i,3));

      rightHandSideA(1,1) = ...
         - (crankAngularAccelerationValue*cosPhi) + ...
           ((crankAngularVelocityValue^2)*sinPhi) - ...
           (omegaAB(i,1)*omegaAB(i,2)*(yBHat(i) + cosPhi - 1)) - ...
           ((omegaAB(i,2)^2)*(2 + sinPhi)) - ...
           ((omegaAB(i,3)^2)*(2 + sinPhi)) - (3*omegaAB(i,1)*omegaAB(i,3));
      rightHandSideA(2,1) = ...
           (3*omegaAB(i,1)*omegaAB(i,1)) + (3*omegaAB(i,2)*omegaAB(i,2)) +...
           (omegaAB(i,1)*omegaAB(i,3)*(2 + sinPhi)) -...
           (omegaAB(i,2)*omegaAB(i,3)*(yBHat(i) + cosPhi - 1));
      rightHandSideA(3,1) = crankAngularVelocityValue*cosPhi*omegaAB(i,1);

      alphaAB(i,:) = coefficientMatrix\rightHandSideA;  % solve matrix equation
      a_B(i,1) = -(3*a*alphaAB(i,1)) - ((2 + sinPhi)*a*alphaAB(i,3)) +...
                 (a*crankAngularAccelerationValue*sinPhi) +...
                 (a*(crankAngularVelocityValue^2)*cosPhi) +...
                 (3*a*omegaAB(i,2)*omegaAB(i,3)) -...
                 (a*omegaAB(i,1)*omegaAB(i,2)*(2 + sinPhi)) -...
                 ((omegaAB(i,1)^2)+(omegaAB(i,3)^2))*a*(yBHat(i) + cosPhi - 1);
   end
                                          ⋮
```

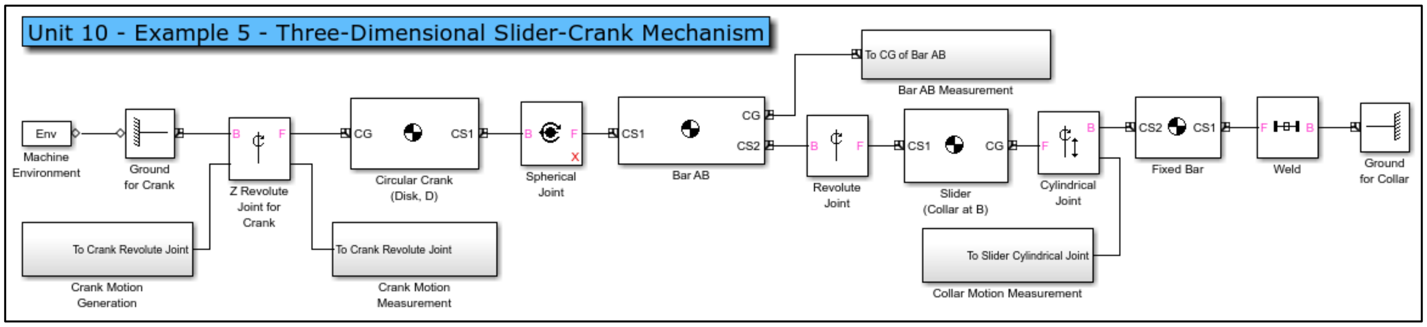
The ***final section*** of the script is shown in ***Panel 5*** below. This section ***plots*** the ***analytical results*** at all times $t \geq 0$ in Figures 8-11. The script pauses after each plot to allow the user to edit the plot. At the end of the script, the pause feature is turned off.

Script – Panel 5: (plot the analytical results for all time $t \geq 0$)

```
%%  Plot the analytical results for t >= 0

    figure(8); clf;
    subplot(3,1,1)
    plot(barAngularVelocity.time,omegaAB(:,1))
    title('Angular Velocity of the Bar - World Components - Analytical Results');
    ylabel('X Component (r/s)'); grid on;
    subplot(3,1,2)
    plot(barAngularVelocity.time,omegaAB(:,2))
    ylabel('Y Component (r/s)'); grid on;
    subplot(3,1,3)
    plot(barAngularVelocity.time,omegaAB(:,3))
    ylabel('Z Component (r/s)'); grid on; xlabel('Time (sec)');

    pause;

    figure(9); clf;
    plot(sliderVelocity.time,v_B(:,1));
    grid; xlabel('Time (sec)'); ylabel('Velocity (m/s)');
    title('Velocity of Slider - Analytical Results');

    pause;

    figure(10); clf;
    subplot(3,1,1)
    plot(barAngularAcceleration.time,alphaAB(:,1))
    title('Angular Acceleration of the Bar - World Components - Analytical Results');
    ylabel('X Component (r/s^2)'); grid on;
    subplot(3,1,2)
    plot(barAngularAcceleration.time,alphaAB(:,2))
    ylabel('Y Component (r/s^2)'); grid on;
    subplot(3,1,3)
    plot(barAngularAcceleration.time,alphaAB(:,3))
    ylabel('Z Component (r/s^2)'); grid on; xlabel('Time (sec)');

    pause;

    figure(11); clf;
    plot(sliderAcceleration.time,a_B(:,1));
    grid; xlabel('Time (sec)'); ylabel('Velocity (m/s^2)');
    title('Acceleration of Slider - Analytical Results');

    pause;

    pause off;
```

<u>SimMechanics Model</u>: Top Layer

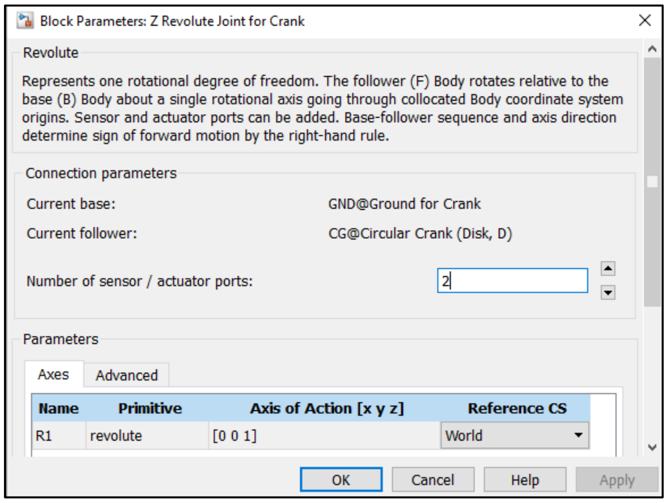
The ***top layer*** of the SimMechanics model is shown in the panel below. As with the previous model, the left side of the diagram begins with a ***Ground*** block and an attached ***Machine Environment*** block. The ***model configuration parameters*** on the ***Simulation menu*** of the model are set to calculate the system's motion for 2 seconds using a ***fixed-step***, ***8th order***, ***Dormand-Prince integration method***. The ***time step*** is set to 0.01 seconds. The ***Machine dimensionality*** is set to "Auto-detect" and the ***Analysis mode*** is set to "Forward dynamics" in the Machine Environment block. Note that an Analysis mode of "Kinematics" can also be applied to this system as it is a closed-loop mechanism. The Ground block locates the center of the circular crank (disk, D) at the world coordinates $(2a, a, 0)$ (meters).



It is evident from the block diagram that the ***circular crank*** is connected to the ***ground*** using a ***revolute*** joint and is connected to ***bar AB*** with a ***spherical*** (ball and socket) joint. The ***bar*** is connected to the ***slider*** using a ***revolute*** joint, and the ***slider*** is connected to a ***fixed bar*** using a ***cylindrical joint***. Due to the closed-loop nature of the mechanism, ***two ground blocks*** are necessary, one for grounding the crank and one for grounding the fixed bar. The diagram also shows ***four subsystems***, one to ***specify*** the ***motion*** of the crank, and the ***other three*** to ***measure*** the ***motions*** of the crank, the bar, and the collar.

<u>Joint Block</u>: *Z*-Revolute Joint for the Crank

The ***dialog box*** for the ***revolute joint*** that connects the ***crank*** to the ***ground*** is shown to the right. It indicates the ***line of action*** of the joint is the ***z-axis*** of the ***world*** coordinate system. The ***base*** of the joint is a ***world*** coordinate system whose origin is specified by the ***ground block***, and the ***follower*** coordinate system is the *CG* of the circular crank. The dialog box for the ground block (not shown here) specifies the ground block is located at the world coordinates $[2a,\ a,\ 0]$.


Block Parameters: Z Revolute Joint for Crank

Revolute

Represents one rotational degree of freedom. The follower (F) Body rotates relative to the base (B) Body about a single rotational axis going through collocated Body coordinate system origins. Sensor and actuator ports can be added. Base-follower sequence and axis direction determine sign of forward motion by the right-hand rule.

