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 Simscape Multibody®

Summary

This unit introduces ***modeling*** mechanical system kinematics using ***MATLAB scripts***, Simulink ***models***, and ***Simscape 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/Simulink environment. Simscape Multibody models are ***block-diagram-based, multibody dynamics*** programs that run in the MATLAB/Simulink/Simscape environment. MATLAB scripts can be used ***alone*** or ***in conjunction with*** Simulink and Simscape Multibody 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 ***build*** and ***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.

Simscape Multibody 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 Simscape Multibody ***develops*** and ***solves*** the equations of motion. These models may 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
49	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 ***three orientation angles***, and using that transformation matrix, it then ***calls a second function*** to calculate the ***four Euler parameters*** associated with that transformation matrix.

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 two transformation matrices should be the same, and the matrix product should yield the identity matrix. Recall that these transformation matrices are ***orthogonal matrices,*** so 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. The script of the main module and a set of ***sample output*** to the Command Window are shown below.

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; % degrees to radians conversion factor

%  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]*transpose[Re] to see how close to identity matrix
   identityCheck = transformationMatrix*(transformationMatrixE');
                                        ⋮
```

```
                                        ⋮
%  display the results
   disp(' ')
   disp('Angles (deg):     (1-2-3 body-fixed rotation sequence)')
   tempstr = ['   Theta_1 = ',num2str(theta1Deg), ...
              ', Theta_2 = ',num2str(theta2Deg), ...
              ', Theta_3 = ',num2str(theta3Deg)];
   disp(tempstr)
   disp(' ')
   disp('Euler Parameters:')
   tempstr = ['   Epsilon_1 = ',num2str(eulerParameter(1)), ...
              ', Epsilon_2 = ',num2str(eulerParameter(2))];
   disp(tempstr)
   tempstr = ['   Epsilon_3 = ',num2str(eulerParameter(3)), ...
              ', Epsilon_4 = ',num2str(eulerParameter(4))];
   disp(tempstr)
   disp(' ')
   disp('Transformation Matrix:     (based on the angles)')
   disp(transformationMatrix)
   disp(' ')
   disp('Transformation Matrix:     (based on the Euler parameters)')
   disp(transformationMatrixE)
   disp(' ')
   disp('Identity matrix?')
   disp(identityCheck)
```

Command Window Output:

```
>> angles123toEulerParameters

Angles (deg):     (1-2-3 body-fixed rotation sequence)
   Theta_1 = 180, Theta_2 = 30, Theta_3 = 20

Euler Parameters:
   Epsilon_1 = 0.95125, Epsilon_2 = -0.16773
   Epsilon_3 = 0.25489, Epsilon_4 = -0.044943

Transformation Matrix:     (based on the angles)
   0.813797681349374  -0.342020143325669   0.469846310392954
  -0.296198132726024  -0.939692620785908  -0.171010071662834
   0.500000000000000  -0.000000000000000  -0.866025403784439

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 gives the form of the coordinate transformation matrix $[R]$ in terms of the 1-2-3 orientation angles and 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 as 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 - 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);

%   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;

    eulerParameter = sqrt(eulerParametersq);

%   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 largerst 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
```

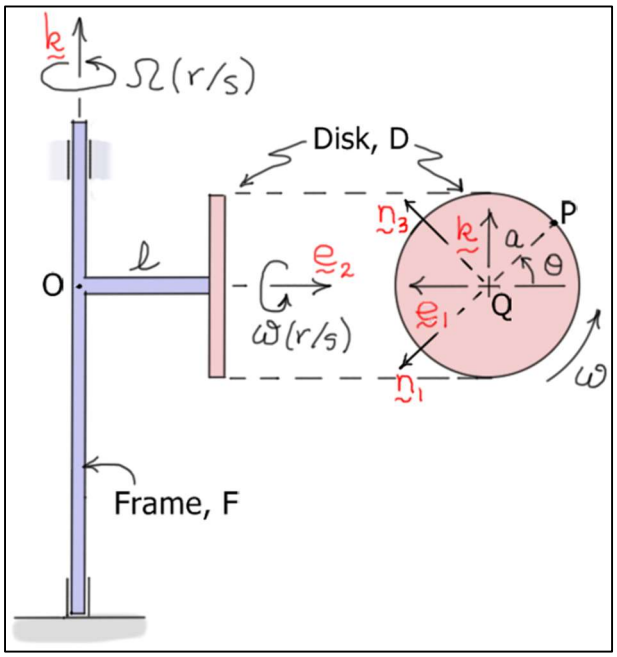
Function: `calculateTransformationMatrixFromEulerParameters`

```
function [transformationMatrix] = ...
                calculateTransformationMatrixFromEulerParameters(eulerParameter)
%
%   This function calculates the coordinate transformation matrix
%   associated with a set of 4 Euler parameters
%
%   Input:
%   4x1 eulerParameter 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 – Frame F and 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.

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

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

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

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 frame D.

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

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

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

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 are 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 all 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 $t = 2$ (sec):

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

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

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

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

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***. Panel 1 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.

Script – Panel 1: (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

   clear variables

%% 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
   phi       = zeros(numberoftimes,1); theta     = zeros(numberoftimes,1);
   capOmega  = zeros(numberoftimes,1); omega     = zeros(numberoftimes,1);
   omegaDinR = zeros(numberoftimes,3); alphaDinR = zeros(numberoftimes,3);
   velocityofPn = zeros(numberoftimes,3); accelerationofPn = zeros(numberoftimes,3);
                                          ⋮
```

Panel 2 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 $0 \rightarrow 4$ (sec) in steps of 0.01 (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.

Panel 3 contains the ***fourth section*** of the script. It contains the statements necessary to create four figures, each having three subplots. The values of the $\underset{\sim}{n}_1$, $\underset{\sim}{e}_2$, 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

for itime = 1:numberoftimes

%  Assuming constant values for omegaDot and capOmegaDot
   capOmega(itime) = capOmegaInitial + (capOmegaDot*time(itime));
   phi(itime)      = phiInitial + (capOmegaInitial*time(itime)) + ...
                     (0.5*capOmegaDot*(time(itime)^2));
   omega(itime)    = omegaInitial + (omegaDot*time(itime));
   theta(itime)    = thetaInitial + (omegaInitial*time(itime)) + ...
                     (0.5*omegaDot*(time(itime)^2));

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

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

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

%  velocity components in frame F
   velocityofPe1 = (diskRadius*omega(itime)*sinTheta) - (armLength*capOmega(itime));
   velocityofPe2 = -diskRadius*capOmega(itime)*cosTheta;
   velocityofPe3 =  diskRadius*omega(itime)*cosTheta;

%  velocity components in disk frame D
   velocityofPn(itime,1) = (velocityofPe1*cosTheta) - (velocityofPe3*sinTheta);
   velocityofPn(itime,2) =  velocityofPe2;
   velocityofPn(itime,3) = (velocityofPe1*sinTheta) + (velocityofPe3*cosTheta);

%  acceleration components in frame F
   accelerationofPe1 = (diskRadius*omegaDot*sinTheta) - ...
                       (armLength*capOmegaDot) + ...
                       (diskRadius*cosTheta*((omega(itime)^2) + ...
                       (capOmega(itime)^2)));
   accelerationofPe2 = (-diskRadius*capOmegaDot*cosTheta) - ...
                       (armLength*(capOmega(itime)^2)) + ...
                       (2.0*diskRadius*omega(itime)*capOmega(itime)*sinTheta);
   accelerationofPe3 = diskRadius*((omegaDot*cosTheta) - ...
                                   ((omega(itime)^2)*sinTheta));

%  acceleration components in disk frame D
   accelerationofPn(itime,1) = (accelerationofPe1*cosTheta) - ...
                               (accelerationofPe3*sinTheta);
   accelerationofPn(itime,2) =  accelerationofPe2;
   accelerationofPn(itime,3) = (accelerationofPe1*sinTheta) + ...
                               (accelerationofPe3*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)');

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)');
sgtitle('Angular Velocity of D in R - Components in Frame D');

figure(2); clf;
subplot(3,1,1); plot(time,alphaDinR(:,1),'b-'); grid on;
xlabel('time(sec)'), ylabel('alphaDinR_1 (rad/s^2)');

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)');
sgtitle('Angular Acceleration of D in R - Components in Frame D');

figure(3); clf;
subplot(3,1,1); plot(time,velocityofPn(:,1),'b-'); grid on;
xlabel('time(sec)'), ylabel('v_1 (m/s)');

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

subplot(3,1,3); plot(time,velocityofPn(:,3),'b-'); grid on;
xlabel('time(sec)'), ylabel('v_3 (m/s)');
sgtitle('Velocity Components of Point P in R expressed in Frame D');

figure(4); clf;
subplot(3,1,1); plot(time,accelerationofPn(:,1),'b-'); grid on;
xlabel('time(sec)'), ylabel('a_1 (m/s^2)');

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

subplot(3,1,3); plot(time,accelerationofPn(:,3),'b-'); grid on;
xlabel('time(sec)'), ylabel('a_3 (m/s^2)');
sgtitle('Acceleration Components of Point P in R expressed in Frame D');
```

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.

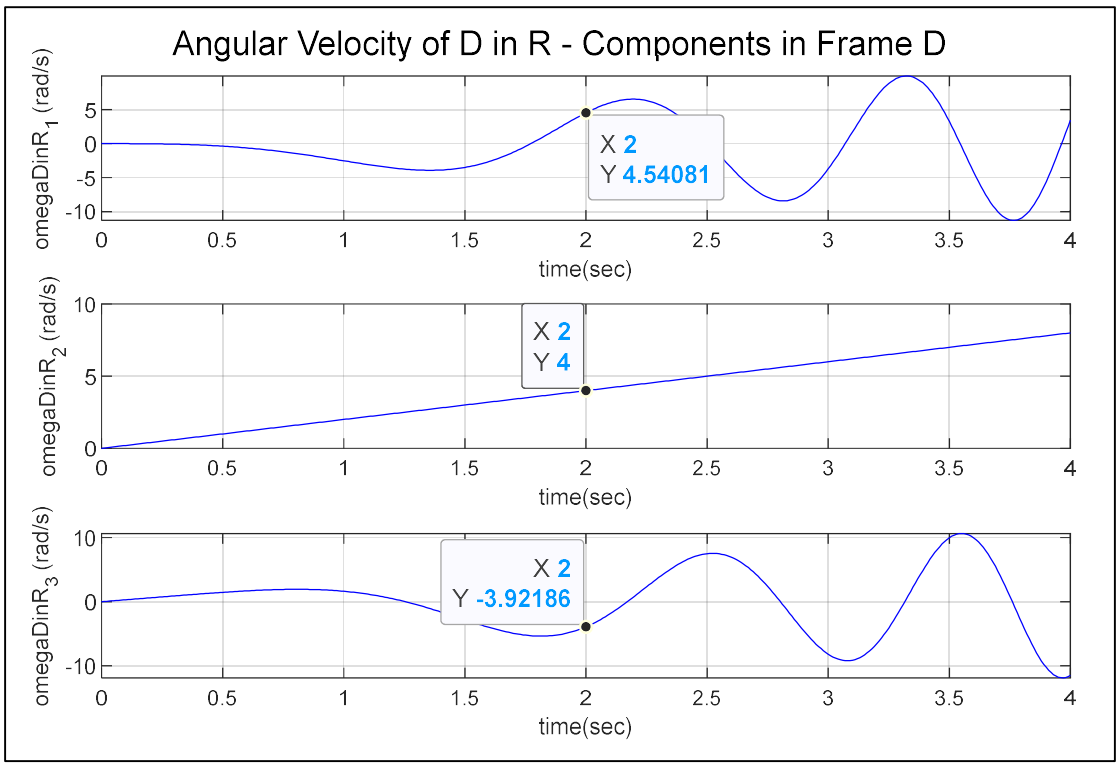


Figure 1. Angular Velocity of D in R - Components in Frame D

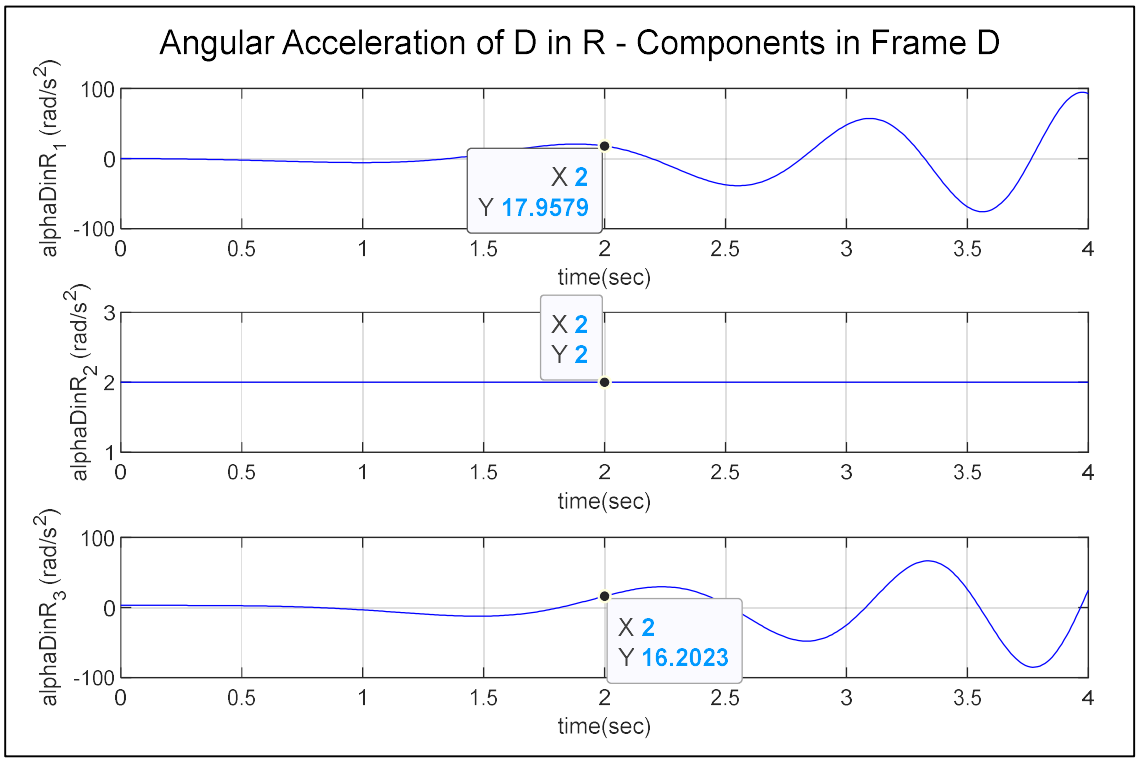


Figure 2. Angular Acceleration of D in R - Components in Frame D

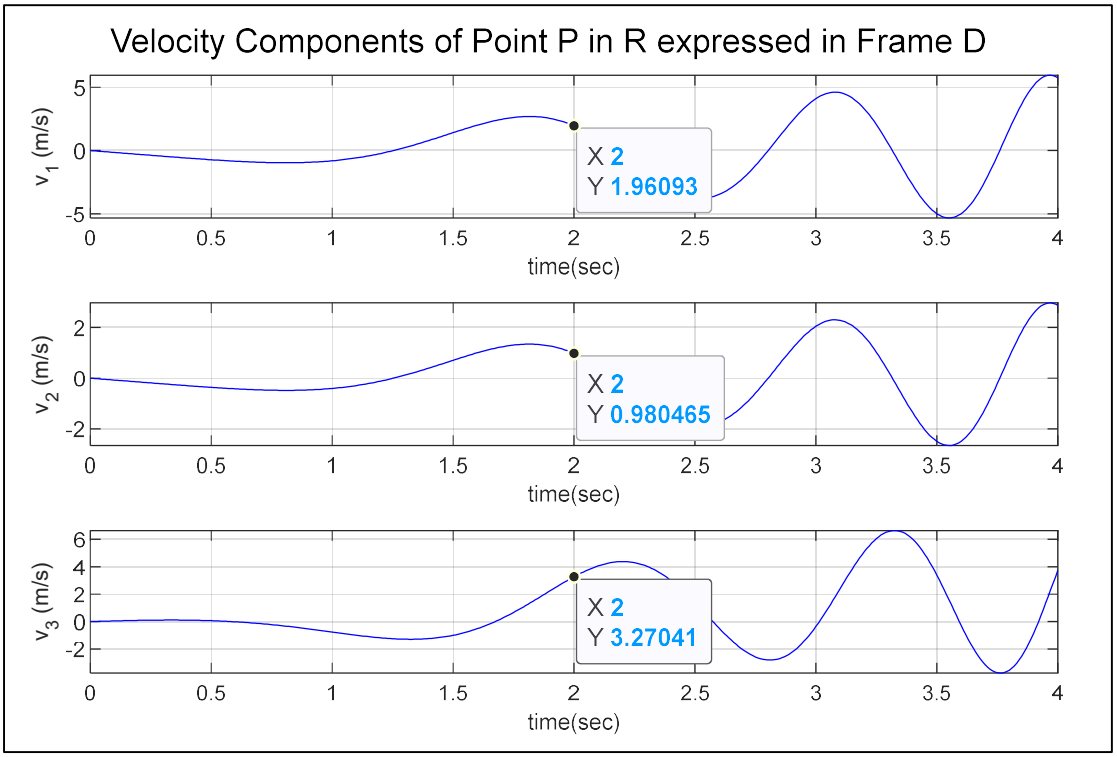


Figure 3. Velocity Components of Point P in R expressed in Frame D

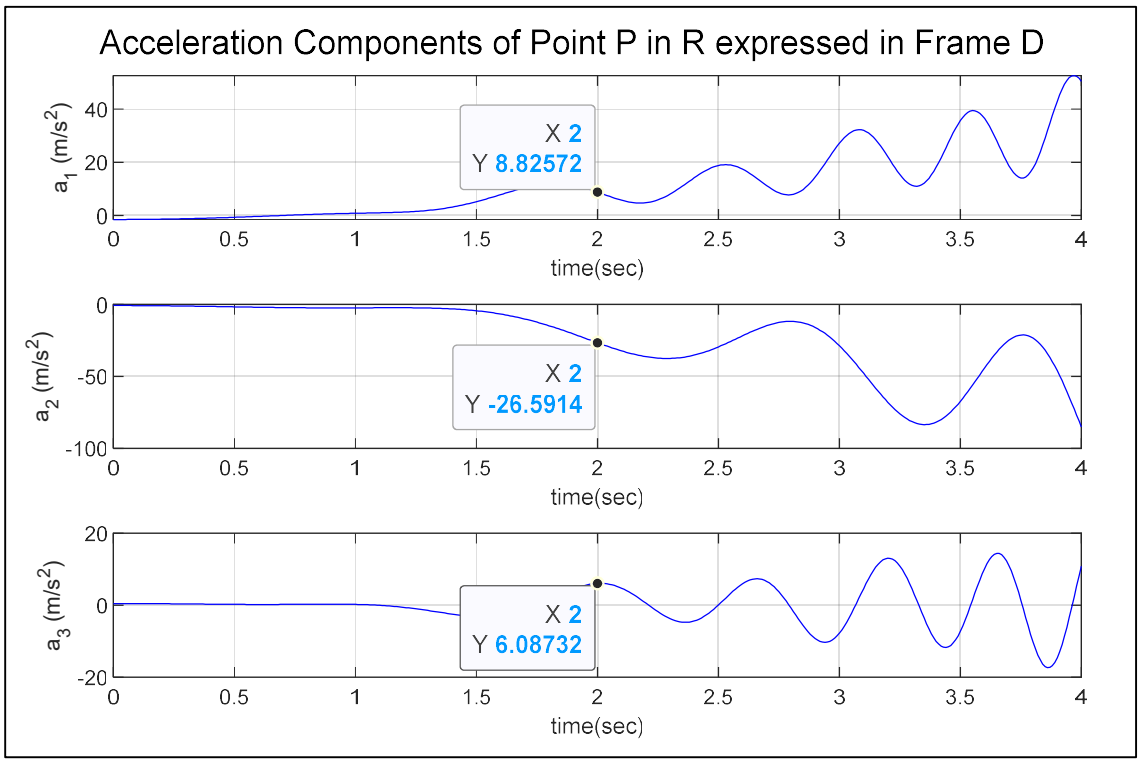
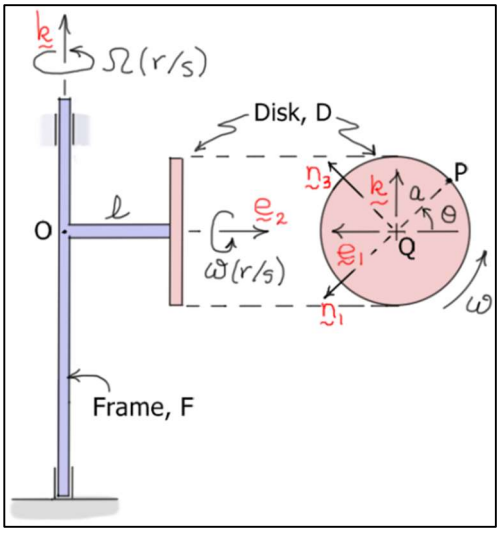