Connection parameters

Current base:	GND@Ground for Crank
Current follower:	CG@Circular Crank (Disk, D)
Number of sensor / actuator ports:	2

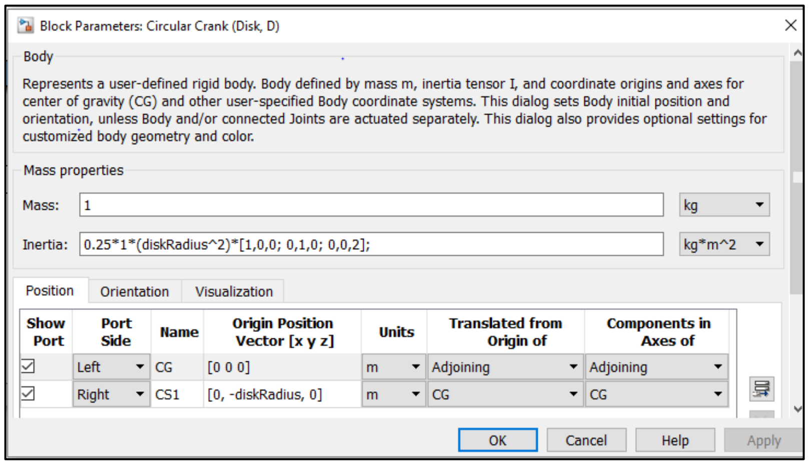
Parameters

Axes | Advanced

Name	Primitive	Axis of Action [x y z]	Reference CS
R1	revolute	[0 0 1]	World

Body block: Circular Crank (Disk, *D*)

The ***dialog box*** for the ***circular crank*** indicates that its ***mass*** is 1 (kg) and its ***inertia matrix*** is consistent with that of a ***thin disk***. It also shows that the crank has ***two coordinate systems***. The origin of the ***CG*** system is ***collocated*** with the ***adjoining*** ground block at $[2a, a, 0]$, and the origin of ***CS1*** is located on the ***left edge*** of the disk where bar *AB* is connected.



Block Parameters: Circular Crank (Disk, D)

Body

Represents a user-defined rigid body. Body defined by mass m, inertia tensor I, and coordinate origins and axes for center of gravity (CG) and other user-specified Body coordinate systems. This dialog sets Body initial position and orientation, unless Body and/or connected Joints are actuated separately. This dialog also provides optional settings for customized body geometry and color.

Mass properties

Mass:	1	kg
Inertia:	0.25*1*(diskRadius^2)*[1,0,0; 0,1,0; 0,0,2];	kg*m^2

Position | Orientation | Visualization

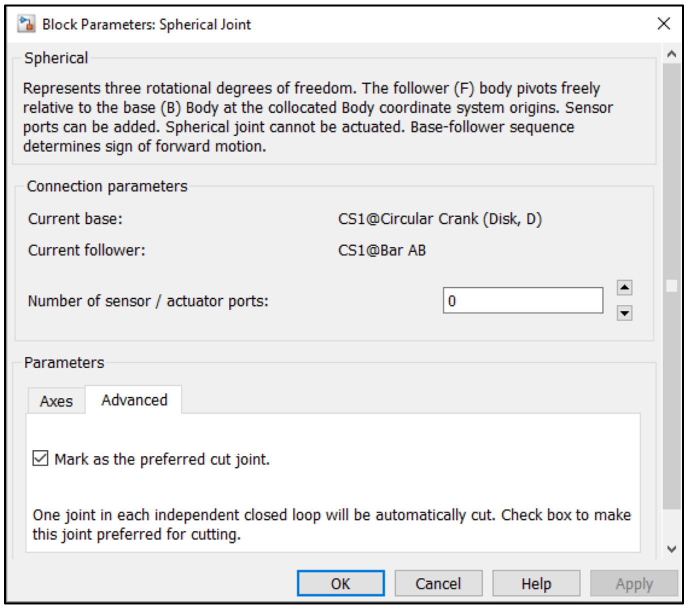
Show Port	Port Side	Name	Origin Position Vector [x y z]	Units	Translated from Origin of	Components in Axes of
✓	Left	CG	[0 0 0]	m	Adjoining	Adjoining
✓	Right	CS1	[0, -diskRadius, 0]	m	CG	CG

The ***orientation angles*** for the two coordinate systems are all set to ***zero*** on the Orientation tab so that these two systems are ***initially aligned*** (at $t = 0$) with the ***world coordinate system***. The body ***color*** is set to ***blue***, and the body ***geometry*** is set to “Equivalent ellipsoid from mass properties”. Hence, the body will appear as a ***blue disk*** in the ***animation*** window.

Spherical Joint: Connecting Crank to Bar *AB*

The ***dialog box*** for the ***spherical joint*** indicates the ***base*** coordinate system of the joint to be ***CS1*** of the ***crank***, and the ***follower*** coordinate system to be ***CS1*** of the ***bar***. As this joint allows rotation about ***any axis***, the “Axes” tab (not shown here) indicates that no “axes of action” or “reference coordinate systems” are associated with this joint. In this example, the spherical joint has been specified to be a “cut joint” on the “Advanced” tab. This is indicated on the top-layer block diagram as an “X” on the joint block.

Cut joints are used by SimMechanics when analyzing systems with ***closed kinematic chains***. SimMechanics ***cuts*** (or opens) one of the joints in each of the closed loops of the system and then formulates a ***constraint*** that requires the joint to remain ***unbroken*** during the motion. The slider-crank mechanism forms a single closed loop, so SimMechanics will cut ***one*** of the joints. The “Advanced” tab allows the user to specify a joint to be the “preferred” joint to cut during the analysis. If the user does ***not specify*** which joint to cut, SimMechanics will decide.



Block Parameters: Spherical Joint

Spherical

Represents three rotational degrees of freedom. The follower (F) body pivots freely relative to the base (B) Body at the collocated Body coordinate system origins. Sensor ports can be added. Spherical joint cannot be actuated. Base-follower sequence determines sign of forward motion.

Connection parameters

Current base:	CS1@Circular Crank (Disk, D)
Current follower:	CS1@Bar AB
Number of sensor / actuator ports:	0

Parameters

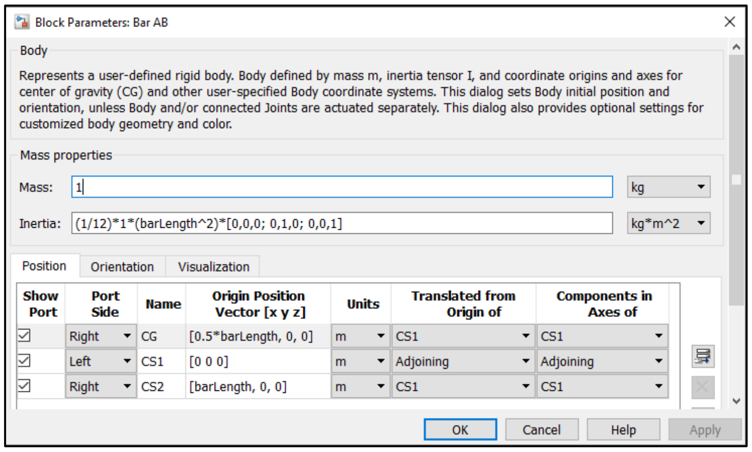
Axes | Advanced

✓ Mark as the preferred cut joint.

One joint in each independent closed loop will be automatically cut. Check box to make this joint preferred for cutting.

<u>Body block</u>: Bar *AB*

The ***dialog box*** for the ***body block*** of bar *AB* indicates that the ***mass*** of the bar is 1 (kg) and its ***inertia matrix*** is consistent with that of a ***slender bar***. The "Position" tab shows there are ***three coordinate systems*** associated with the bar. The origin of ***CS1*** is ***collocated*** with the ***adjoining coordinate system*** which is ***CS1*** of the ***circular crank***. The origin of ***CG*** is at the ***midpoint*** of the bar, and the origin of ***CS2*** is at the ***other end*** of the bar.



Block Parameters: Bar AB

Body

Represents a user-defined rigid body. Body defined by mass m, inertia tensor I, and coordinate origins and axes for center of gravity (CG) and other user-specified Body coordinate systems. This dialog sets Body initial position and orientation, unless Body and/or connected Joints are actuated separately. This dialog also provides optional settings for customized body geometry and color.

Mass properties

Mass: 1 kg

Inertia: (1/12)*1*(barLength^2)*[0,0,0; 0,1,0; 0,0,1] kg*m^2

Position | Orientation | Visualization

Show Port	Port Side	Name	Origin Position Vector [x y z]	Units	Translated from Origin of	Components in Axes of
✓	Right	CG	[0.5*barLength, 0, 0]	m	CS1	CS1
✓	Left	CS1	[0 0 0]	m	Adjoining	Adjoining
✓	Right	CS2	[barLength, 0, 0]	m	CS1	CS1

The "Visualization" and "Orientation" tabs of the dialog box of the body block are shown below. The Visualization tab indicates the geometry will be based on the coordinate systems defined for the body and the color will be red. As the information on the Orientation tab for this body is more complicated than in previous examples, a detailed explanation is necessary and provided in the following paragraphs.

As defined on the "Position" tab, the ***three coordinate systems*** of the bar are ***aligned*** along the ***x-axis*** of the body and ***each coordinate system*** is ***aligned*** with the world system. In this position, one end of the bar is at the ***spherical joint*** at *A* and the other end points ***outward*** along the world's ***x-axis***. To realign the bar so that coordinate system *CS2* is positioned at the location of the collar, a set of non-zero ***orientation angles*** must be provided on the Orientation tab.

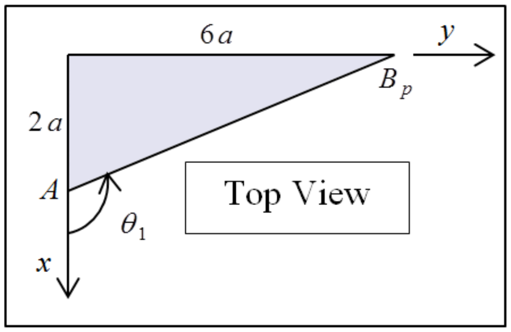
In this case, a ***3-2-1 body-fixed***, orientation angle sequence is used to define the initial orientation of the bar's coordinate systems ***relative*** to the ***world system***. The ***first rotation*** (angle θ_1) about the ***world z-axis*** rotates the bar from its position along the positive *x*-axis to a position along the ***projection*** of the bar on the *xy* plane. The ***second rotation*** (angle θ_2) about the ***bar's y-axis*** elevates the bar into its correct initial position. A third angle is not needed.

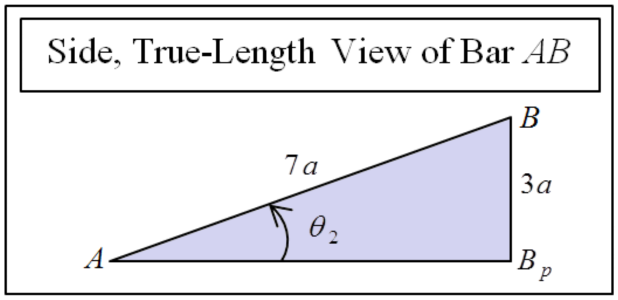
The angles θ_1 and θ_2 are illustrated in the diagrams below. The point B_p represents the ***projection*** of point *B* onto the *xy* plane. These angles may be calculated as follows

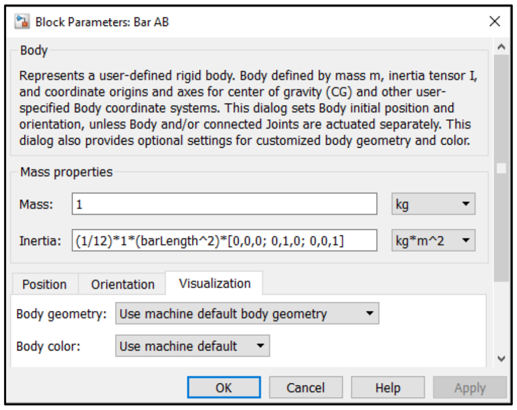
$$\boxed{\begin{aligned}\theta_1 &= 180-\left[\left(\tfrac{180}{\pi}\right)\tan^{-1}\left(\tfrac{6a}{2a}\right)\right] = 180-\left[\left(\tfrac{180}{\pi}\right)\tan^{-1}\left(\tfrac{6}{2}\right)\right]\\ &= 108.435 \text{ (deg)}\end{aligned}}$$

$$\boxed{\begin{aligned}\theta_2 &= -\left(\tfrac{180}{\pi}\right)\sin^{-1}\left(\tfrac{3a}{7a}\right) = -\left(\tfrac{180}{\pi}\right)\sin^{-1}\left(\tfrac{3}{7}\right)\\ &= -25.3769 \text{ (deg)}\end{aligned}}$$

Note that angle θ_2 must be ***negative*** to ***elevate*** the bar above the plane. See the ***formulae*** in the "Orientation Vector box" of the Orientation tab.







Body

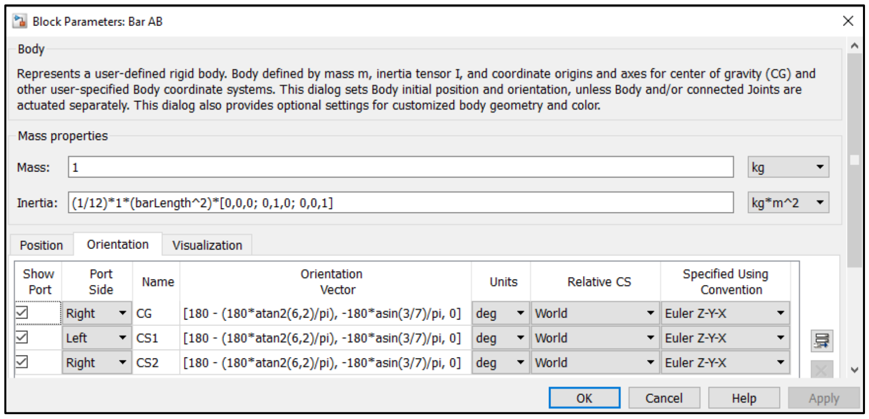
Represents a user-defined rigid body. Body defined by mass m, inertia tensor I, and coordinate origins and axes for center of gravity (CG) and other user-specified Body coordinate systems. This dialog sets Body initial position and orientation, unless Body and/or connected Joints are actuated separately. This dialog also provides optional settings for customized body geometry and color.

Mass properties

Mass:	1	kg
Inertia:	(1/12)*1*(barLength^2)*[0,0,0; 0,1,0; 0,0,1]	kg*m^2

Position | Orientation | Visualization

Body geometry:	Use machine default body geometry
Body color:	Use machine default



Body

Represents a user-defined rigid body. Body defined by mass m, inertia tensor I, and coordinate origins and axes for center of gravity (CG) and other user-specified Body coordinate systems. This dialog sets Body initial position and orientation, unless Body and/or connected Joints are actuated separately. This dialog also provides optional settings for customized body geometry and color.

Mass properties

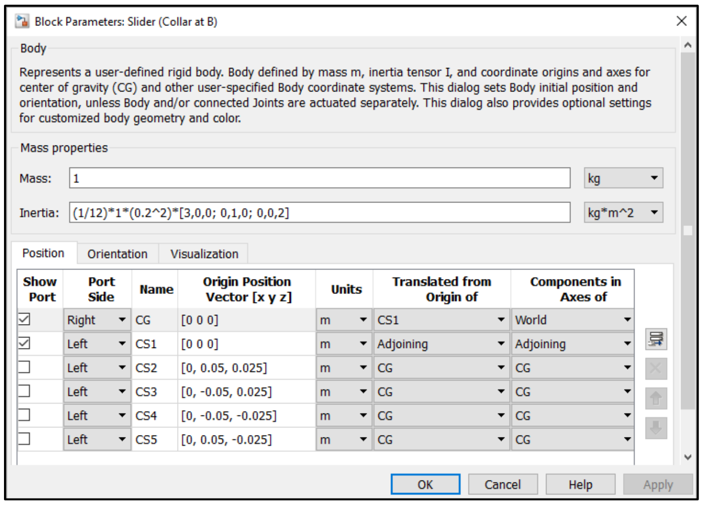
Mass:	1	kg
Inertia:	(1/12)*1*(barLength^2)*[0,0,0; 0,1,0; 0,0,1]	kg*m^2

Position | Orientation | Visualization

Show Port	Port Side	Name	Orientation Vector	Units	Relative CS	Specified Using Convention
☑	Right	CG	[180 - (180*atan2(6,2)/pi), -180*asin(3/7)/pi, 0]	deg	World	Euler Z-Y-X
☑	Left	CS1	[180 - (180*atan2(6,2)/pi), -180*asin(3/7)/pi, 0]	deg	World	Euler Z-Y-X
☑	Right	CS2	[180 - (180*atan2(6,2)/pi), -180*asin(3/7)/pi, 0]	deg	World	Euler Z-Y-X

Body block: Slider (Collar) at B

The "Position" and "Orientation" tabs of the dialog box for the body block of the ***slider*** at B are shown below. The ***mass*** of the slider is 1 (kg) and its ***inertia matrix*** is consistent with that of a ***slender bar***. Coordinate system ***CS1*** of the slider is ***collocated*** with the adjoining coordinate system ***CS2*** of bar AB, and the coordinate system ***CG*** is collocated with ***CS1***. The slider coordinate systems ***CS2-CS5*** define a simple rectangular shape.