Figure 4. Acceleration Components of Point *P* in R expressed in Frame D

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, Frame F is driven relative to the ground and Disk D is driven relative to F 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***. Panel 1 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: Unit10Example03TwoBodyKinematicsSimulink
%
%  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.

   clear variables
%% 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('Unit10Example03TwoBodyKinematicsSimulinkV2.slx');
                                          ⋮
```

The final section of the script 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 ***structured arrays***. For example, for ${}^R\underset{\sim}{\omega}_D$ the time values are stored in "`omegaDinRinD.time`" and the vector components resolved in the disk frame are

stored in "`omegaDinRinD.signals.values`". See the syntax in Panel 2 below. As with the script for Example 2, 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)

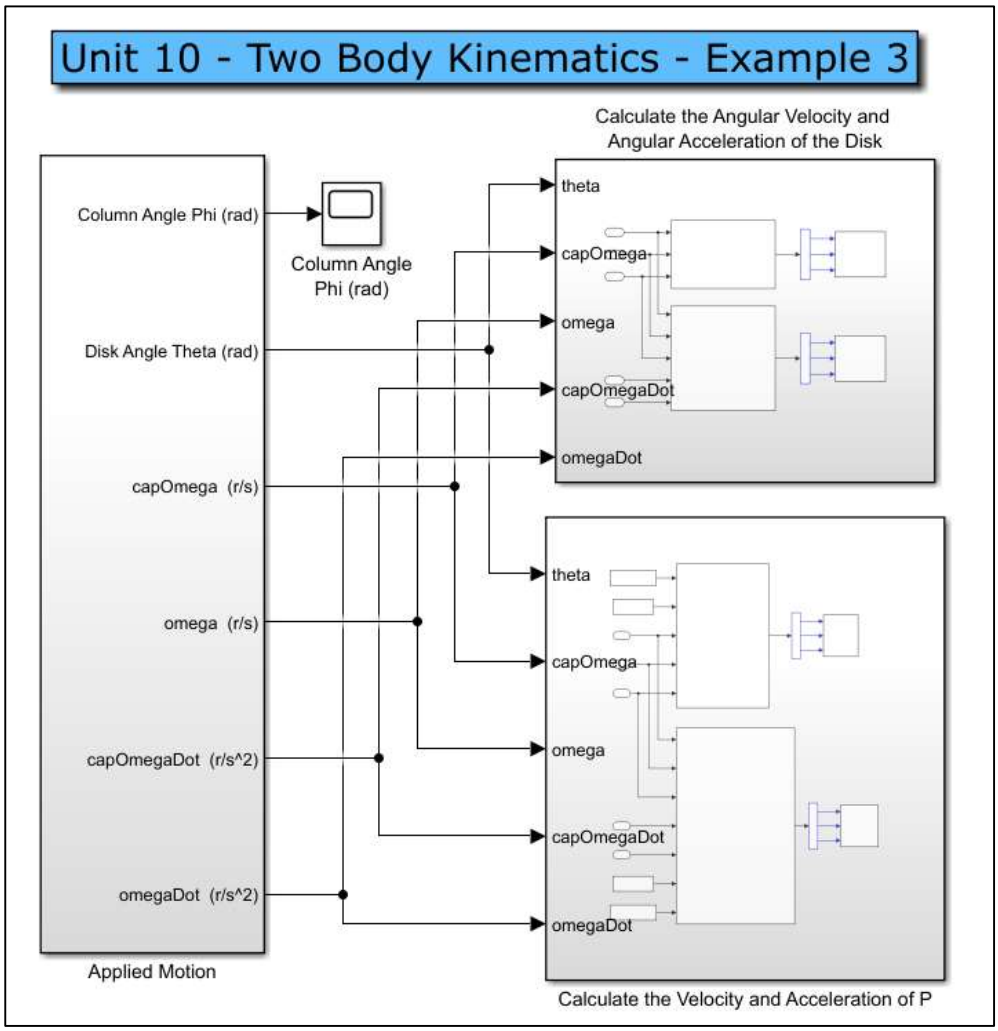
```
                                                                    ⋮
%% Plot the results

   figure(5); clf;
   subplot(3,1,1); plot(omegaDinRinD.time,omegaDinRinD.signals(1).values(:));
   grid; xlabel('Time (sec)'); ylabel('m_1 Component (rad/s)')
   subplot(3,1,2); plot(omegaDinRinD.time,omegaDinRinD.signals(2).values(:));
   grid; xlabel('Time (sec)'); ylabel('m_2 Component (rad/s)')
   subplot(3,1,3); plot(omegaDinRinD.time,omegaDinRinD.signals(3).values(:));
   grid; xlabel('Time (sec)'); ylabel('n_3 Component (rad/s)')
   sgtitle('Angular Velocity of D in Resolved in D (rad/s)');

   figure(6); clf;
   subplot(3,1,1); plot(alphaDinRinD.time,alphaDinRinD.signals(1).values(:));
   grid; xlabel('Time (sec)'); ylabel('m_1 Component (rad/s)')
   subplot(3,1,2); plot(alphaDinRinD.time,alphaDinRinD.signals(2).values(:));
   grid; xlabel('Time (sec)'); ylabel('m_2 Component (rad/s)')
   subplot(3,1,3); plot(alphaDinRinD.time,alphaDinRinD.signals(3).values(:));
   grid; xlabel('Time (sec)'); ylabel('n_3 Component (rad/s)')
   sgtitle('Angular Acceleration of D in R Resolved in D (rad/s)');

   figure(7); clf;
   subplot(3,1,1); plot(velocityPinRinD.time,velocityPinRinD.signals(1).values(:));
   grid; xlabel('Time (sec)'); ylabel('m_1 Component (rad/s)')
   subplot(3,1,2); plot(velocityPinRinD.time,velocityPinRinD.signals(2).values(:));
   grid; xlabel('Time (sec)'); ylabel('m_2 Component (rad/s)')
   subplot(3,1,3); plot(velocityPinRinD.time,velocityPinRinD.signals(3).values(:));
   grid; xlabel('Time (sec)'); ylabel('n_3 Component (rad/s)')
   sgtitle('Velocity of P in R Resolved in D (rad/s)');

   figure(8); clf;
   subplot(3,1,1); plot(accelerationPinRinD.time,accelerationPinRinD.signals(1).values(:));
   grid; xlabel('Time (sec)'); ylabel('m_1 Component (rad/s)')
   subplot(3,1,2); plot(accelerationPinRinD.time,accelerationPinRinD.signals(2).values(:));
   grid; xlabel('Time (sec)'); ylabel('m_2 Component (rad/s)')
   subplot(3,1,3); plot(accelerationPinRinD.time,accelerationPinRinD.signals(3).values(:));
   grid; xlabel('Time (sec)'); ylabel('n_3 Component (rad/s)')
   sgtitle('Acceleration of P in R Resolved in D (rad/s)');
```

Simulink Model – Top Layer:

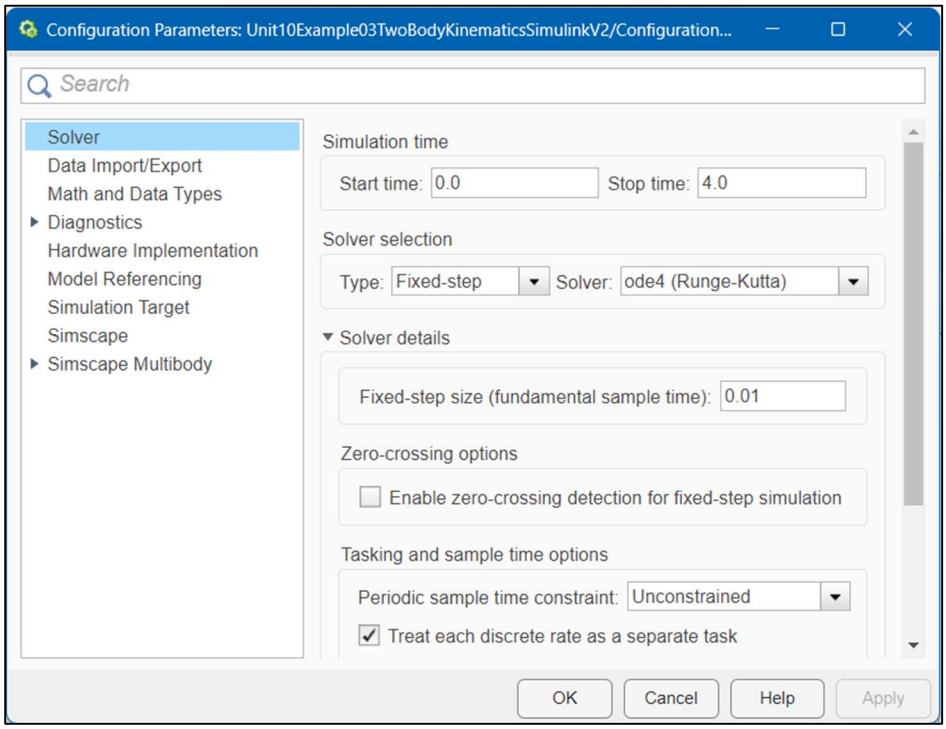
The ***top layer*** of the Simulink model shows the ***overall operation*** of the model and is shown in Panel 1 below. The subsystem on the left side of the diagram generates values for the ***motion variables*** – ϕ, Ω $(=\dot{\phi})$, $\dot{\Omega}$, θ, ω $(=\dot{\theta})$, and $\dot{\omega}$. Variables 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 "Logging" tab of the "Configuration Parameters" menu of each scope determines the variable name and format to be used.

Panel 2 below shows the model "Configuration Parameters" found on the "Modeling" menu of the model. The parameters shown are for the Simulink solver and indicate the simulation will run from $0 \rightarrow 4$ (sec) in fixed steps of 0.01 (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.

Simulink Model – Panel 1 (top layer and plot parameters window)



Simulink Model – Panel 2 (configuration parameters for the model)



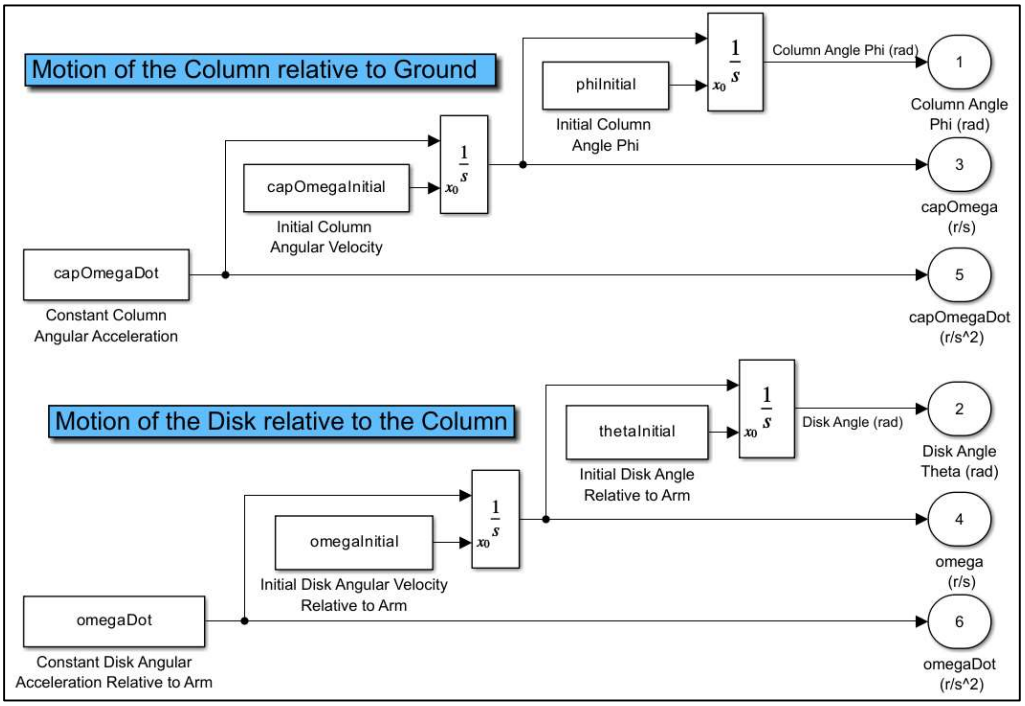
Simulink Model – Motion Subsystem:

Panel 3 below shows the ***applied motion subsystem*** which calculates the ***relative angular motions*** of the system ***assuming*** the relative angular accelerations $\left(\dot{\Omega} \text{ and } \dot{\omega}\right)$ 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". Clearly, ***more complicated motion profiles*** could be generated using this process by simply expressing the relative angular accelerations themselves as functions of time.

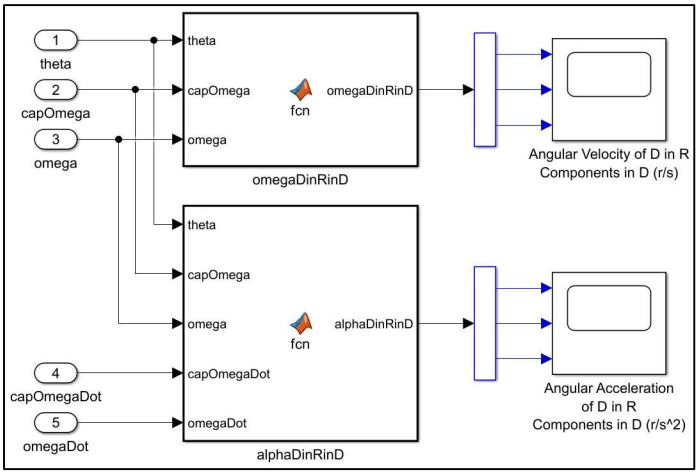
Simulink Model – Vector Component Subsystems:

The variables calculated by the ***applied motion subsystem*** are passed to two ***vector component subsystems*** – one calculates the angular velocity and angular acceleration of disk D and the other calculates the velocity and acceleration of point P on the edge of D. See Panel 4 below. Functions within the subsystems ***first calculate*** the vector components in frame $F:\left(\underset{\sim}{e}_1,\underset{\sim}{e}_2,\underset{\sim}{k}\right)$, and then they ***transform*** them into vector components in the disk frame $D:\left(\underset{\sim}{n}_1,\underset{\sim}{e}_2,\underset{\sim}{n}_3\right)$. The ***analytical equations*** used in the calculations were derived and presented in Unit 3. ***Coordinate transformation*** (rotation) ***matrices*** used to relate vector components in different reference frames are described in Unit 5. Two of the four MATLAB function scripts are shown in Panel 5 below.

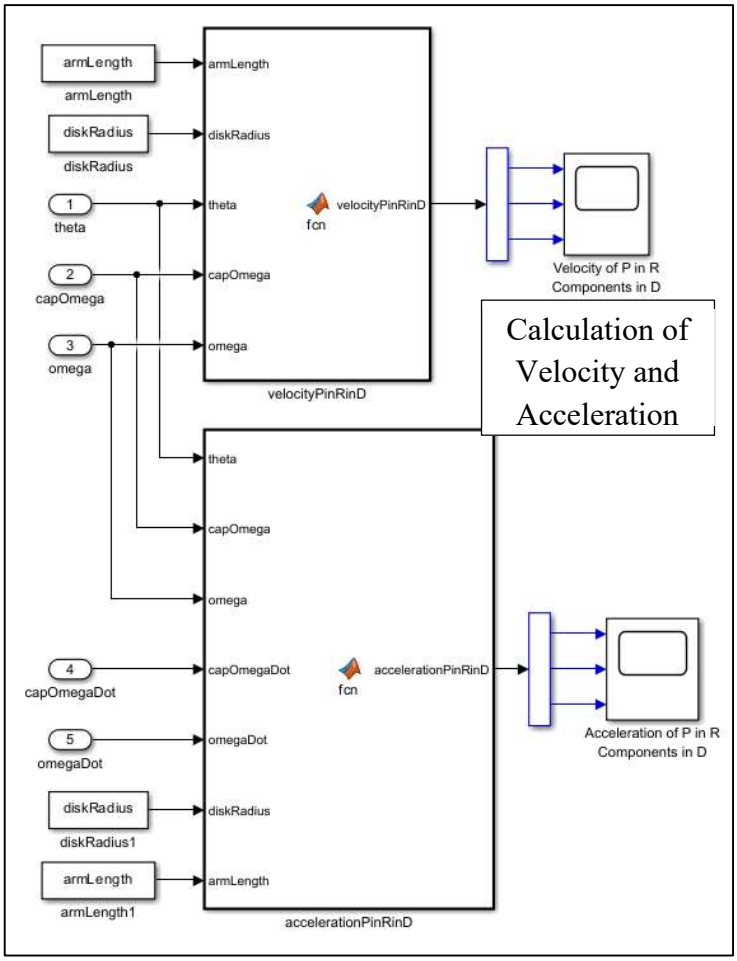
Simulink Model – Panel 3 (applied motion subsystem)



Simulink Model – Panel 4 (subsystems to find angular velocity, angular acceleration, velocity, and acceleration)



Calculation of Angular Velocity and Angular Acceleration



Simulink Model – Panel 5 (function scripts for the angular acceleration and acceleration components)