Body

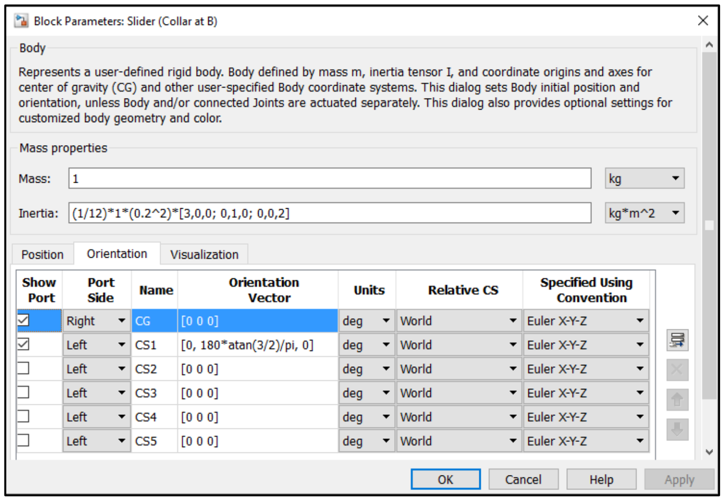
Represents a user-defined rigid body. Body defined by mass m, inertia tensor I, and coordinate origins and axes for center of gravity (CG) and other user-specified Body coordinate systems. This dialog sets Body initial position and orientation, unless Body and/or connected Joints are actuated separately. This dialog also provides optional settings for customized body geometry and color.

Mass properties

Mass:	1	kg
Inertia:	(1/12)*1*(0.2^2)*[3,0,0; 0,1,0; 0,0,2]	kg*m^2

Position | Orientation | Visualization

Show Port	Port Side	Name	Origin Position Vector [x y z]	Units	Translated from Origin of	Components in Axes of
☑	Right	CG	[0 0 0]	m	CS1	World
☑	Left	CS1	[0 0 0]	m	Adjoining	Adjoining
☐	Left	CS2	[0, 0.05, 0.025]	m	CG	CG
☐	Left	CS3	[0, -0.05, 0.025]	m	CG	CG
☐	Left	CS4	[0, -0.05, -0.025]	m	CG	CG
☐	Left	CS5	[0, 0.05, -0.025]	m	CG	CG



Body

Represents a user-defined rigid body. Body defined by mass m, inertia tensor I, and coordinate origins and axes for center of gravity (CG) and other user-specified Body coordinate systems. This dialog sets Body initial position and orientation, unless Body and/or connected Joints are actuated separately. This dialog also provides optional settings for customized body geometry and color.

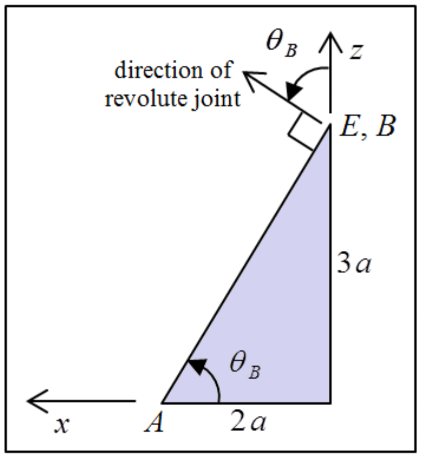
Mass properties

Mass:	1	kg
Inertia:	(1/12)*1*(0.2^2)*[3,0,0; 0,1,0; 0,0,2]	kg*m^2

Position | Orientation | Visualization

Show Port	Port Side	Name	Orientation Vector	Units	Relative CS	Specified Using Convention
☑	Right	CG	[0 0 0]	deg	World	Euler X-Y-Z
☑	Left	CS1	[0, 180*atan(3/2)/pi, 0]	deg	World	Euler X-Y-Z
☐	Left	CS2	[0 0 0]	deg	World	Euler X-Y-Z
☐	Left	CS3	[0 0 0]	deg	World	Euler X-Y-Z
☐	Left	CS4	[0 0 0]	deg	World	Euler X-Y-Z
☐	Left	CS5	[0 0 0]	deg	World	Euler X-Y-Z

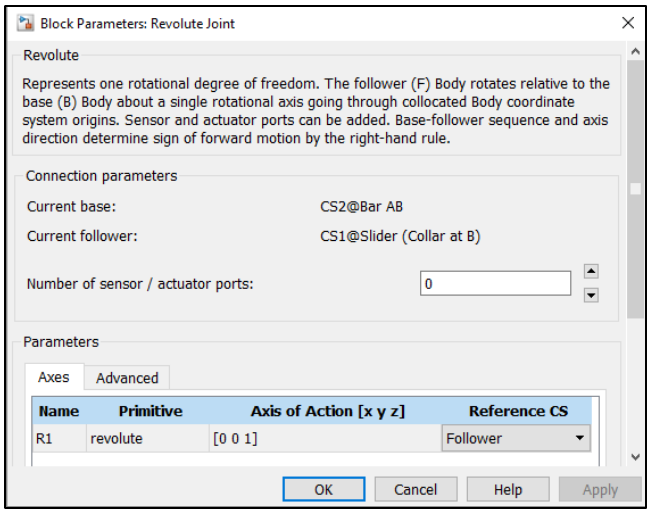
The orientation of ***all*** the coordinate systems of the slider are ***initially aligned*** with the world coordinate system, ***except CS1***. ***CS1*** is rotated relative to the world system about the ***world's y-axis***, so the *z*-axis of ***CS1*** is ***perpendicular*** to ***plane AEB*** and ***initially aligned*** with the ***axis*** of the ***revolute joint*** between the bar and the slider. See the diagram to the right. The angle θ_B in the diagram represents the angle through which ***CS1*** must be rotated. As shown in the dialog box, θ_B may be calculated as



$$\boxed{\theta_B = \left(\frac{180}{\pi}\right)\tan^{-1}\left(\frac{3a}{2a}\right) = \left(\frac{180}{\pi}\right)\tan^{-1}\left(\frac{3}{2}\right) = 56.3099 \text{ (deg)}}$$

<u>Revolute Joint</u>: Connecting Bar *AB* to Slider (Collar)

The ***dialog box*** for the ***revolute joint*** that connects bar *AB* to the slider is shown in the panel to the right. The ***base*** is coordinate system ***CS2*** of the ***bar***, and the follower is coordinate system ***CS1*** on the ***slider***. Note the joint is specified (at the bottom of the box) as a ***revolute joint*** about the ***z-axis*** of the ***follower***. As indicated above, the ***z-axis*** of the ***slider*** (follower) has been ***initially oriented perpendicular*** to ***plane AEB***. As the system ***moves away*** from the initial configuration, the ***axis*** of the ***revolute joint*** will ***remain perpendicular*** to plane *AEB*.


Block Parameters: Revolute Joint

Revolute

Represents one rotational degree of freedom. The follower (F) Body rotates relative to the base (B) Body about a single rotational axis going through collocated Body coordinate system origins. Sensor and actuator ports can be added. Base-follower sequence and axis direction determine sign of forward motion by the right-hand rule.