```
function alphaDinRinD = fcn(theta,capOmega,omega,capOmegaDot,omegaDot)

   alphaDinRinF = zeros(3,1);
   RF2D         = zeros(3,3);

%  Calculate the transformation matrix from frame F to frame D
   sTheta = sin(theta); cTheta = cos(theta);
   RF2D(1,1) = cTheta; RF2D(1,3) = -sTheta;
   RF2D(2,2) = 1.0;
   RF2D(3,1) = sTheta; RF2D(3,3) = cTheta;

%  Calculate the F frame components of alpha D in R
   alphaDinRinF(1) = -omega*capOmega;
   alphaDinRinF(2) = omegaDot;
   alphaDinRinF(3) = capOmegaDot;

%  Transform the F frame components into D frame components
   alphaDinRinD = RF2D*alphaDinRinF;

end
```

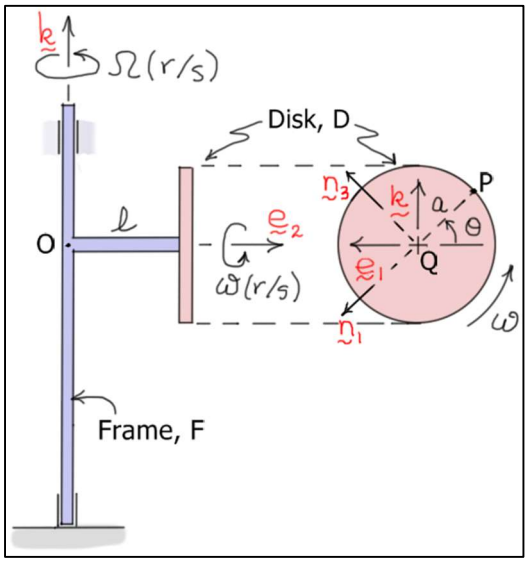
```
function accelerationPinRinD = fcn(theta,capOmega,omega,capOmegaDot,...
                                   omegaDot,diskRadius,armLength)
   accelerationPinRinF = zeros(3,1);
   RF2D                = zeros(3,3);

%  Calculate the transformation matrix from frame F to frame D
   sTheta = sin(theta); cTheta = cos(theta);
   RF2D(1,1) = cTheta; RF2D(1,3) = -sTheta;
   RF2D(2,2) = 1.0;
   RF2D(3,1) = sTheta; RF2D(3,3) = cTheta;

%  Calculate the F frame components of the acceleration of P in R
   accelerationPinRinF(1) = (diskRadius*omegaDot*sTheta) - ...
                            (armLength*capOmegaDot) + ...
                            (diskRadius*cTheta*((omega^2) + (capOmega^2)));
   accelerationPinRinF(2) = (-diskRadius*cTheta*capOmegaDot) + ...
                            (2*diskRadius*sTheta*omega*capOmega) - ...
                            (armLength*(capOmega^2));
   accelerationPinRinF(3) = (diskRadius*cTheta*omegaDot) - ...
                            (diskRadius*sTheta*(omega^2));

%  Transform the F frame components into the D frame components
   accelerationPinRinD = RF2D*accelerationPinRinF;

end
```

Simulink Model – Results:

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*** represented in the model.

Example 4: Simscape Multibody Modeling

In this example, the system of Example 2 is modeled again, this time using a ***MATLAB script*** in conjunction with a ***Simscape Multibody model***. As before, Frame *F* is driven relative to the ground and Disk *D* is driven relative to the column at ***constant relative 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***. Panel 1 shows the ***first section*** of the script which provides a ***functional description*** of the model and initializes ***variables*** it uses.

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, Frame *F* is treated as three bodies ***welded together*** (to form a single body) – a wide ***base***, a ***vertical column***, and a ***horizontal arm***. The ***base***, ***column***, ***arm***, and ***disk*** are all assumed to be ***right circular cylinders***. The ***disk*** is assumed to be ***thin***.

The script first defines the ***time duration*** ("`timeMax`") and ***step size*** ("`timeIncrement`") to be used in the ***fixed-step*** numerical analysis. These variables are used to set ***model parameters*** (model settings) under the Modeling tab. It also defines the ***masses***, ***lengths***, ***radii***, and ***moments of inertia*** of the base, column, and horizontal arm, and it defines the ***mass***, ***radius***, and ***moments of inertia*** for the thin disk. Finally, it defines the ***angular motion data*** for Frame *F* relative to ground and Disk *D* relative to the arm.

Note that values of mass and moments of 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 are considered in Volume II of this text.

Panels 2 and 3 show the ***last three sections*** of the script. The first section ***executes*** the Simscape Multibody model, the second ***plots*** results for the vectors ${}^R\underset{\sim}{\omega}_D$, ${}^R\underset{\sim}{\alpha}_D$, ${}^R\underset{\sim}{v}_P$, 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 Simscape Multibody model are ***structured arrays*** which include time signals. For example, the time values for ${}^R\underset{\sim}{\omega}_D$ are stored in "`diskAngularVelocity.time`", and the vector components of ${}^R\underset{\sim}{\omega}_D$ are stored in "`diskAngularVelocity.signals.values`". See the syntax in Panel 2 below. A similar naming convention is used for the other vectors. As noted earlier, these data are stored using Simulink scopes.

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

```
%% Unit 10 - Example 4: Script for the Simscape Multibody model:
%                       Unit10Example04TwoBodyKinematicsSimscape
%
%  This script sets values for the Simscape Multibody model of Unit 10,
%  Example 4 - a two-body system with applied motions, and then it 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.

   clear variables;

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

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

   IxxColumn = columnMass*((3*(columnRadius^2))+(columnLength^2))/12;   % kg-m^2
   IyyColumn = IxxColumn;   % kg-m^2
   IzzColumn = 0.5*columnMass*(columnRadius^2);   % kg-m^2

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

   IxxArm = armMass*((3*(armRadius^2))+(armLength^2))/12;   % kg-m^2
   IzzArm = IxxArm;   % kg-m^2
   IyyArm = 0.05*armMass*(armRadius^2);   % kg-m^2

%  Disk Data
   diskMass   = 1.0;     %kg
   diskRadius = 0.25;    %meters

   IxxDisk = 0.25*diskMass*(diskRadius^2);   % kg-m^2
   IzzDisk = IxxDisk;   % kg-m^2
   IyyDisk = 0.50*diskMass*(diskRadius^2);   % 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)
                                          ⋮
```

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

```
                                                    ⋮
%%  Run the Simscape Multibody Model

    sim('Unit10Example04TwoBodyKinematicsSimscape.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 on; xlabel('Time (sec)'); ylabel('X-Component (rad/s)')

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

    subplot(3,1,3); plot(diskAngularVelocity.time(:),diskAngularVelocity.signals(3).values(:));
    grid on; xlabel('Time (sec)'); ylabel('Z-Component (rad/s)')
    sgtitle('Body-Fixed Angular Velocity Components of Disk, D (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 on; xlabel('Time (sec)'); ylabel('X-Component (rad/s^2)')

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

    subplot(3,1,3); plot(diskAngularAcceleration.time(:),diskAngularAcceleration.signals(3).values(:));
    grid on; xlabel('Time (sec)'); ylabel('Z-Component (rad/s^2)')
    sgtitle('Body-Fixed Angular Acceleration Components of Disk, D (rad/s^2)');
%   Velocity Components of Point P (m/s)
    figure(3); clf;
    subplot(3,1,1); plot(velocityPointP.time(:),velocityPointP.signals(1).values(:));
    grid on; xlabel('Time (sec)'); ylabel('X-Component (m/s)')

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

    subplot(3,1,3); plot(velocityPointP.time(:),velocityPointP.signals(3).values(:));
    grid on; xlabel('Time (sec)'); ylabel('Z-Component (m/s)')
    sgtitle('Body-Fixed Components of the Velocity of Point P (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 on; xlabel('Time (sec)'); ylabel('X-Component (m/s^2)')

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

    subplot(3,1,3); plot(accelerationPointP.time(:),accelerationPointP.signals(3).values(:));
    grid on; xlabel('Time (sec)'); ylabel('Z-Component (m/s^2)')
    sgtitle('Body-Fixed Components of the Acceleration of Point P (m/s^2)');
                                                    ⋮
```

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

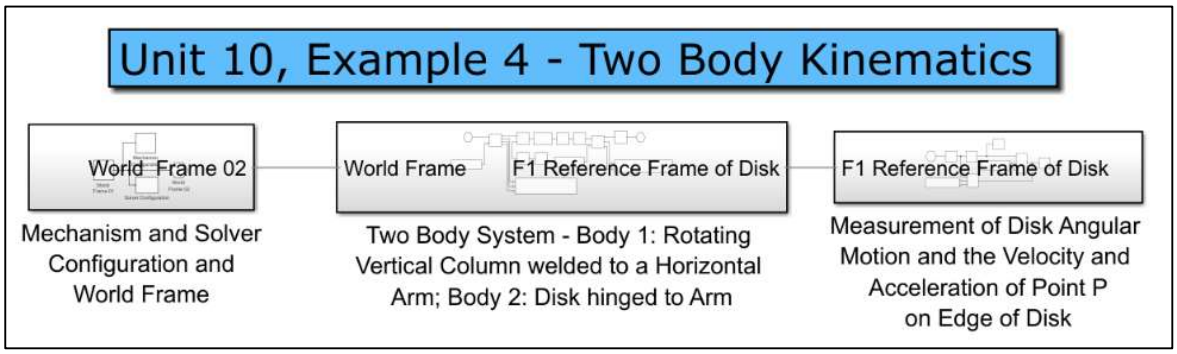
```
                                                    ⋮
%%  Initial angular velocity and angular acceleration of Disk D
%   Initial velocity and acceleration of point A

    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))
```

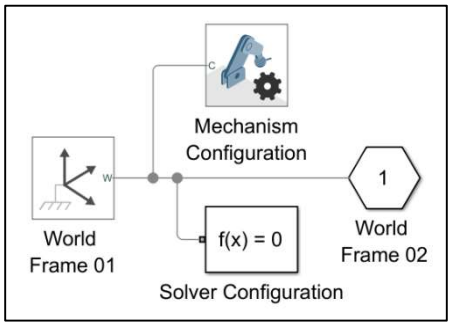
Simscape Multibody Model:

The ***Simulink model*** presented in Example 3 focused on ***calculations*** based on a set of previously derived model equations. The ***block diagram*** represents the ***numerical process*** used to solve the model equations. In contrast, a ***Simscape Multibody block diagram model*** focuses on the ***physical system***. Simscape then ***formulates*** and ***solves*** the model equations associated with the block diagram. ***Interconnecting joints*** in the model can be used to ***actuate*** or ***sense*** motion at that joint. ***Actuation signals*** can be ***generated*** in Simulink and sent to Simscape, and Simscape ***sensor signals*** can be sent to Simulink using ***signal converters***. ***Motions*** of ***bodies*** or ***points*** within the system can be measured using ***transform sensors*** to measure the ***relative motion*** between reference frames defined within the model.

The ***top layer*** of the Simscape Multibody model of the two-body system of this example is shown in Panel 1 below. It contains ***three major subsystems***. These subsystems ***define parameters*** for the model, ***build*** the physical model, ***apply*** the specified motions, and ***measure*** the desired results. The results are ***stored*** in the MATLAB workspace for plotting by the MATLAB script.

Simscape Multibody Model – Panel 1: (top layer of the two-body model)



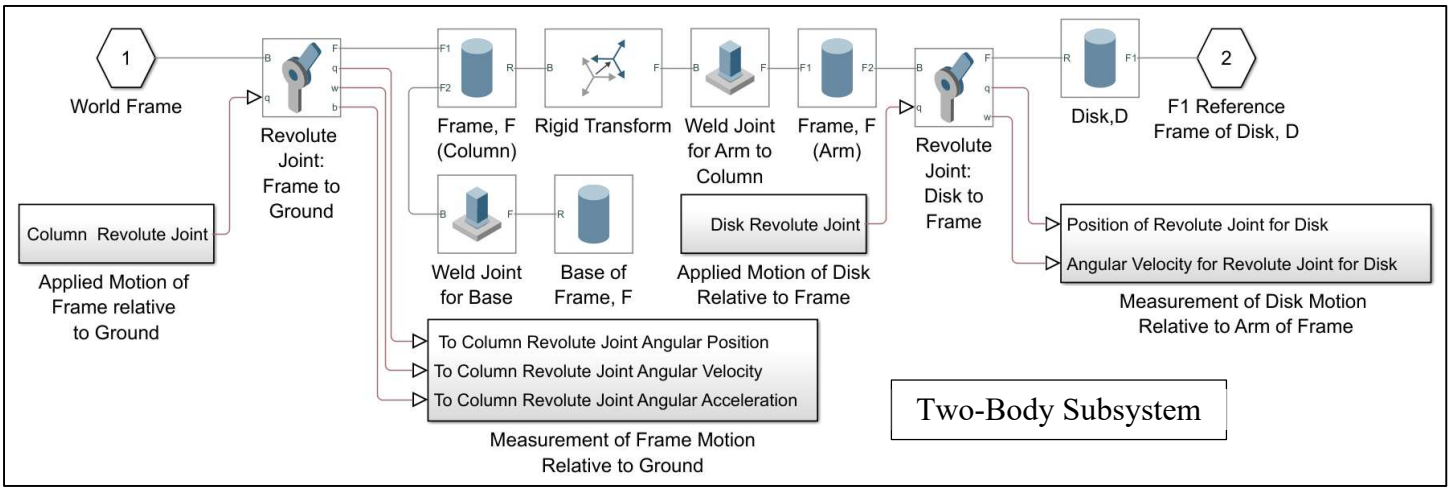


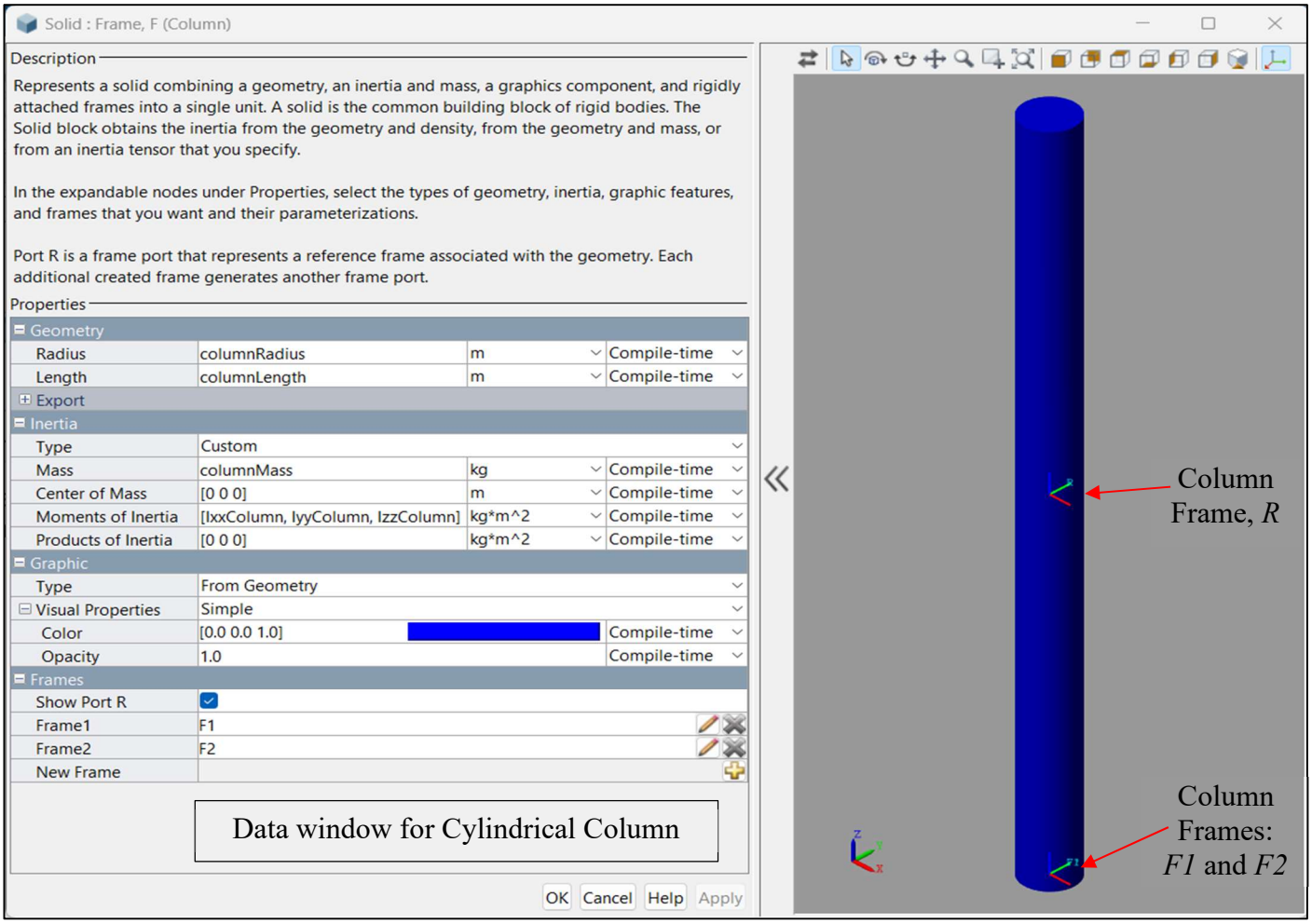
The subsystem on the ***left*** (in Panel 1) contains blocks for the ***World Frame***, ***Mechanism Configuration***, and ***Solver Configuration***. The ***World Frame*** (or ground frame) is a fixed, orthogonal, right-handed coordinate frame in which the physical model is constructed. All world frame blocks in the model refer to the same frame. The ***Mechanism Configuration*** block can be used to set the local acceleration of gravity and its direction in the world frame. This setting applies to all the bodies in the model. The ***Solver Configuration*** block is ***required*** for all models, and it can be used to ***set tolerances*** that determine how ***accurately*** model constraints are maintained during the simulation of the model. For example, some points of a physical system should ***ideally coincide*** throughout its motion; however, in the simulation of the motion using a Simscape model, the distance between these points must simply be ***less than*** the set ***tolerance***. ***Smaller tolerances*** give ***more accurate results*** but will require ***longer execution times***.

Details of the ***two-body subsystem*** (in the center of Panel 1) are shown in Panel 2 below. This subsystem ***creates*** the two-body physical model (Frame *F* and Disk *D*), and it calculates, measures, and stores the applied motions at the two revolute joints. All Simscape Multibody revolute joints are about the local *z*-axis, so the leftmost revolute joint (which connects Frame *F* to the world frame) allows *F* to rotate about the world frame's *z*-axis.

Frame *F* consists of three parts – a wide cylindrical base, a cylindrical column, and a cylindrical arm. All three components of *F* are connected in the model with ***Weld joints***. Three coordinate frames are defined for the cylindrical column. Column coordinate frames *F1* and *F2* are both located at the ***bottom*** of the column, and coordinate frame *R* is located at its ***mass (geometric) center***. All the coordinate frames are initially ***aligned*** with the world frame. A revolute joint connects column frame *F1* to the world frame, and a Weld joint connects column frame *F2* to the mass center frame *R* of the cylindrical base.

Simscape Multibody Model – Panel 2: (the two-body subsystem)



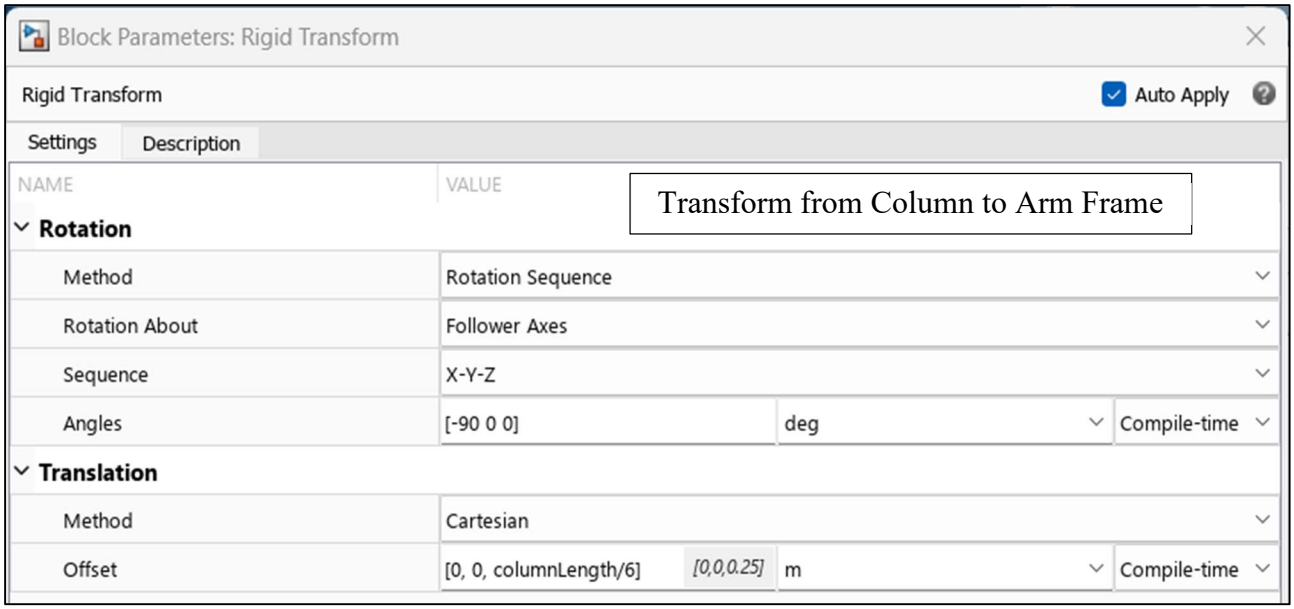


The parameter window for the cylindrical column is also shown in Panel 2 above. The radius, length, mass, moments of inertia, products of inertia, color, and reference frames of the column are all set using this window. Note the parameter values are set using MATLAB ***workspace variables*** that are defined in the associated script described above. ***Similar windows*** are used to set the ***geometric*** and ***physical parameters*** associated with the

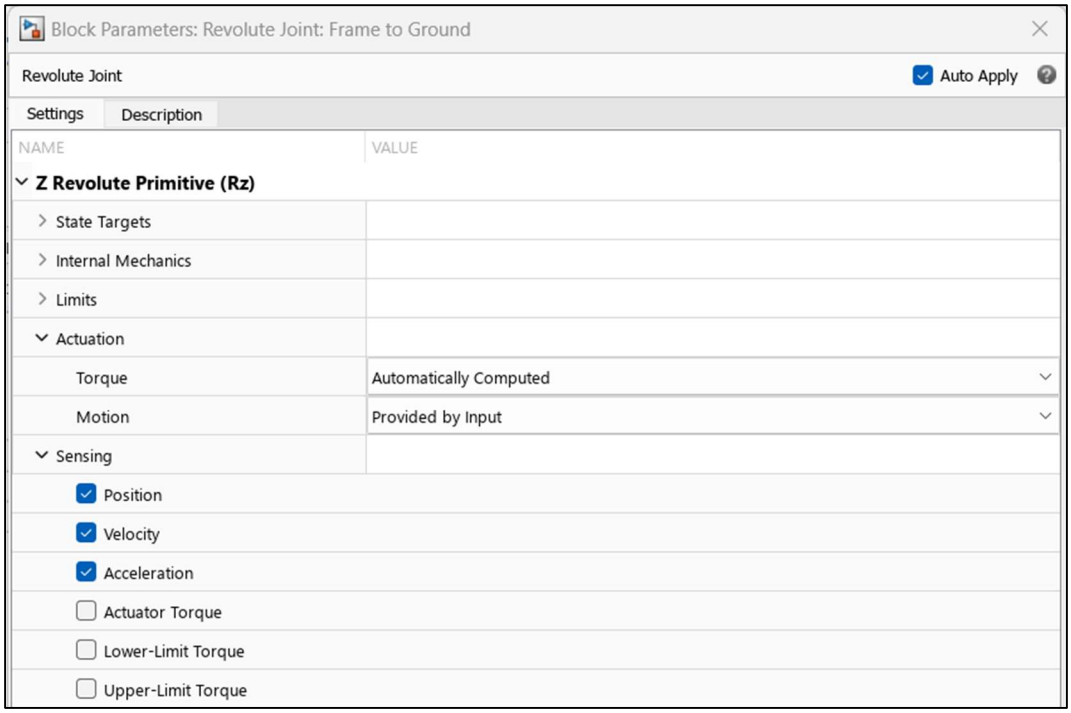
cylindrical base, cylindrical arm, and the disk. Note for all cylindrical bodies the axis of the cylinder is aligned with the local *z*-axis.

The horizontal cylindrical arm is connected to the vertical column using a Weld joint two-thirds of the length up the column. To accomplish this, a new coordinate frame is created whose ***origin*** is ***located*** one-sixth of the column length above its mass (geometric) center with its *z*-axis pointed ***horizontally***. This is accomplished using a ***Rigid Transform*** block connected to the column's mass center coordinate frame (*R*). A translation of one-sixth the column length (from the mass center) in the direction of the column's *z*-axis is followed by a –90 (deg) rotation about its *x*-axis. See the ***Block Parameters*** window in the panel below.

The output of the Rigid Transform block is the resulting coordinate frame *F* which is connected using a Weld joint to the arm frame *F1* at the left end of the arm. The parameter window for the arm is used to create another coordinate frame *F2* at the right end of the arm. Finally, a revolute joint is used to connect arm frame *F2* to the mass center frame (*R*) of the disk. The *z*-axes of these frames are both pointed horizontally outward along the arm. The disk frame *F1* is coincident with its mass center frame *R*.

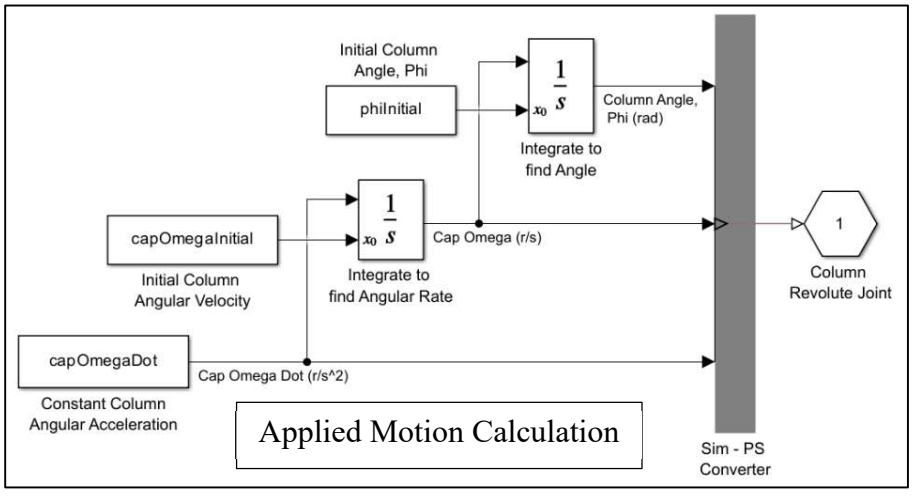


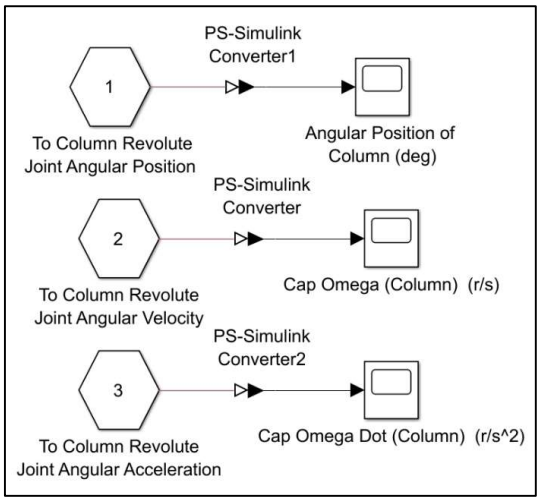
Both ***revolute joints*** have ***applied motion*** that is calculated and supplied to the joint. These conditions are set in the ***Block Parameters*** windows for the joints. The Block Parameter window for the revolute joint connection to the ground is shown in the panel below. Note the joint motion is labeled as "`Provided by Input`", and the joint torque is labeled as "`Automatically Computed`". If the motion of the joint is free with an applied torque, these two labels would be reversed. Also, note the Position (angle), Velocity (relative angular velocity), and Acceleration (relative angular acceleration) of the joint are being sensed (measured).



The subsystems that ***calculate*** and ***store*** the motion of the revolute joint that connects Frame *F* to the ground are shown in the panel below. The panel on the left shows the subsystem that calculates the applied angular acceleration, angular velocity, and angle of Frame *F* assuming constant angular acceleration. The constant angular acceleration and the initial values of the angular velocity and angle are set using MATLAB workspace variables in the associated script. The signals are first generated in Simulink and then ***converted*** to Simscape physical signals (PS) using a Sim-PS Converter. The block on the right shows the measurement and storage subsystem that converts the physical signals (PS) to Simulink signals and sends those signals to ***scopes*** for plotting and storage in the workspace.

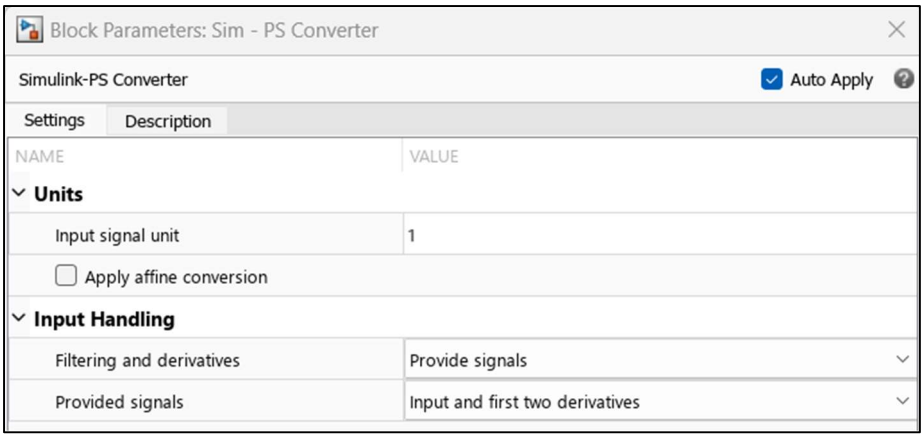