Connection parameters

Current base:	CS2@Bar AB
Current follower:	CS1@Slider (Collar at B)
Number of sensor / actuator ports:	0

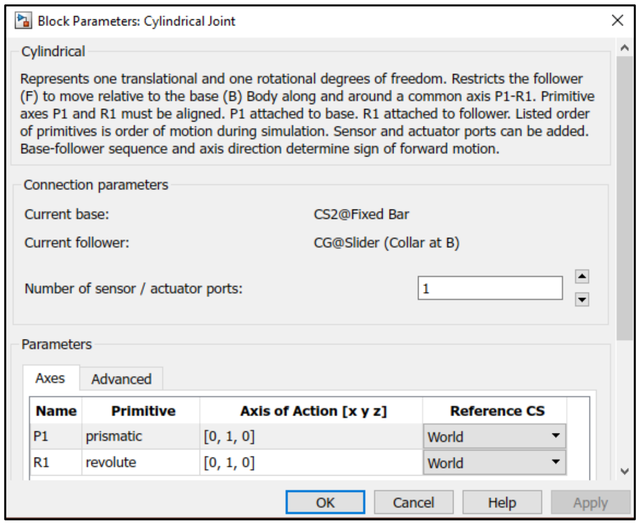
Parameters

Axes | Advanced

Name	Primitive	Axis of Action [x y z]	Reference CS
R1	revolute	[0 0 1]	Follower

<u>Cylindrical Joint</u>: Connecting Slider (Collar) to Ground

The ***dialog box*** for the ***cylindrical joint*** that connects the ***slider*** to the ***fixed bar*** is shown in the panel to the right. The ***base*** of the joint is coordinate system ***CS2*** of the fixed bar located at world coordinates $[0,6a,3a]$, and the ***follower*** is the ***CG*** coordinate system of the ***slider***. This joint has ***two primitive actions***, one ***prismatic*** and one ***revolute***. The ***prismatic*** primitive allows ***translation*** and the ***revolute*** primitive allows ***rotation*** of the slider about the ***world coordinate system's y-axis***. The coordinates of ***CS2*** of the ***fixed bar*** are specified in its dialog box.


Block Parameters: Cylindrical Joint

Cylindrical

Represents one translational and one rotational degrees of freedom. Restricts the follower (F) to move relative to the base (B) Body along and around a common axis P1-R1. Primitive axes P1 and R1 must be aligned. P1 attached to base. R1 attached to follower. Listed order of primitives is order of motion during simulation. Sensor and actuator ports can be added. Base-follower sequence and axis direction determine sign of forward motion.

Connection parameters

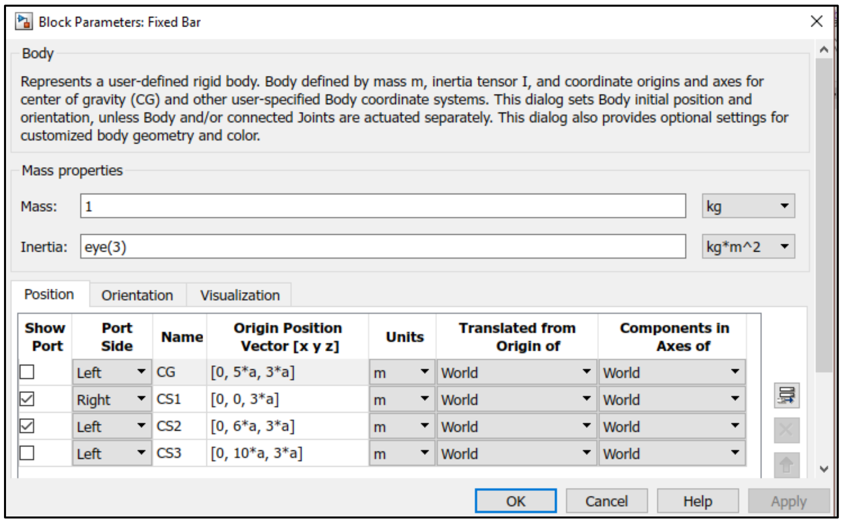
Current base:	CS2@Fixed Bar
Current follower:	CG@Slider (Collar at B)
Number of sensor / actuator ports:	1

Parameters

Axes | Advanced

Name	Primitive	Axis of Action [x y z]	Reference CS
P1	prismatic	[0, 1, 0]	World
R1	revolute	[0, 1, 0]	World

<u>Body block</u>: Fixed Bar


Block Parameters: Fixed Bar

Body

Represents a user-defined rigid body. Body defined by mass m, inertia tensor I, and coordinate origins and axes for center of gravity (CG) and other user-specified Body coordinate systems. This dialog sets Body initial position and orientation, unless Body and/or connected Joints are actuated separately. This dialog also provides optional settings for customized body geometry and color.

Mass properties

Mass: 1 kg

Inertia: eye(3) kg*m^2

Position | Orientation | Visualization

Show Port	Port Side	Name	Origin Position Vector [x y z]	Units	Translated from Origin of	Components in Axes of
☐	Left	CG	[0, 5*a, 3*a]	m	World	World
✓	Right	CS1	[0, 0, 3*a]	m	World	World
✓	Left	CS2	[0, 6*a, 3*a]	m	World	World
☐	Left	CS3	[0, 10*a, 3*a]	m	World	World

The dialog box for the fixed bar that the slider moves along is shown to the right. Coordinate systems ***CS1*** and ***CS3*** mark the ends of the bar at $[0,0,3a]$ and $[0,10a,3a]$ in the world coordinate system, and coordinate system ***CG*** is located at the midpoint of the bar. Coordinate system ***CS2*** is located at the cylindrical joint between bar *AB* and the slider in the initial position.

Clearly, the fixed bar is ***parallel*** to the ***y-axis*** of the world coordinate system. Not shown here is the "Visualization" tab which indicates the locations of the body's coordinate systems will be used to define the bar's geometry.

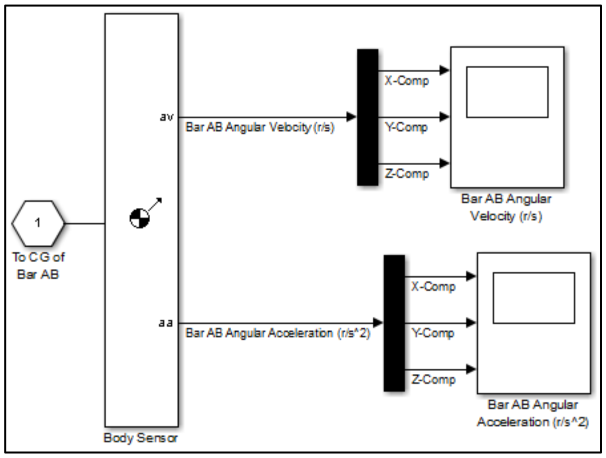
<u>Crank Motion Subsystems</u>:

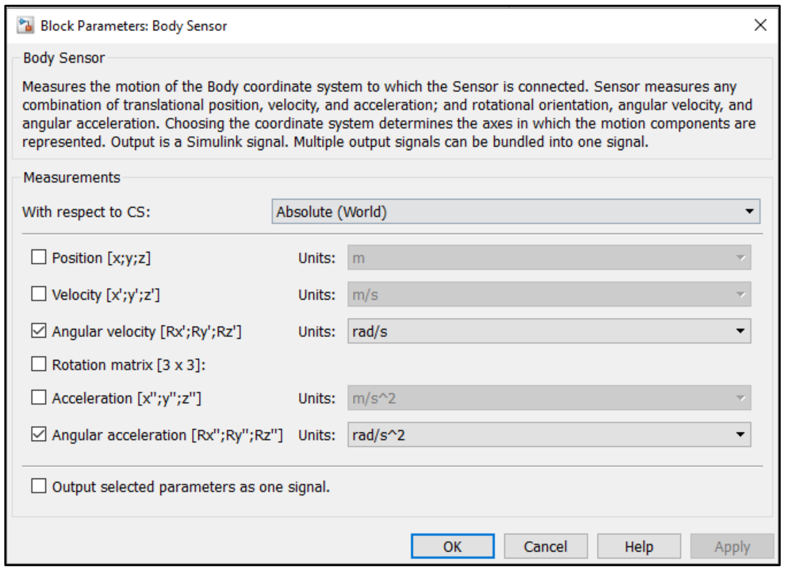
The ***crank motion subsystems*** are very similar to those presented in Example 4. The subsystem that ***specifies*** the ***motion*** of the crank is connected to an ***actuator*** block, and the subsystem that ***measures*** the ***motion*** is connected to a ***sensor*** block. The measurement subsystem provides ***verification*** to the user that the motion specified is ***correctly implemented***.

<u>Bar *AB* Measurement Subsystem</u>:

The ***measurement subsystem*** for **bar** *AB* and the ***dialog box*** for the ***body sensor*** are shown in the panels below. In this case, the ***body sensor*** is used to ***measure*** the ***angular velocity*** and ***angular acceleration*** of the bar. As the dialog box indicates, the ***components*** of the vectors are ***resolved*** in the directions of the ***world coordinate system***. The ***drop down box*** in the Measurements section of the dialog box also allows the user to resolve the vector components in the directions of the ***local body coordinate system***.

The subsystem ***splits*** each of the vectors into components and ***plots*** their time histories. The "History" tabs on "Parameters" menu of the scopes indicate that these signals are also sent to the MATLAB ***workspace*** using a structure with the associated time values. The name of the ***angular velocity structure*** is "barAngularVelocity" and the name of the ***angular acceleration structure*** is "barAngularAcceleration".