Simscape Multibody Model – Panel 3: (calculation, measurement, and storage of applied motion)





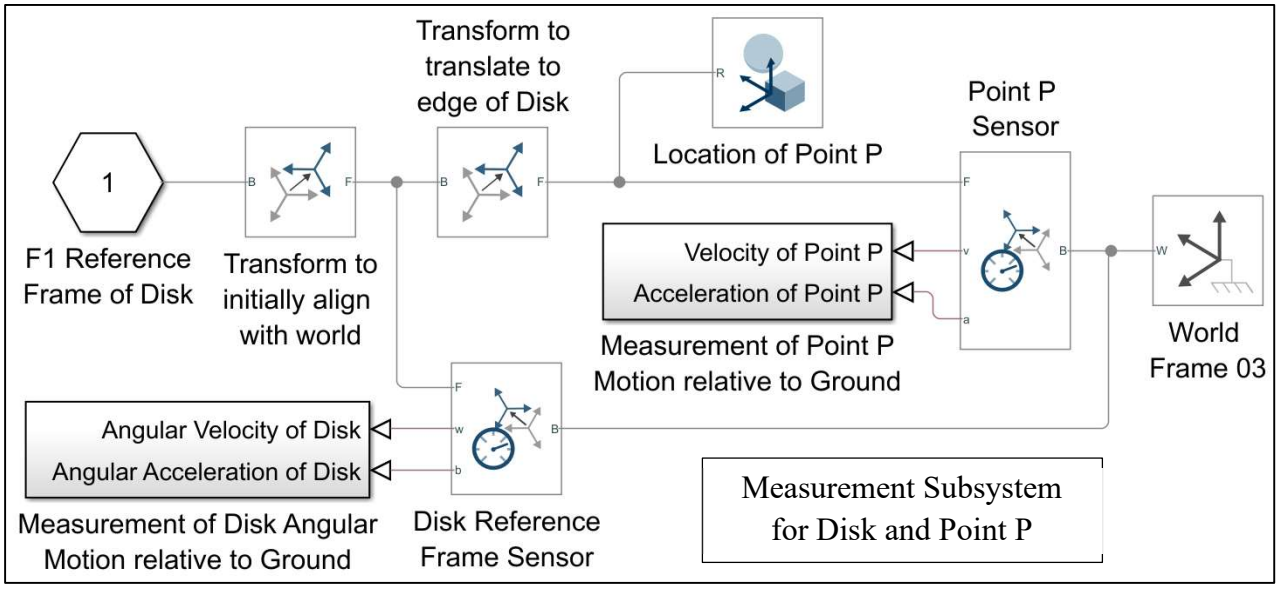
Joint motion can be defined using a single signal, the signal and its first derivative, or the signal and its first two derivatives. This is set in the Block Parameters window for the Sim-PS Converter. In this example, the ***signal*** and its ***first two derivatives*** are supplied, so the Sim-PS Converter accepts three signals as input. The output is a

single physical signal (PS) sent to the joint. See the Block Parameters window for the Sim-PS Converter below. The calculation and data storage subsystems for the revolute joint between the Frame *F* and Disk *D* have identical structure as those described above and are not presented here.



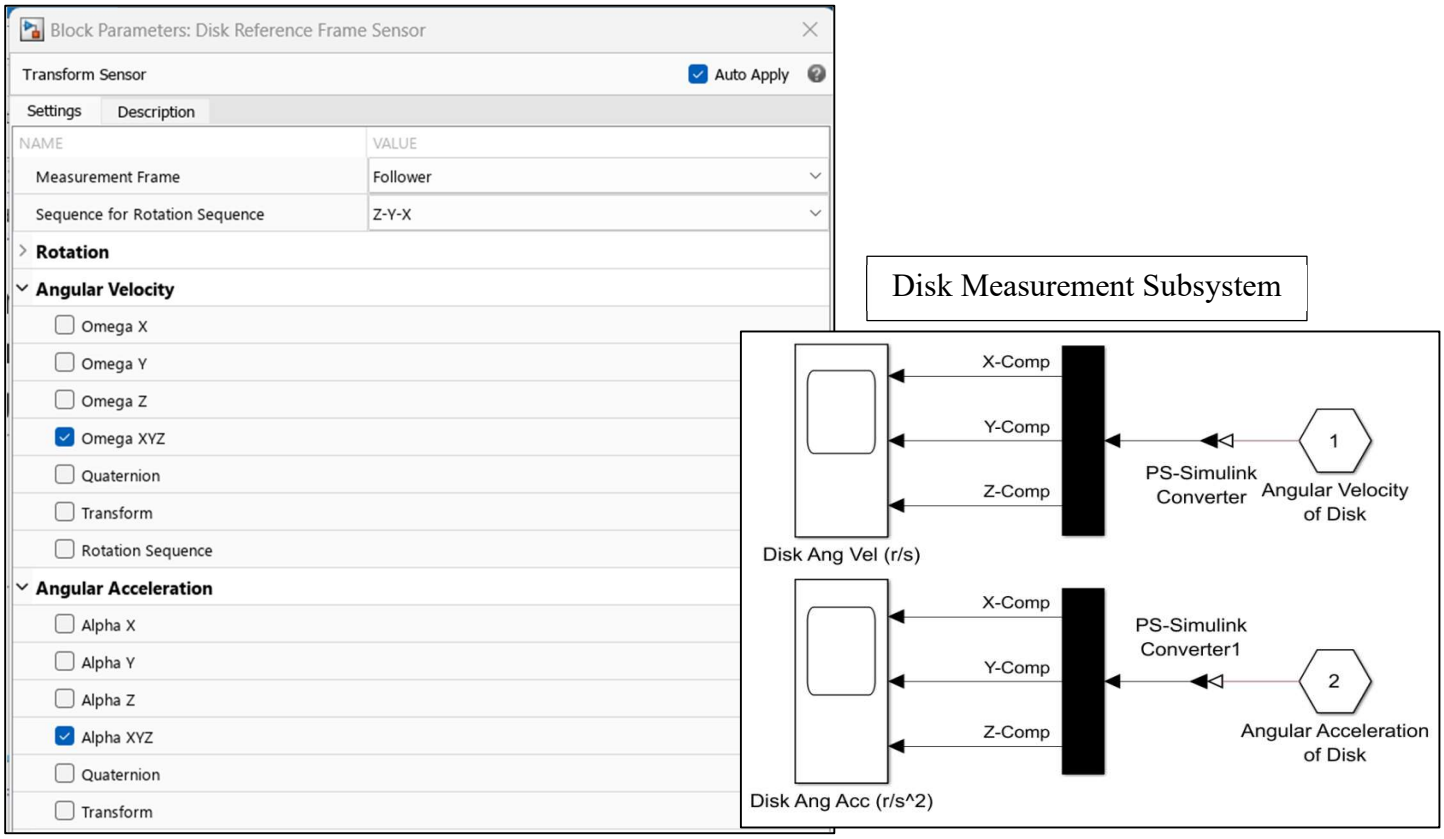
The third major subsystem is shown in Panel 4 below. This subsystem ***measures*** the ***angular*** velocity and ***angular acceleration*** of Disk *D* and locates point *P* and measures its ***velocity*** and ***acceleration***. The first Rigid Transform ***creates*** a reference frame *F* (from disk frame *F1*) that is fixed in the disk and is ***initially aligned*** with the World Frame. This is accomplished with a +90 (deg) rotation about the *x*-axis of disk frame *F1*.

Simscape Multibody Model – Panel 4: (calculation, measurement, and storage of calculated motions)

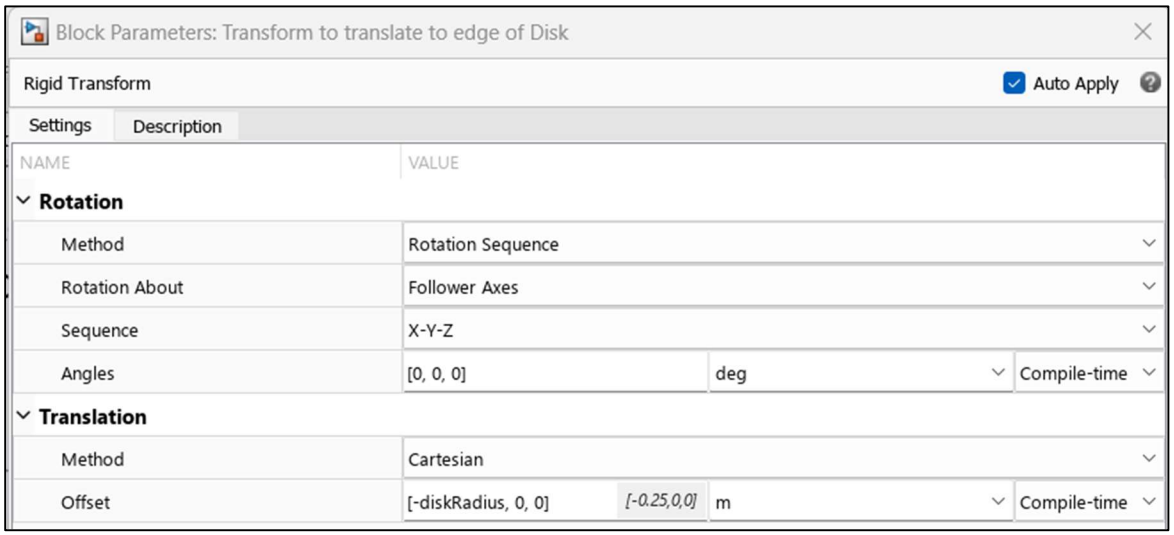


A ***Transform Sensor*** is used to ***measure*** the ***angular velocity*** and ***angular acceleration*** of this frame (fixed in Disk *D*) relative to the world frame. The ***Block Parameters*** window for the disk reference frame sensor is shown below. The measurement frame is listed as the "`Follower`" ***frame***, so the vector components are resolved in the ***disk frame***. The "`Omega XYZ`" and "`Alpha XYZ`" boxes are checked, so the angular velocity and angular acceleration vectors are both output as ***single signals***. The subsystem that records these signals is also shown below. PS-Simulink Converter blocks are used to convert physical signals (PS) to Simulink signals. Then Demux

blocks are used to separate the vector signals into their individual components. These signals are sent to Simulink scopes for plotting and data storage in the workspace.



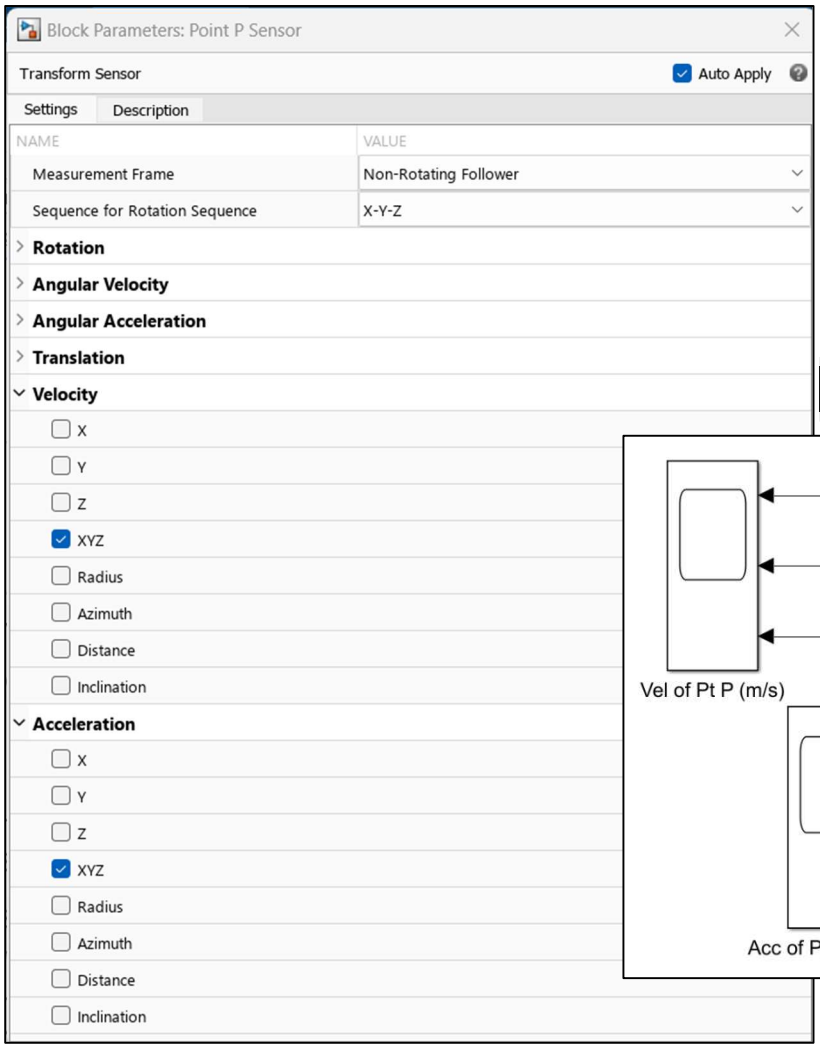
To measure the ***velocity*** and ***acceleration*** of point *P*, a new ***reference frame*** is created at the ***edge*** of the ***disk***. This is accomplished using a ***Rigid Transform*** block with an ***origin translation*** along the negative local *x*-axis. The length of the translation is equal to the radius of the disk. The Block Parameters window for this transform is shown below.



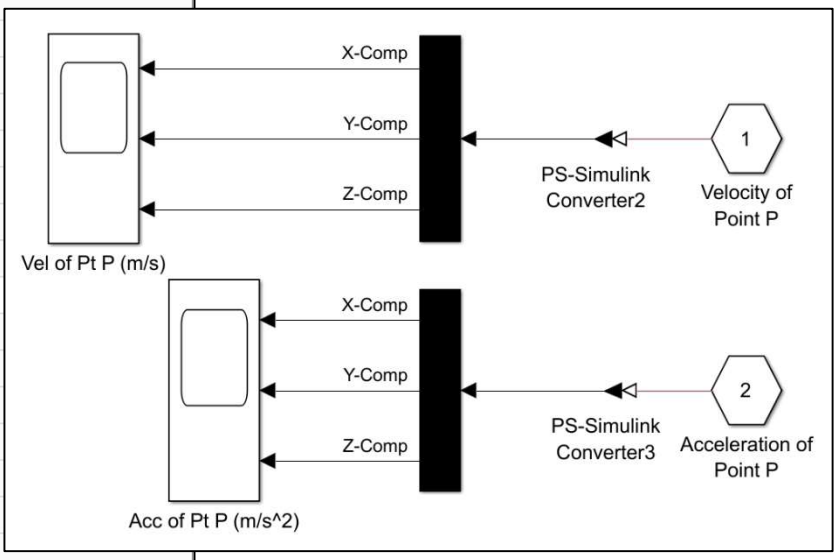
A Transform Sensor is used to ***measure*** the ***velocity*** and ***acceleration*** of the origin of this new reference frame which is located at point *P*. The Block Parameters window for this sensor is shown below. As indicated, the sensor

will measure the "XYZ" components of the velocity and acceleration, and it will output them as vector signals. As in the Disk Measurement Subsystem, the physical signals (PS) are converted to Simulink signals, separated using Demux blocks, and sent to scopes for plotting and storage in the workspace.

Special Note: The Measurement Frame in the Block Parameters window for the sensor lists the frame as "`Non-Rotating Follower`". This is required to get the ***correct disk-frame components*** of the vectors. The labelling here is confusing because the disk frame is a rotating frame. The drop-down window lists the "`Follower`" frame and the "`Non-Rotating Follower`" frame as separate frames. It is not clear what the distinction is between these two frames.



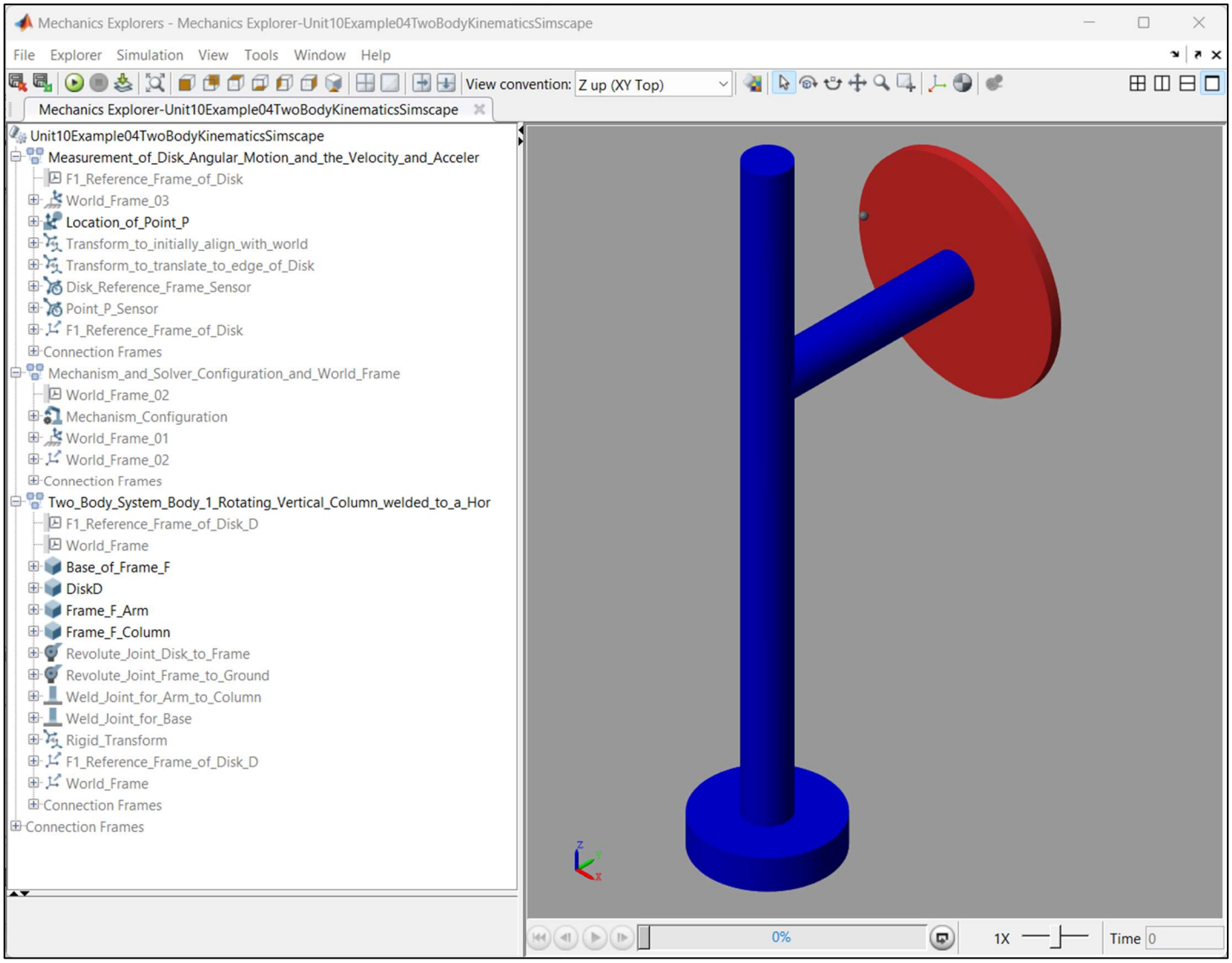
Point *P* Measurement Subsystem



Model Execution:

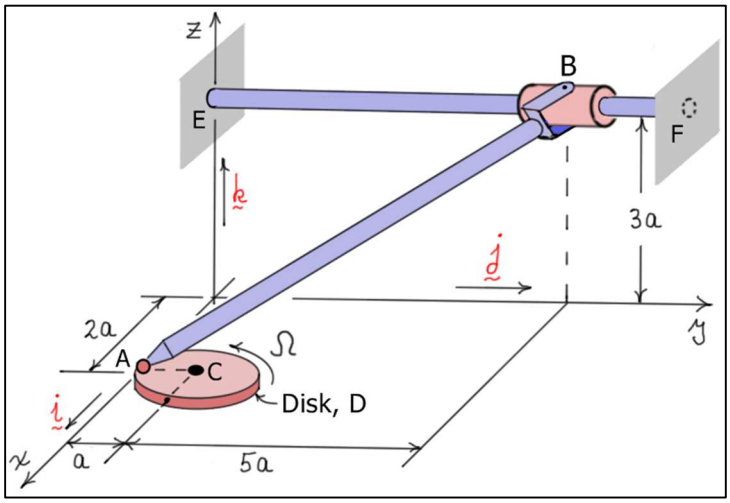
When the MATLAB script is executed, all the model parameters are defined, the Simscape model is executed, run data is stored in the MATLAB workspace, and the results are plotted. In this process, the ***Mechanics Explorer*** and animation window opens. This window has a ***model explorer*** on the left side and an ***animation window*** on the right. See the figure below. The model explorer can be used to ***verify*** the ***model connectivity***. The animation

window displays the motion of the system as the model runs. A ***small graphical sphere*** has been added for easy ***tracking*** of point *P* as the system moves. This sphere has no mass or inertia.



Example 5: Simscape Multibody Modeling of a Closed Loop Mechanism

The figure shows the three-dimensional slider-crank mechanism that was analyzed in Unit 8. Disk *D* (the crank) 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* (the follower) is attached to the disk with a spherical (ball-in-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 collar at *B* is assumed ***negligible***.



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

As in Unit 8, the ***motion*** of ***Disk D*** is ***specified***, and the ***motions*** of ***bar AB*** and the ***collar*** at *B* are ***calculated***. In this case, a ***MATLAB script*** is used to ***execute*** a Simscape Multibody model that ***computes*** the ***motions*** of the ***bar*** and the ***collar***. A second script is used to solve the ***analytical equations*** from Unit 8 to compute analytical results. The results of the two methods are ***identical***.

The ***first four sections*** of the MATLAB script are shown in Panel 1 below. These sections define MATLAB workspace variables for the simulation time scale, the crank motion, the geometric dimensions of the mechanism, and a transformation matrix that defines the initial orientation of bar *AB*. These variables are all required to build and execute the Simscape Multibody model. It also calculates the "build length" for the connecting rod and prints that length and the required length in the Command window for comparison and verification.

The variables "`timeFinal`" and "`timeIncrement`" define values for the ***final time*** and the ***fixed time increment*** for the model calculations. Variables "`crankInitialAngularVelocity`" and "`crankInitialAngularAcceleration`" define values for the initial angular velocity and angular acceleration of Disk *D*. The geometric data are all calculated in terms of the length "*a*". The radius of *D* is equal to "*a*", and the operational length of bar *AB* is "7*a*". The thickness of *D* is assumed to be one-tenth of its radius.

Bar *AB* (the connecting rod) consists of three major components – a spherical end, a cylindrical central section, and a U-shaped end with a cylindrical pin. The working length of *AB* is equal to the distance from the spherical joint to the pin connecting it with the collar at *B*. In the Simscape model this length (the build length) is the sum of the length of the central cylindrical body, the bottom plate thickness, and the length of the lower side plate. The bottom and side plates are portions of the U-shaped end that holds the cylindrical pin for connection at *B*. The build length is calculated by the script and sent to the Command window for comparison with the required length of "7*a*". If these lengths are ***not*** the ***same***, the Simscape program will ***not execute***.

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