av
Bar AB Angular Velocity (r/s)
X-Comp
Y-Comp
Z-Comp
Bar AB Angular Velocity (r/s)
1
To CG of Bar AB
aa
Bar AB Angular Acceleration (r/s^2)
X-Comp
Y-Comp
Z-Comp
Bar AB Angular Acceleration (r/s^2)
Body Sensor


Block Parameters: Body Sensor

Body Sensor

Measures the motion of the Body coordinate system to which the Sensor is connected. Sensor measures any combination of translational position, velocity, and acceleration; and rotational orientation, angular velocity, and angular acceleration. Choosing the coordinate system determines the axes in which the motion components are represented. Output is a Simulink signal. Multiple output signals can be bundled into one signal.

Measurements

With respect to CS: Absolute (World)

☐ Position [x;y;z] Units: m

☐ Velocity [x';y';z'] Units: m/s

☑ Angular velocity [Rx';Ry';Rz'] Units: rad/s

☐ Rotation matrix [3 x 3]:

☐ Acceleration [x'';y'';z''] Units: m/s^2

☑ Angular acceleration [Rx'';Ry'';Rz''] Units: rad/s^2

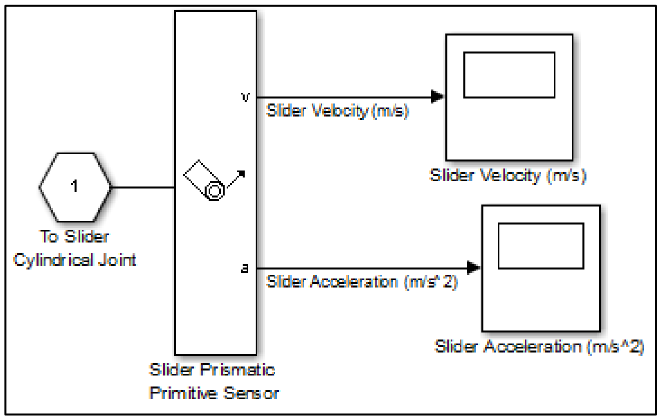
☐ Output selected parameters as one signal.

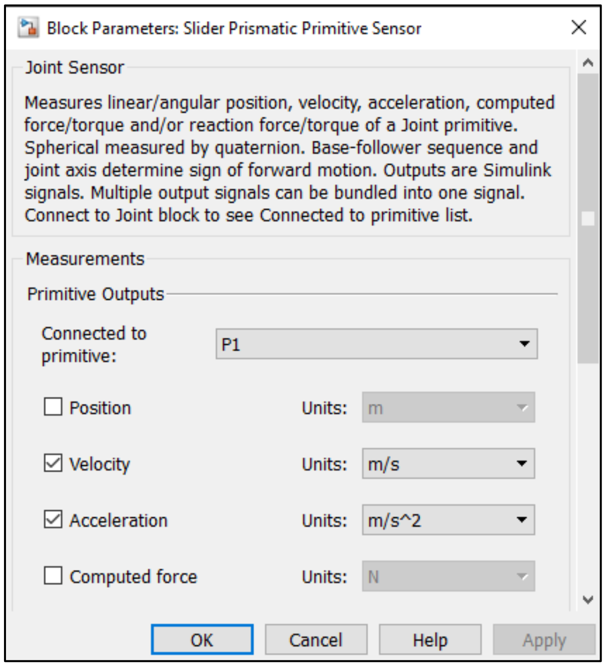
OK Cancel Help Apply

<u>Cylindrical Joint Measurement Subsystem</u>:

The ***measurement subsystem*** for the ***cylindrical joint*** and the ***dialog box*** for the ***joint sensor*** are shown in the panels below. In this case, the ***sensor*** is used to ***measure*** the ***velocity*** and ***acceleration*** of the ***slider*** along the joint axis. Using the dialog box, the user can measure results associated with either the "P1" (translational) or "R1" (rotational) ***primitives*** associated with the joint. In this case the dialog box indicates the ***sensor*** is ***connected*** to the ***translational primitive*** and is ***measuring*** the ***velocity*** and ***acceleration*** of the slider along the joint axis.

The "History" tabs on "Parameters" menu of the ***scopes*** indicate that these signals are also sent to the MATLAB ***workspace*** using a structure with the associated time values. The name of the ***velocity structure*** is "sliderVelocity" and the name of the ***acceleration structure*** is "sliderAcceleration".


v
Slider Velocity (m/s)
Slider Velocity (m/s)
1
To Slider Cylindrical Joint
a
Slider Acceleration (m/s^2)
Slider Acceleration (m/s^2)
Slider Prismatic Primitive Sensor


Block Parameters: Slider Prismatic Primitive Sensor

Joint Sensor

Measures linear/angular position, velocity, acceleration, computed force/torque and/or reaction force/torque of a Joint primitive. Spherical measured by quaternion. Base-follower sequence and joint axis determine sign of forward motion. Outputs are Simulink signals. Multiple output signals can be bundled into one signal. Connect to Joint block to see Connected to primitive list.

Measurements

Primitive Outputs

Connected to primitive: P1

☐ Position Units: m

☑ Velocity Units: m/s

☑ Acceleration Units: m/s^2

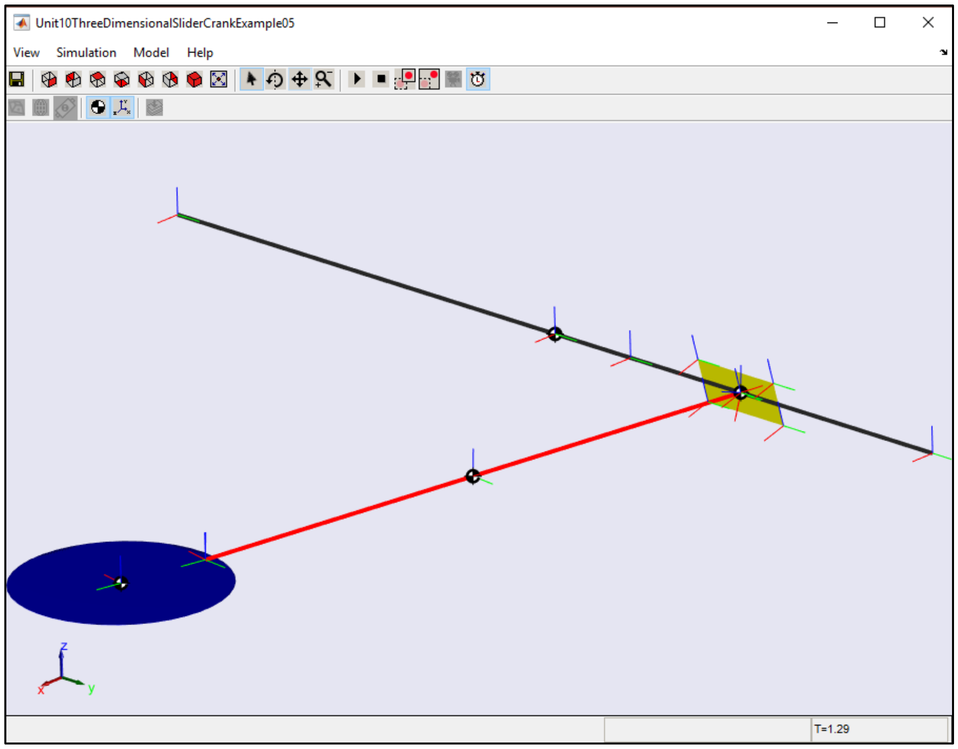
☐ Computed force Units: N

OK Cancel Help Apply

Results:

When the MATLAB script is executed, model input ***variables*** are ***defined***, the SimMechanics ***model executed***, SimMechanics ***results*** are ***plotted*** in Figures 1-7, ***analytical results*** are ***calculated*** at $t = 0$ and ***displayed*** in the MATLAB command window, and finally, ***analytical results*** are ***calculated*** for $t \geq 0$ and ***plotted*** in Figures 8-11. When the SimMechanics model is executed, an ***animation window*** opens to display the system in motion.