```
%% Volume I, Unit 10, 3D Slider-Crank Mechanism, Example 5 in Simscape

%  This script first defines the simulation time scale, crank motion data,
%  geometric parameters associated with the 3D slider-crank mechanism, and
%  the transformation matrix required to orient the connecting rod in the
%  initial configuration. It then runs the Simscape model and plots the
%  results. It also calculates the connecting rod length and prints it to
%  the command window for verification.
%
   clear variables

%% Time Data
   timeFinal     = 2.0;    % (sec)
   timeIncrement = 0.01;   % (sec)

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

%% Geometric data
   a                    = 0.1;                          % (meters)
   diskRadius           = a;                            % (meters)
   diskThickness        = 0.1*a;                        % (meters)
   connectingRodLength  = 7*a;                          % (meters)
   cylinderLength       = 0.84*connectingRodLength;     % (meters)
   cylinderRadius       = 0.015*cylinderLength;         % (meters)
   sphereRadius         = 1.5*diskThickness;            % (meters)
   bottomPlateLength    = 8*cylinderRadius;             % (meters)
   bottomPlateDepth     = cylinderRadius;               % (meters)
   bottomPlateThickness = 0.01*connectingRodLength;     % (meters)
   sidePlateLength      = 0.15*connectingRodLength;     % (meters)
   sidePlateDepth       = bottomPlateDepth;             % (meters)
   sidePlateThickness   = bottomPlateThickness;         % (meters)

%  Calculate the "build length" and compare it to the required length
   disp('Required Connecting Rod Length')
   disp(' ')
   disp(connectingRodLength);
   buildLength = cylinderLength + bottomPlateThickness + ...
                 sidePlateLength;
   disp('Total Composite Body (build) Length')
   disp(' ')
   disp(buildLength);

%  Transformation Matrix for Initial Connecting Rod Orientation
   connectingRodTransformationMatrix = zeros(3,3);
   sqrt13 = sqrt(13);  sevenSqrt13 = 7*sqrt13;

   connectingRodTransformationMatrix(1,1) = 3/sqrt13;
   connectingRodTransformationMatrix(1,2) = 0;
   connectingRodTransformationMatrix(1,3) = 2/sqrt13;

   connectingRodTransformationMatrix(2,1) =  12/sevenSqrt13;
   connectingRodTransformationMatrix(2,2) =  13/sevenSqrt13;
   connectingRodTransformationMatrix(2,3) = -18/sevenSqrt13;

   connectingRodTransformationMatrix(3,1) = -2/7;
   connectingRodTransformationMatrix(3,2) =  6/7;
   connectingRodTransformationMatrix(3,3) =  3/7;

   connectingRodTransformationMatrix = connectingRodTransformationMatrix';
                                       ⋮
```

Finally, the script calculates a ***coordinate transformation matrix*** used by the Simscape model to orient bar *AB* in its initial position. *AB* is built within the Simscape model in a local *xz* plane. The U-shaped end would appear in edge view in this plane, and the local *y* axis is normal to this plane. To orient *AB* in its ***initial position*** relative to the world frame axes, the following must be true.

1. The local *x* axis of *AB* is normal to plane *AEB*. A unit vector $\underset{\sim}{n}_x$ normal to plane *AEB* can then be calculated as follows.

$$\underset{\sim}{n}_x = \frac{\underset{\sim}{j}\times\underset{\sim}{r}_{B/A}}{\left|\underset{\sim}{j}\times\underset{\sim}{r}_{B/A}\right|} = \frac{\underset{\sim}{j}\times a\left(-2\,\underset{\sim}{i}+6\,\underset{\sim}{j}+3\,\underset{\sim}{k}\right)}{\left|\underset{\sim}{j}\times\underset{\sim}{r}_{B/A}\right|} = \frac{a\left(3\,\underset{\sim}{i}+2\,\underset{\sim}{k}\right)}{\left|\underset{\sim}{j}\times\underset{\sim}{r}_{B/A}\right|} = \frac{a\left(3\,\underset{\sim}{i}+2\,\underset{\sim}{k}\right)}{\sqrt{9a^2+4a^2}} = \frac{\left(3\,\underset{\sim}{i}+2\,\underset{\sim}{k}\right)}{\sqrt{13}}$$

2. The local *z* axis points away from the spherical joint at *A*. A unit vector $\underset{\sim}{n}_z$ along in this direction can then be calculated as follows.

$$\underset{\sim}{n}_z = \left(-2a\,\underset{\sim}{i}+6a\,\underset{\sim}{j}+3a\,\underset{\sim}{k}\right)\Big/7a = -\left(\tfrac{2}{7}\right)\underset{\sim}{i}+\left(\tfrac{6}{7}\right)\underset{\sim}{j}+\left(\tfrac{3}{7}\right)\underset{\sim}{k}$$

3. For a right-handed system, the local *y* axis is in the direction of $\underset{\sim}{n}_z\times\underset{\sim}{n}_x$.

$$\underset{\sim}{n}_y = \underset{\sim}{n}_z\times\underset{\sim}{n}_x = \begin{vmatrix} \underset{\sim}{i} & \underset{\sim}{j} & \underset{\sim}{k} \\ -\frac{2}{7} & \frac{6}{7} & \frac{3}{7} \\ \frac{3}{\sqrt{13}} & 0 & \frac{2}{\sqrt{13}} \end{vmatrix} = \left(\frac{12}{7\sqrt{13}}\right)\underset{\sim}{i}+\left(\frac{9+4}{7\sqrt{13}}\right)\underset{\sim}{j}+\left(\frac{-18}{7\sqrt{13}}\right)\underset{\sim}{k} = \left(12\,\underset{\sim}{i}+13\,\underset{\sim}{j}-18\,\underset{\sim}{k}\right)\Big/7\sqrt{13}$$

These results can be written as a single equation to identify the transformation matrix that relates the world frame and the frame of bar *AB*.

$$\begin{Bmatrix} \underset{\sim}{n}_x \\ \underset{\sim}{n}_y \\ \underset{\sim}{n}_z \end{Bmatrix} = \begin{bmatrix} \frac{3}{\sqrt{13}} & 0 & \frac{2}{\sqrt{13}} \\ \frac{12}{7\sqrt{13}} & \frac{13}{7\sqrt{13}} & \frac{-18}{7\sqrt{13}} \\ \frac{-2}{7} & \frac{6}{7} & \frac{3}{7} \end{bmatrix} \begin{Bmatrix} \underset{\sim}{i} \\ \underset{\sim}{j} \\ \underset{\sim}{k} \end{Bmatrix} \triangleq \left[R_{AB}\right] \begin{Bmatrix} \underset{\sim}{i} \\ \underset{\sim}{j} \\ \underset{\sim}{k} \end{Bmatrix}$$

The transformation matrix $\left[R_{AB}\right]$ allows the ***local directions*** to be written in terms of the ***world directions***. Recall that $\left[R_{AB}\right]$ is an ***orthogonal matrix*** and so the ***transpose*** of $\left[R_{AB}\right]$ allows the ***world directions*** to be written in terms of the ***local directions***. The ***transpose*** of $\left[R_{AB}\right]$ is used in the Simscape Multibody model at compile time to correctly position *AB* within the world frame. See Unit 5 of this volume for more details on coordinate transformation matrices.

The last two sections of the script are shown in Panel 2 below. These sections execute the Simscape model and plot the applied and calculated motions. The data are all stored in arrays – time values are in the first column, and calculated data values are in subsequent columns. The results are plotted in figure windows 1-5.

Script – Panel 2: (execute the Simscape model and plot the stored results)

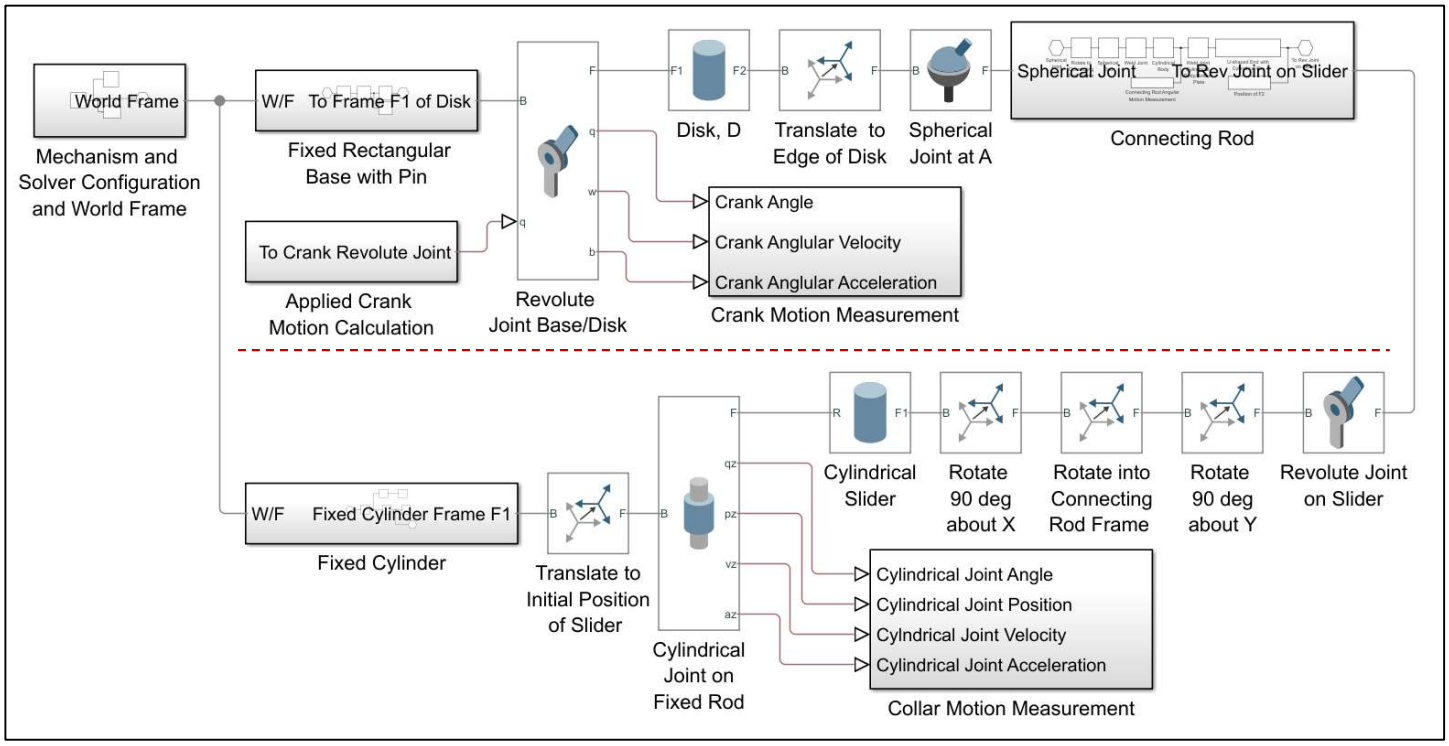
```
                                                    ⋮
%% Run Simscape Model

   sim('Unit10Example05_3DSliderCrankSimscape')

%% Plot results

%  Input Motion at the Crank (Disk)
   figure(1); clf;
   subplot(3,1,1); plot(crankAngle(:,1),(180/pi)*crankAngle(:,2),'b'); grid on;
   xlabel('Time (sec)'); ylabel('Angle (deg)');
   subplot(3,1,2); plot(crankAngularVelocity(:,1),crankAngularVelocity(:,2),'b');
   grid on; xlabel('Time (sec)'); ylabel('Angular Velocity (r/s)');
   subplot(3,1,3); plot(crankAngularAcceleration(:,1),crankAngularAcceleration(:,2),'b');
   grid on; xlabel('Time (sec)'); ylabel('Angular Acceleration (r/s^2)');
   sgtitle('Crank (Disk) Angle, Angular Velocity, and Angular Acceleration')

%  Connecting Rod Angular Velocity Components (Base Frame)
   figure(2); clf;
   subplot(3,1,1); plot(omega_AB_x(:,1),omega_AB_x(:,2),'b'); grid on;
   xlabel('Time (sec)'); ylabel('Omega(AB)_x (r/s)');
   subplot(3,1,2); plot(omega_AB_y(:,1),omega_AB_y(:,2),'b'); grid on;
   xlabel('Time (sec)'); ylabel('Omega(AB)_y (r/s)');
   subplot(3,1,3); plot(omega_AB_z(:,1),omega_AB_z(:,2),'b'); grid on;
   xlabel('Time (sec)'); ylabel('Omega(AB)_z (r/s)');
   sgtitle('Angular Velocity Components of Bar AB')

%  Connecting Rod Angular Acceleration Components (Base Frame)
   figure(3); clf;
   subplot(3,1,1); plot(alpha_AB_x(:,1),alpha_AB_x(:,2),'b'); grid on;
   xlabel('Time (sec)'); ylabel('Alpha(AB)_x (r/s^2)');
   subplot(3,1,2); plot(alpha_AB_y(:,1),alpha_AB_y(:,2),'b'); grid on;
   xlabel('Time (sec)'); ylabel('Alpha(AB)_y (r/s^2)');
   subplot(3,1,3); plot(alpha_AB_z(:,1),alpha_AB_z(:,2),'b'); grid on;
   xlabel('Time (sec)'); ylabel('Alpha(AB)_z (r/s^2)');
   sgtitle('Angular Acceleration Components of Bar AB')

%  Cylindrical Joint Position and Angle (relative to starting position)
   figure(4); clf;
   subplot(2,1,1); plot(cylindricalJointAngle(:,1),(180/pi)*cylindricalJointAngle(:,2),'b');
   grid on; xlabel('Time (sec)'); ylabel('Angle (deg)');
   subplot(2,1,2); plot(cylindricalJointPosition(:,1),cylindricalJointPosition(:,2),'b');
   grid on; xlabel('Time (sec)'); ylabel('Position (m)');
   sgtitle('Cylindrical Joint Angle and Position (relative to starting position)');