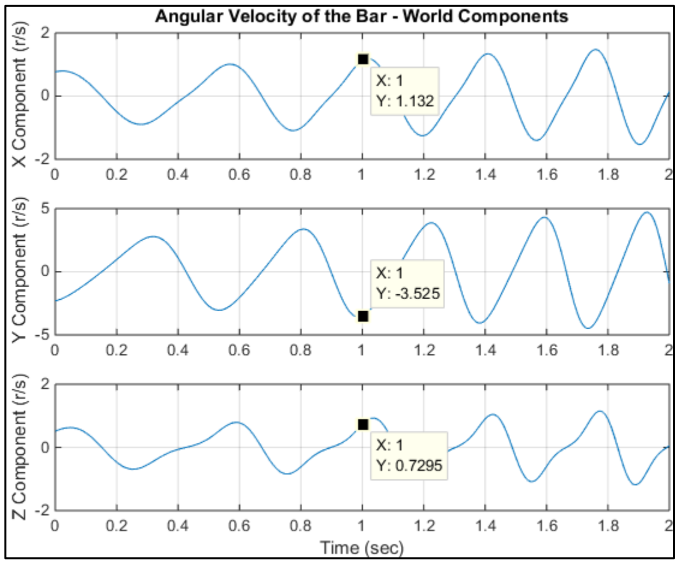
A ***snapshot*** of the animation of the system at $t = 1.29\ (\text{sec})$ is shown below. As previously noted, the ***slider*** is ***depicted*** as a ***rectangle*** for animation purposes. Using this approach clearly shows that the ***slider rotates*** and ***translates*** along the *y*-direction as the crank rotates. This rotational motion could be ***measured*** by ***connecting*** a ***sensor*** to the "R1" ***primitive*** of the cylindrical joint.

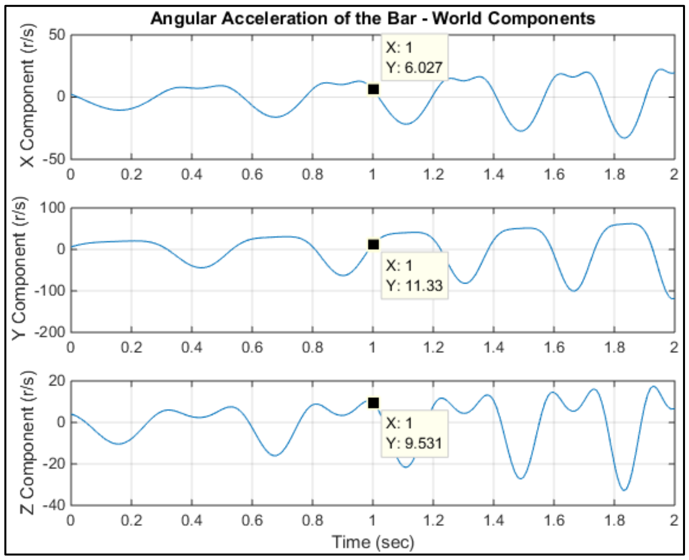


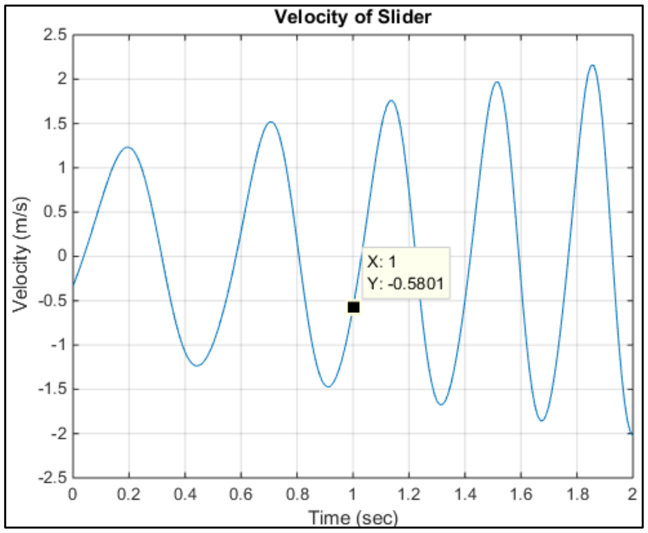
SimMechanics Model Results:

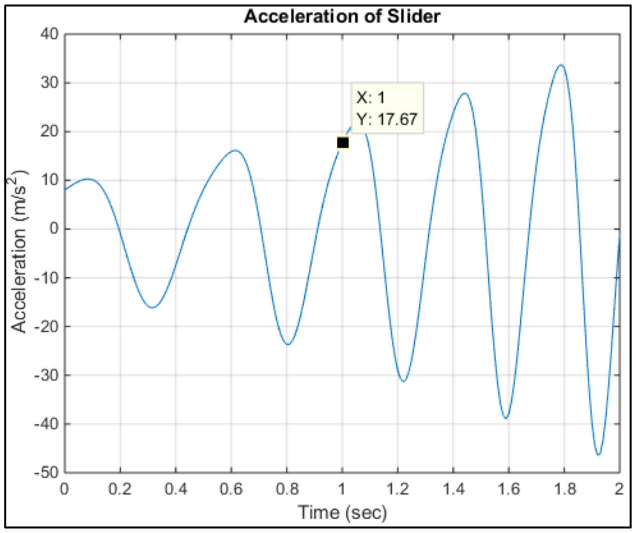
The script generates ***eleven plot figures*** as it executes. ***Figures 1-3*** are used to ***verify*** that the ***angular motion*** of the ***crank*** has been correctly calculated and are not shown here. ***Figures 4-5*** show the ***world components*** of the ***angular velocity*** and ***angular acceleration*** of ***bar AB*** and are shown in the first set of plots below. The ***velocity*** and ***acceleration*** of the ***slider*** are plotted in ***Figures 6-7*** and are shown in the second set of plots. Data cursors have been used to show values at $t = 1\ (\text{sec})$.

Results generated using the ***analytical equations*** from ***Unit 8*** are plotted in ***Figures 8-11***. The results (not shown here) are found to be identical to those generated by SimMechanics.





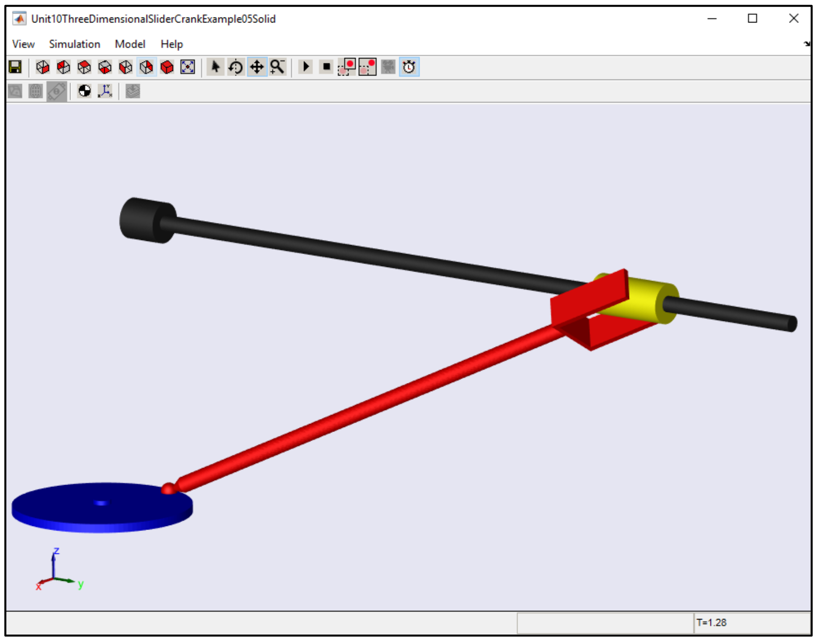


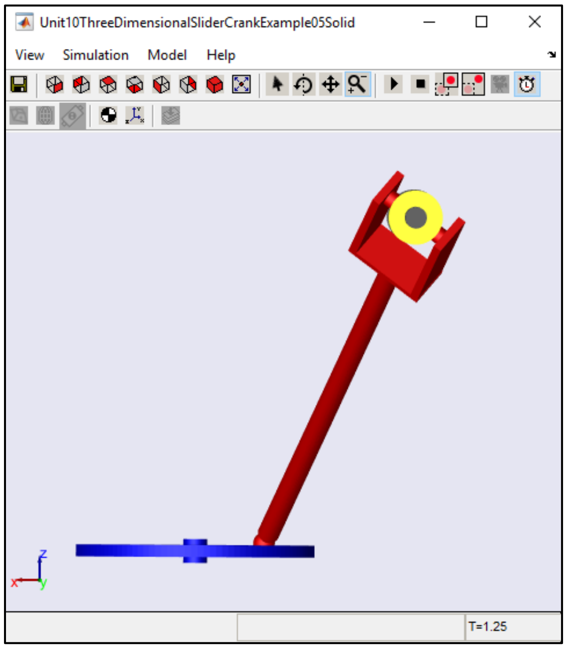


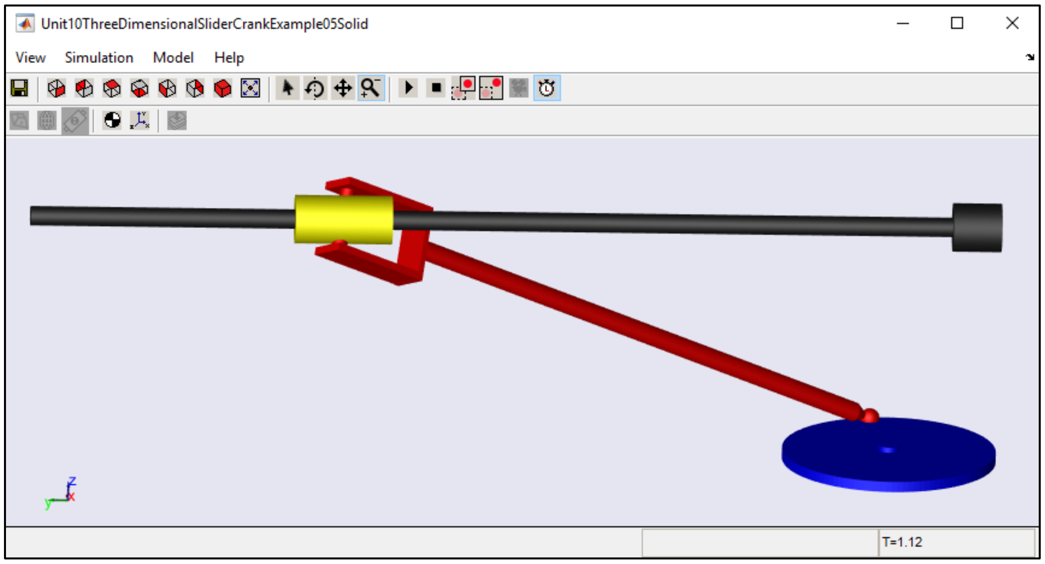
Increasing the Quality of the Animation:

As presented above, the body geometry of the ***disk*** was determined using an "Equivalent ellipsoid from the mass properties", and the body geometries of the ***arm***, ***slider***, and ***fixed arm*** were determined by the ***locations*** of the ***body coordinate systems*** as indicated on the "Visualization" tabs of the body dialog boxes. As in Example 4, the animation results for this example can be improved using ***external stereolithographic files*** to provide the geometries of the bodies.