%  Slider Position, Velocity, and Acceleration
   figure(5); clf;
   subplot(3,1,1); plot(sliderPosition(:,1),sliderPosition(:,2),'b'); grid on;
   xlabel('Time (sec)'); ylabel('Position (m)');
   subplot(3,1,2); plot(sliderVelocity(:,1),sliderVelocity(:,2),'b'); grid on;
   xlabel('Time (sec)'); ylabel('Velocity (m/s)');
   subplot(3,1,3); plot(sliderAcceleration(:,1),sliderAcceleration(:,2),'b');
   grid on; xlabel('Time (sec)'); ylabel('Acceleration (m/s^2)');
   sgtitle('Slider Position, Velocity, and Acceleration');
```

Simscape Multibody Model: Top Layer

The ***top layer*** of the Simscape Multibody model is shown in Panel 1 below. As with the previous model, this model contains a subsystem (on the left side of the diagram) that contains blocks for the ***World Frame***, ***Mechanism Configuration***, and ***Solver Configuration***. Recall that the Simscape Multibody model is built within the world frame. Additional frames are defined within the model as the system is constructed.

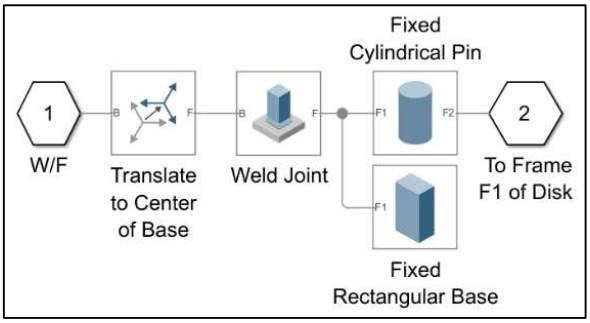
A ***dashed line*** has been added to the diagram to show the model can be viewed as two systems that are connected. The portion ***above*** the dashed line consists of three bodies – a fixed rectangular base (with a pin), a disk (Disk *D*), and a connecting rod (bar *AB*). The fixed rectangular base is connected to Disk *D* with a revolute joint, and *D* is connected to bar *AB* using a spherical joint. The portion ***below*** the dashed line consists of two bodies – a fixed horizontal cylinder and a cylindrical collar. The upper and lower portions are ***connected*** using a revolute joint. Details of the upper and lower portions of the system are provided below.

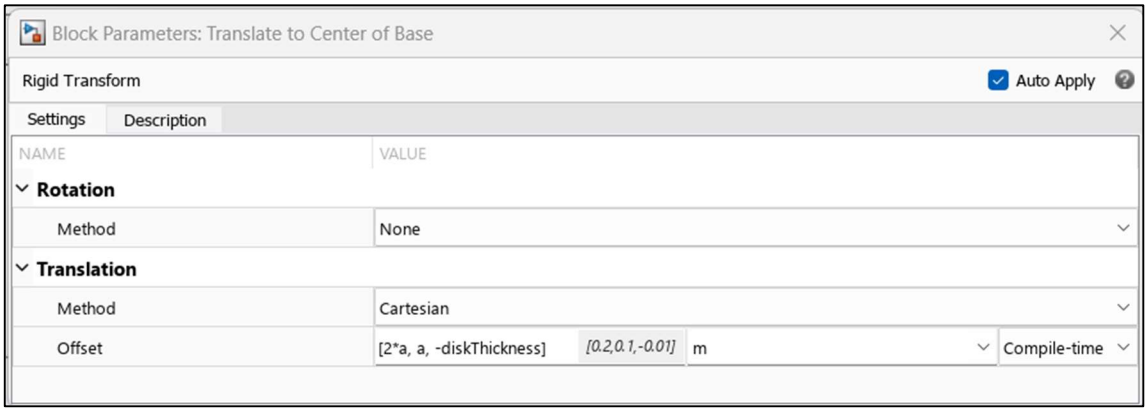
Simscape Multibody Model – Panel 1: (top layer of the slider crank model)



Rectangular Base, Disk *D*, and bar *AB*:

Details of the subsystem for the ***fixed rectangular base*** with ***pin*** are shown in the diagram below. To locate the base, a coordinate system translation is done using a Rigid Transform block to the location $(2a, a, -\text{diskThickness})$ in the world frame. Frame *F1* of the rectangular base which is located at the ***center*** of its ***top surface*** is welded to this point. Frame *F1* of the cylindrical pin which is located at the ***center*** of its ***bottom surface*** is also welded to this point. Frame *F2* of the cylindrical pin is collocated with its *F1* frame.



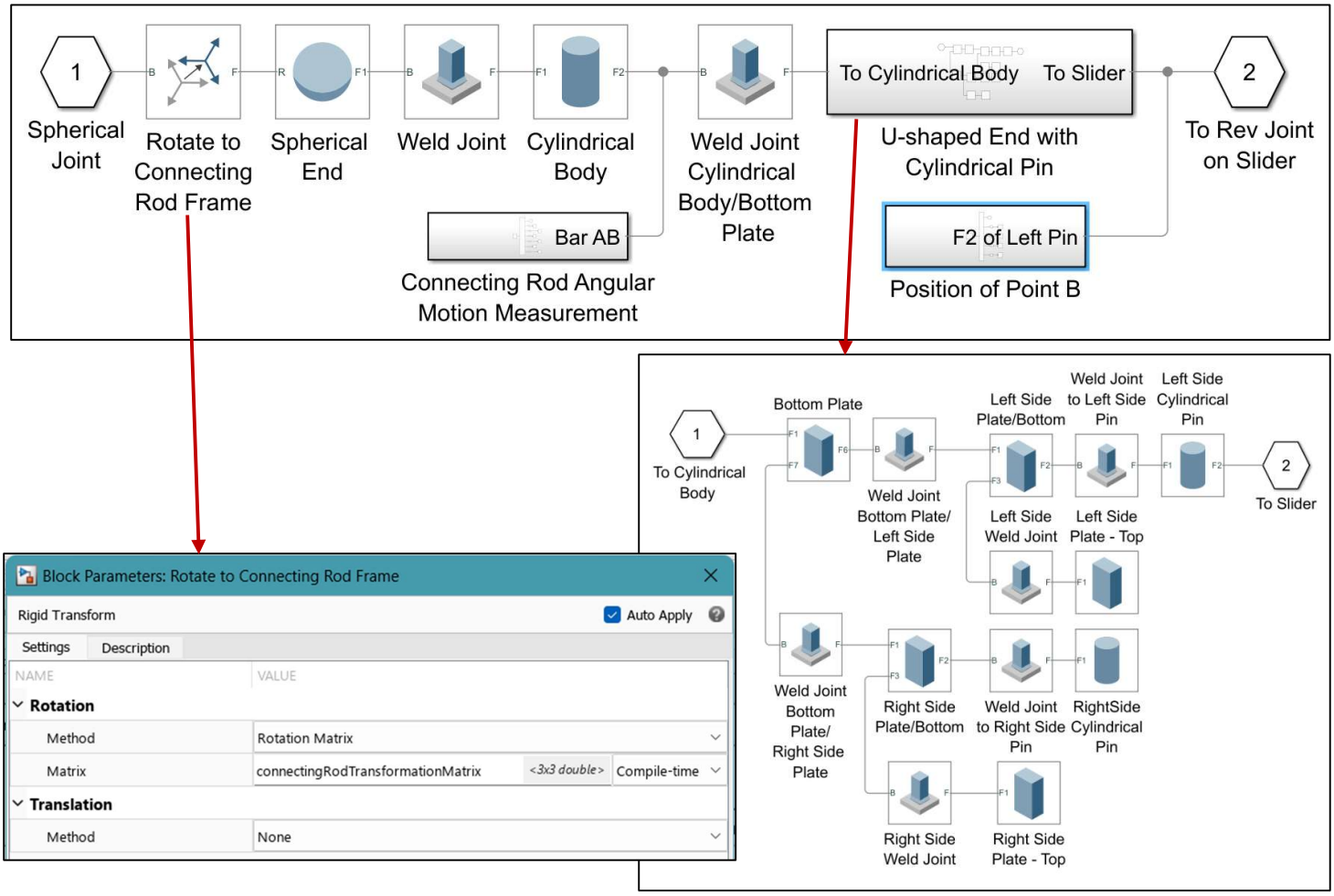


Referring again to Panel 1, frame *F1* of Disk *D* which is located at the ***center*** of its ***bottom surface*** is connected using a ***revolute joint*** to the ***center*** of the ***top surface*** of the fixed rectangular base. This means the ***center*** of the ***top surface*** of *D* has coordinates $(2a, a, 0)$ in the world frame consistent with the original diagram. Note that the disk and the pin ***overlap*** near the center of *D*, but this does ***not affect*** the motion of the disk. To be more consistent with the physical system, a hole could be included at the center of *D*, but this would not change the kinematic results presented below. The hole at the center of *D* is assumed to be ***small*** compared to its size. The pin is included in the model for ***visual purposes*** only.

Frame *F2* of the disk is located at the ***center*** of its ***upper surface***. Point *A* is located from this point by a coordinate translation of length "*a*" in the ***negative*** *y* direction. This is done using a Rigid Transformation block. A ***spherical joint*** is located at this point that connects Disk *D* with the connecting rod (bar *AB*).

The ***connecting rod subsystem*** is shown in Panel 2 below. As noted above, bar *AB* consists of three major components – a spherical end (at *A*), a cylindrical middle section, and a U-shaped end with a cylindrical pin. *AB* is built within the Simscape model in a local *xz* plane. The U-shaped end appears in edge view in this plane, and the local *y* axis is normal to this plane. A Rigid Transform block is used to orient these local axes correctly relative to the world reference frame. The transpose of the coordinate transformation (rotation) matrix $[R_{AB}]$ defined above is stored as the variable "`connectingRodTransformationMatrix`" for this purpose. See the Block Parameters window for the Rigid Transform below. Note this transformation is applied at Compile-time as it does not apply after the system starts to move.

Simscape Multibody Model – Panel 2: (coordinate transformation, bar *AB*, measurement subsystem)



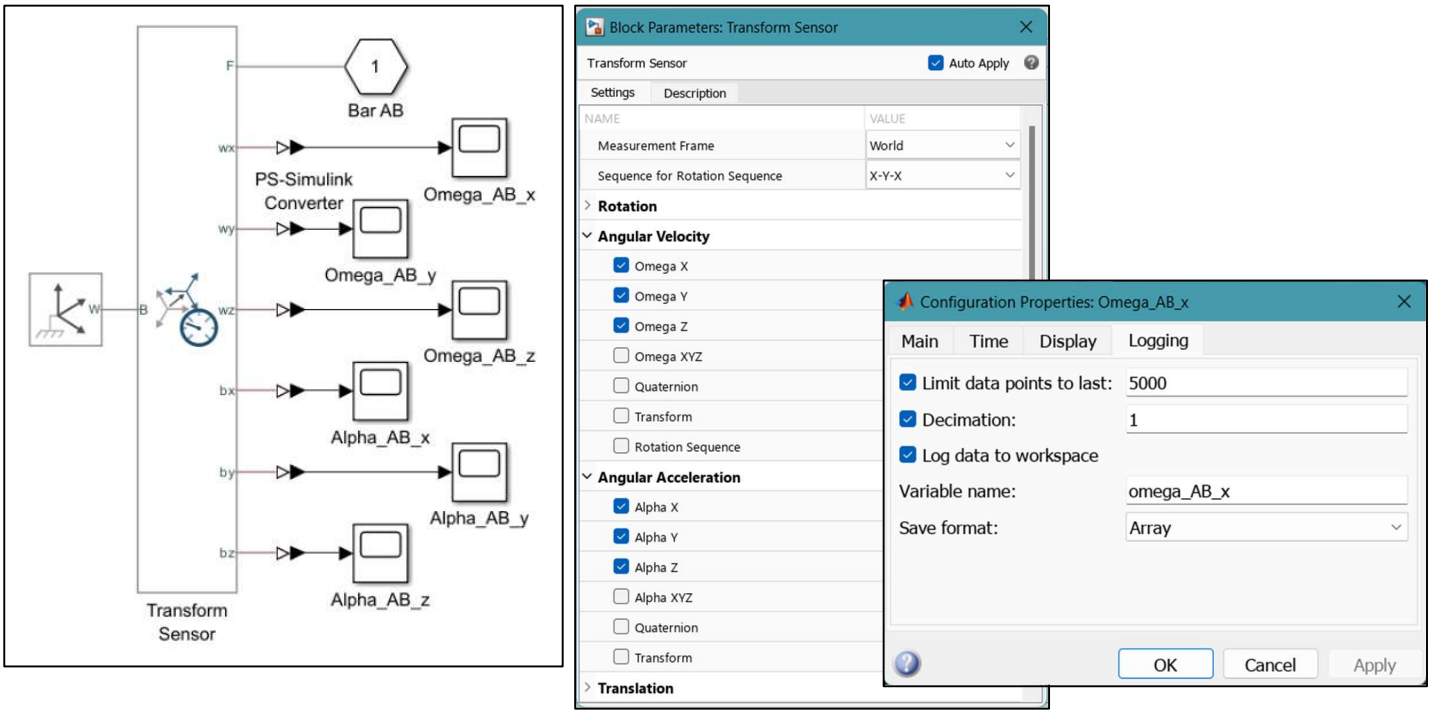
Frames *R* and *F1* of the ***spherical end*** are both located at its ***mass center***. Frame *R* is connected to the ***spherical joint*** at *A*, and frame *F1* is connected to ***one end*** of the cylindrical middle section of *AB*. Frames *F1* and *F2* of the ***cylindrical middle section*** are at opposite ends of the cylinder. Frame *F1* is connected to the ***spherical end***, and frame *F2* is connected to the ***U-shaped end***.

The ***subsystem*** that creates the ***U-shaped end*** with a ***spherical pin*** is also shown in Panel 2 above. The U-shaped end consists of a ***bottom plate*** and ***two side plates*** (on both sides). The ***bottoms*** of the ***first*** of the ***side plates*** are welded to the ***ends*** of the ***bottom plate***. The ***tops*** of the ***first side plates*** are welded to ***cylindrical pins*** and to the ***bottom edges*** of the ***second side plates***. The *F2* frames of ***both bottom side plates*** have been ***rotated*** so their local *z*-axes are perpendicular to the line from *A* to *B* making axes of the cylindrical pins along this line. The ends of the two cylindrical pin sections are positioned and touch along the centerline of *AB*. A subsystem is used to calculate the position of the center of the end of the left side cylindrical pin to verify that it has the correct initial position. This point should be at the location of point *B*.