Three ***improved*** animation windows are shown below. The figure on the top left shows the position of the system from a prospective like that shown above for the previous model. The other two figures each show the position of the system from a prospective that more clearly shows the nature of the revolute joint between the bar and the slider. Note that a third angle was used to orient bar *AB* so the fork would be aligned to accept the slider.


Unit10ThreeDimensionalSliderCrankExample05Solid
View Simulation Model Help
T=1.28

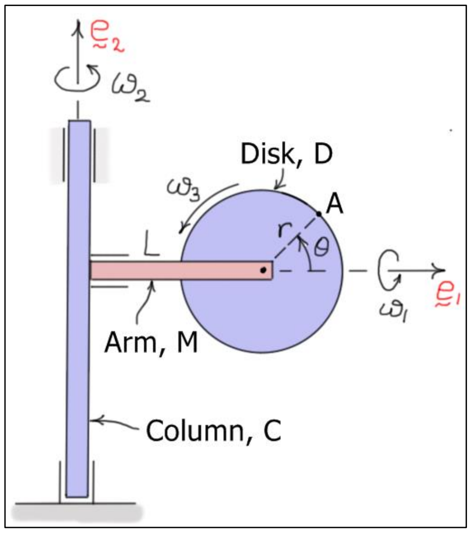

Unit10ThreeDimensionalSliderCrankExample05Solid
View Simulation Model Help
T=1.25


Unit10ThreeDimensionalSliderCrankExample05Solid
View Simulation Model Help
T=1.12

Exercises:

10.1 Write a MATLAB script to convert a 2-3-1 body-fixed orientation angle sequence to an equivalent set of Euler parameters. Check your results as in Example 1. You can make use of the scripts provided above or you can write your own.

10.2 Write a MATLAB script to calculate the velocity and acceleration of point A at a series of times starting in the position shown with $\theta = 0$. Resolve the components of the two vectors in the disk-fixed system. At $t = 0$ the coordinate systems of all the bodies are aligned, and all relative angular accelerations are constant. Use the following data:



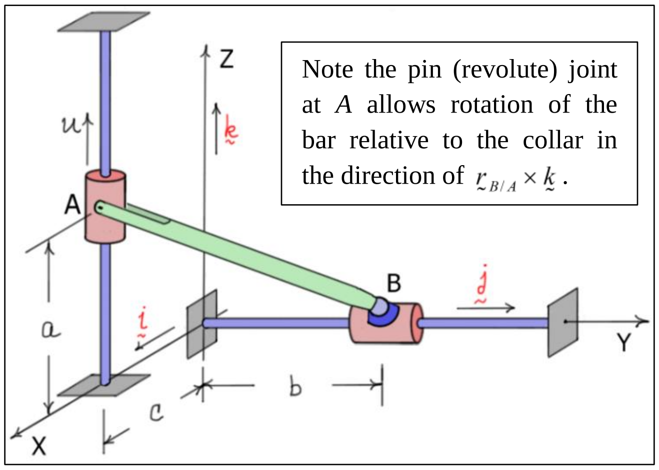
$L = 0.5$ (m)	$r = 0.25$ (m)
$\dot{\omega}_2 = 3$ (rad/s^2) $=$ constant	$\omega_2(t=0) = 2$ (rad/s)
$\dot{\omega}_1 = 4$ (rad/s^2) $=$ constant	$\omega_1(t=0) = 3$ (rad/s)
$\dot{\omega}_3 = 5$ (rad/s^2) $=$ constant	$\omega_3(t=0) = 4$ (rad/s)

Plot the disk-fixed components of ${}^R\underset{\sim}{\omega}_D$, ${}^R\underset{\sim}{\alpha}_D$, ${}^R\underset{\sim}{v}_A$, and ${}^R\underset{\sim}{a}_A$ for $0 \le t \le 3$ (sec).

10.3 Develop a Simulink model to calculate the motion for the system of Exercise 10.2. Plot your results and compare them with the script results from Exercise 10.2. Use a script to define the necessary variables, run the Simulink model, and plot the results.

10.4 Develop a SimMechanics model to calculate the motion for the system of Exercise 10.2. Use a column length of 1.5 (m) and attach the arm two-thirds of the way up the column. Plot your results and compare them with the script results from Exercise 10.2 and the Simulink results from Exercise 10.3. Use a script to define the necessary variables, run the SimMechanics model, and plot the results.

10.5 The system shown consists of bar AB whose ends are connected to collars that slide along the two fixed poles. The collar at B can only translate along the horizontal bar (prismatic joint), while the collar at A can both translate and rotate relative to the vertical bar (cylindrical joint). The bar is connected to the collar at B using a ball and socket joint, and it is connected to the collar at A using a pin joint. Neglect the size of the collars. Use the following data:



Note the pin (revolute) joint at A allows rotation of the bar relative to the collar in the direction of $\underset{\sim}{r}_{B/A} \times \underset{\sim}{k}$.

$a = 0.3$ (m) $\quad b = 0.6$ (m) $\quad c = 0.2$ (m) $\quad u(t) = 0.1\sin(2\pi t)$ (m)

Write a script to calculate ${}^R\underset{\sim}{\omega}_{AB}$, ${}^R\underset{\sim}{\alpha}_{AB}$, ${}^R\underset{\sim}{v}_B$, ${}^R\underset{\sim}{a}_B$. Plot the results for $0 \le t \le 2$ (sec).

10.6 Develop a Simulink model for the same calculations. Plot your results and compare them with the script results from Exercise 10.5. Use a script to define the necessary variables, run the Simulink model, and plot the results.

10.7 Develop a SimMechanics model for the same calculations. Plot your results and compare them with the script results from Exercise 10.5 and the Simulink results from Exercise 10.6. Use a script to define the necessary variables, run the SimMechanics model, and plot the results.

References:

1. H. Baruh, *Analytical Dynamics*, McGraw-Hill, 1999
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. Josephs and R.L. Huston, *Dynamics of Mechanical Systems*, CRC Press, 2002
6. R.C. Hibbeler, *Engineering Mechanics: Dynamics*, 13th Ed., Pearson Prentice Hall, 2013
7. J.L. Meriam and L.G. Craig, *Engineering Mechanics: Dynamics*, 3rd Ed, 1992