The ***Connecting Rod Angular Motion Subsystem*** is shown in Panel 3 below. It uses a ***Transform Sensor*** to measure the ***angular velocity*** and ***angular acceleration*** of the central cylindrical section of the ***connecting rod*** relative to the world frame. The output signals of the sensor are ***physical signals*** (PS) that must be ***converted*** into

Simulink signals for plotting and storage. The Block Parameters window of the sensor shows that the x, y, and z components are all measured individually, and they are expressed in the world frame. The logging properties window of the scope for the x-component of ${}^{W}\underset{\sim}{\omega}_{AB} \triangleq {}^{R}\underset{\sim}{\omega}_{AB}$ shows the data will be stored in a MATLAB array with variable name "`omega_AB_x`". Similar arrays are generated for the other components. These arrays are used in the MATLAB script to plot the results.

Simscape Multibody Model – Panel 3: (angular motion measurement subsystem for AB)

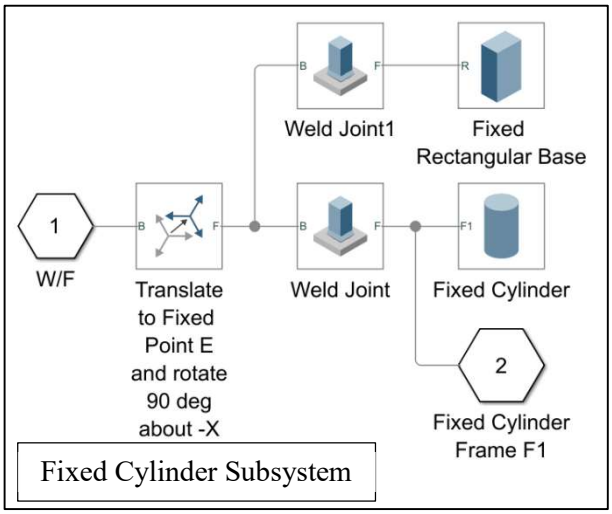


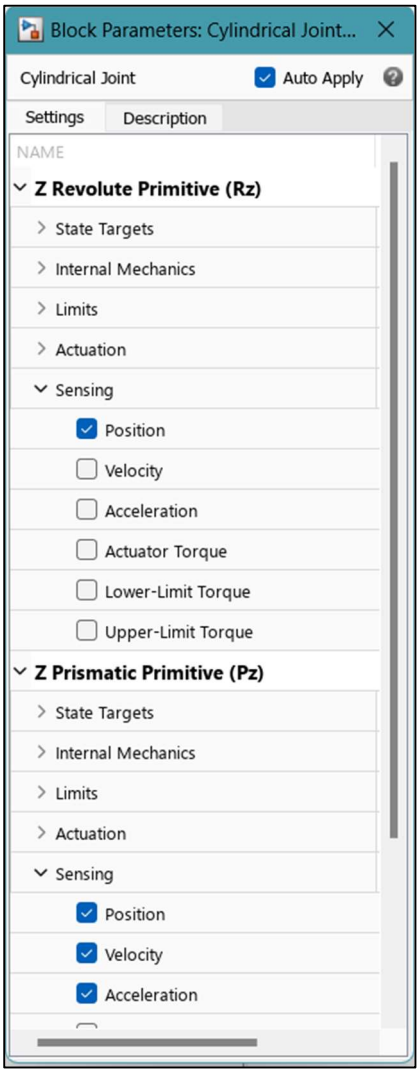
The portion of the system ***below*** the ***dashed line*** in Panel 1 ***positions*** the ***fixed horizontal cylinder*** (on which the collar moves), ***connects*** the ***cylindrical collar*** to the ***fixed horizontal cylinder*** with a ***cylindrical joint***, and performs the necessary ***coordinate transformations*** to ***align*** the pin joint at B with the cylindrical pin of the connecting rod. The upper and lower portions of the diagram are connected by the revolute joint at B.

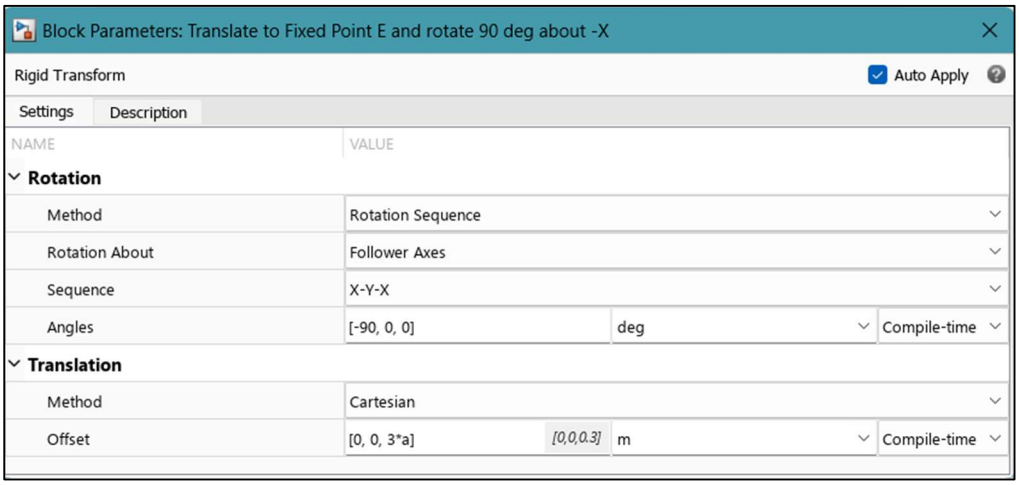
The Fixed Cylinder subsystem is shown in Panel 4 below. First, a Rigid Transform creates a new coordinate system at point E with its local z axis pointed in the direction of the World Frame's y axis. This is accomplished by translating to world frame coordinates $(0,0,3a)$ and then performing a -90 (deg) rotation about the x axis. See the Block Parameters box below the subsystem diagram below. The fixed horizontal cylinder and its rectangular base are both welded to this point.

The cylindrical joint is located along the fixed horizontal cylinder at B using a Rigid Transform to translate a distance of "$6a$" along the local z axis at compile time. This joint is connected to the mass center frame of the ***cylindrical collar***, and it is used to ***measure*** the changes in angle, position, velocity, and acceleration of the collar. See the Block Parameters window for the cylindrical joint in Panel 4.

Simscape Multibody Model – Panel 4: (Fixed Cylinder Subsystem, Cylindrical Joint Sensing)



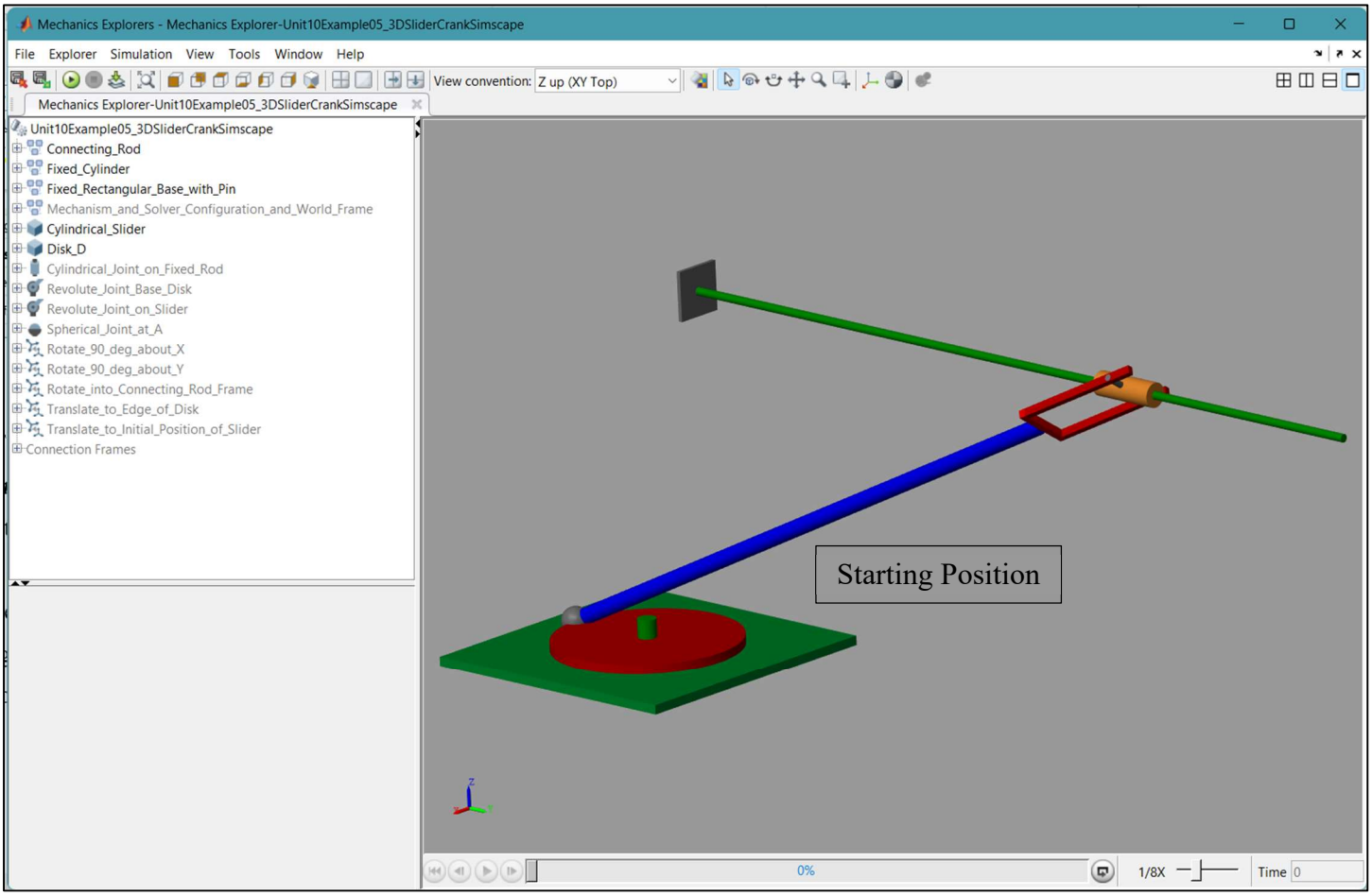




Finally, to ***connect*** the ***cylindrical slider*** to the ***revolute joint*** along the ***cylindrical pin axis*** of the connecting rod, ***three*** coordinate transformations are done using Rigid Transform blocks. The ***first*** is a 90-degree rotation about the local *x* axis of the collar. This creates a reference frame at the mass center of the collar which is aligned with the world coordinate system. The ***second transformation*** uses the ***connecting rod transformation matrix*** to create a reference frame which is aligned with the connecting rod. The local *x* axis of the connecting rod is pointed along its cylindrical pin, so a ***third transformation*** is required. This transformation is a 90-degree rotation about the local *y* axis which aligns the resulting local *z* axis with the axis of the revolute joint.

When the Simscape model is complete and all variables defined, ***updating*** the model from the Modeling tab ***opens*** the Mechanics Explorer and Animation window as shown in Panel 5. The left side of the window shows model explorer details which can be used to check the connectivity of the model. The animation window is used to visualize the initial position of the system and to animate its motion when the model is executed.

Simscape Multibody Model – Panel 5: (Mechanics Explorer and Animation Window)

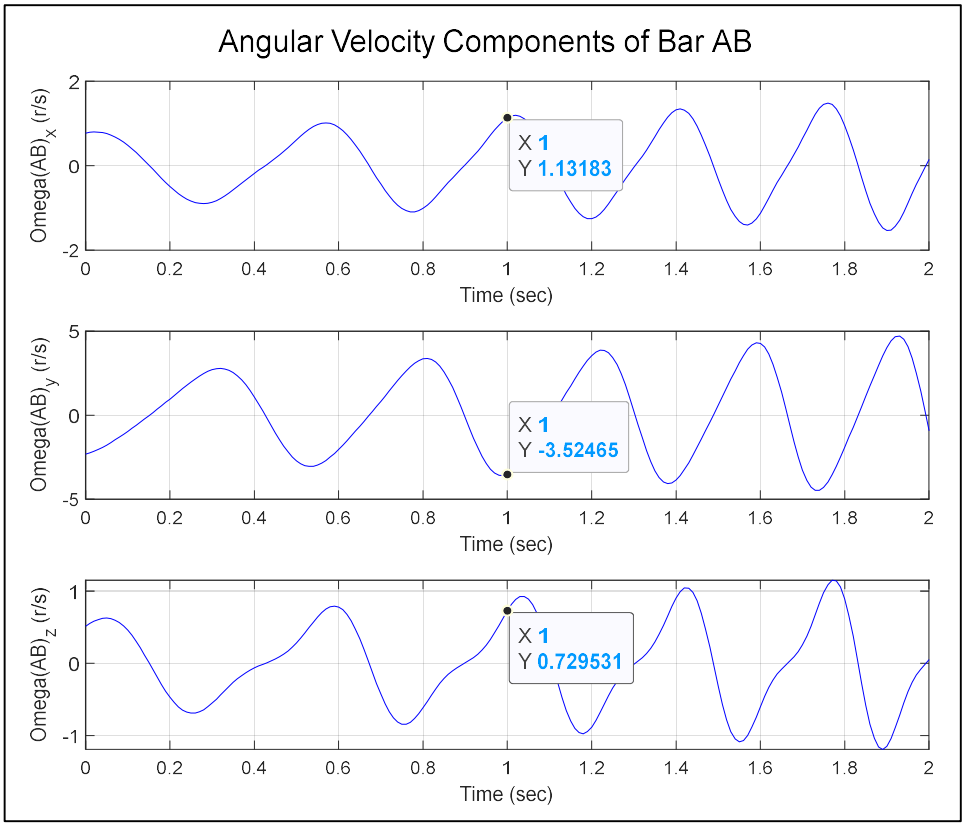


Execution of the MATLAB Script:

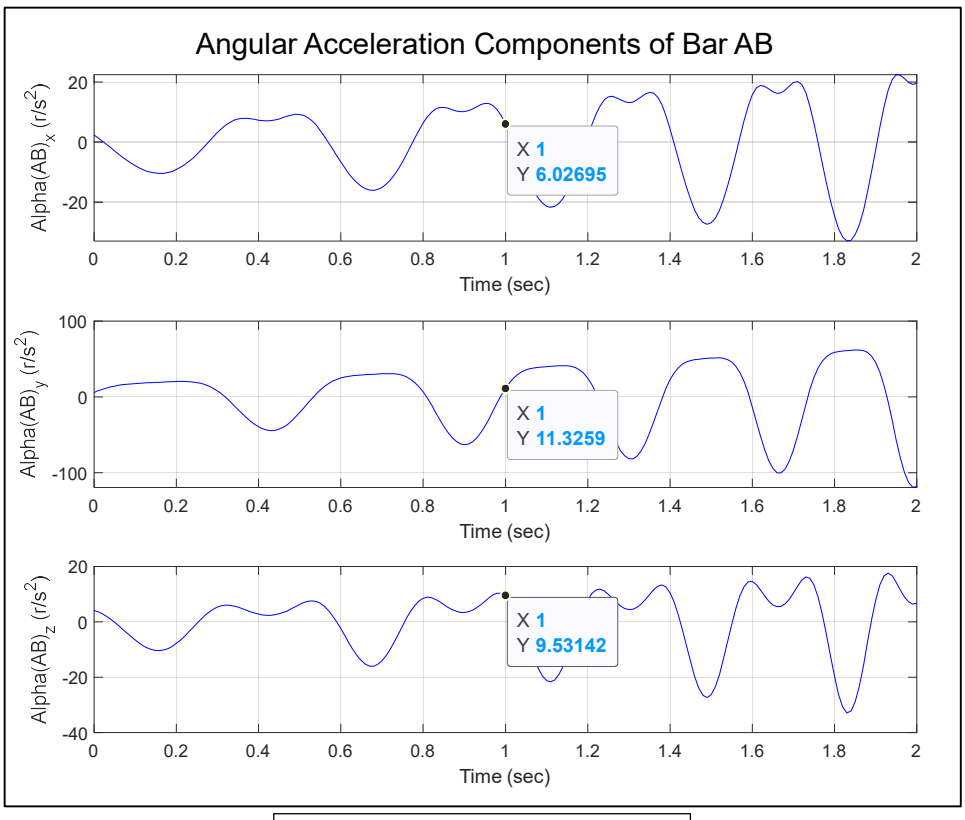
When the MATLAB script is executed, the model ***variables*** are ***defined***, the Simscape Multibody ***model*** is ***executed***, and the ***results*** are ***plotted*** in five figure windows. The animated motion of the system is displayed in the ***animation window***.

The first two plots below show the world components of ${}^{R}\underset{\sim}{\omega}_{AB}$ and ${}^{R}\underset{\sim}{\alpha}_{AB}$ the angular velocity and angular acceleration of bar AB as measured in the world frame. The third plot shows the position, velocity, and acceleration of the collar at B. Data tips are used to present the specific results at $t = 1$ (sec) for easy comparison with analytical results. The fourth plot shows the relative angle and position changes of the collar. Note that the collar both translates and rotates as it moves. Plots of the applied motion of the disk are not shown.

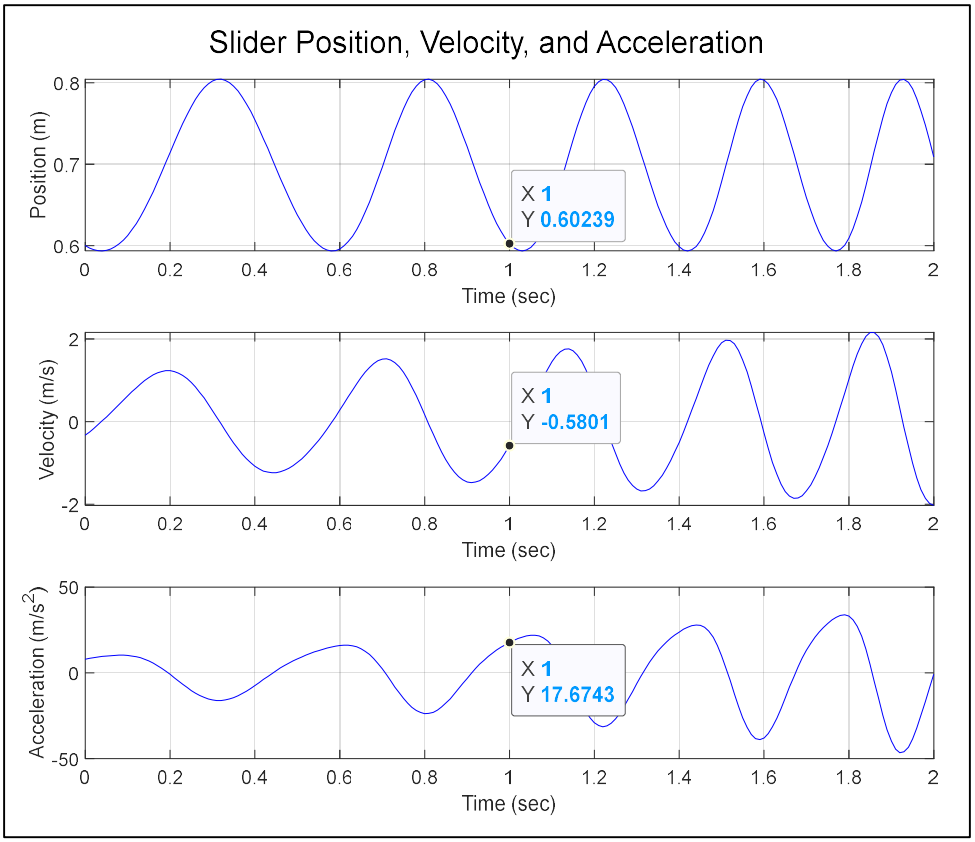
Three snap-shot views of the system from the animation window are shown below the plots. The snapshots are taken at $t = 0.9$ (sec) .



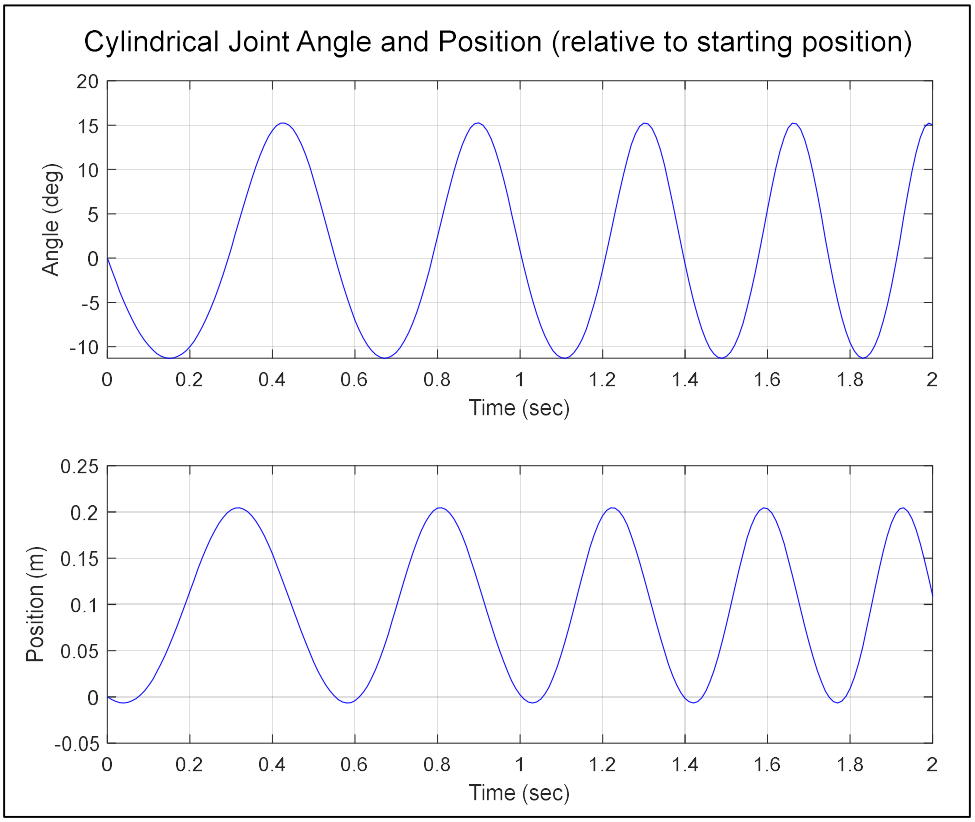
World Components of ${}^{R}\underset{\sim}{\omega}_{AB}$



World Components of ${}^{R}\underset{\sim}{\alpha}_{AB}$



Position, Velocity, and Acceleration of the Collar in the World Frame

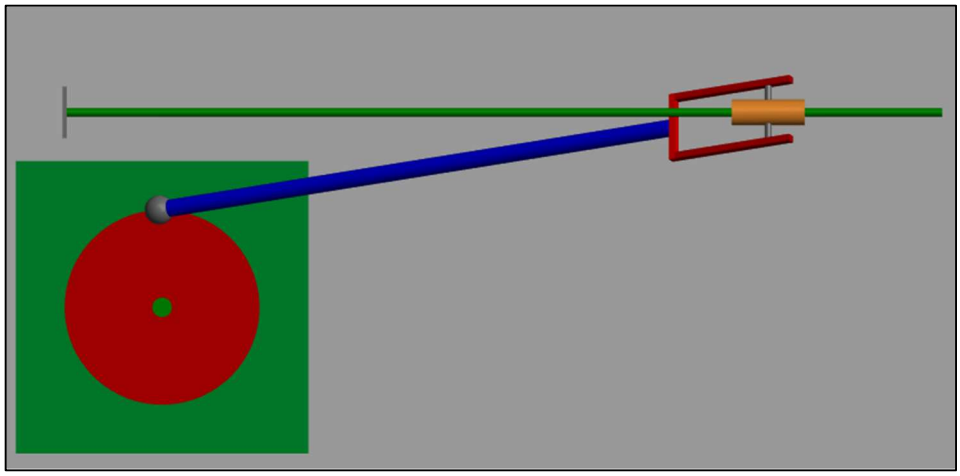


Relative Position and Angle Changes of the Collar at *B*

Simscape Multibody Snapshots: $t = 0.9$ (sec)

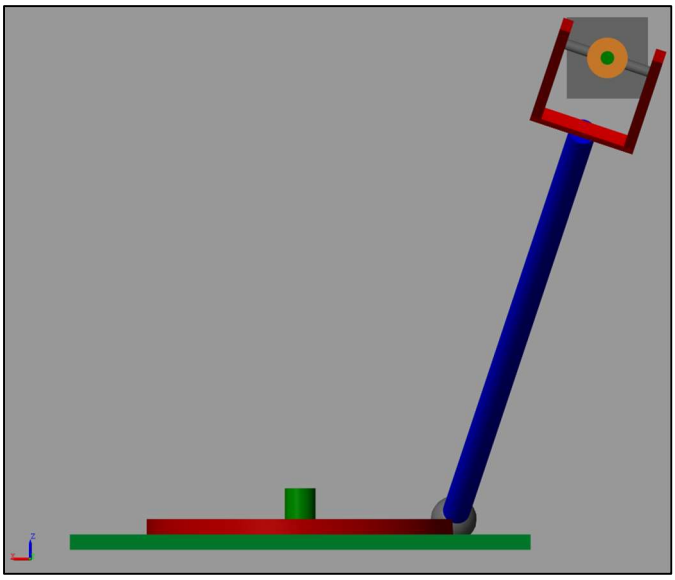
Top View:

Looking Down the World Z Axis

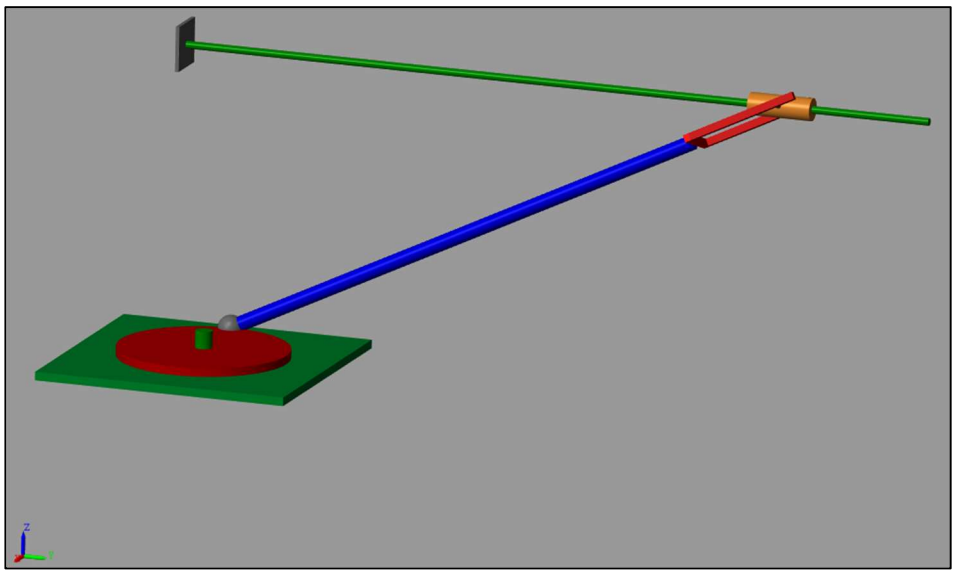


Right Side View:

Looking Down the World Y Axis



Isometric View



Analytical Results – MATLAB Script

To ***verify*** the results of the Simscape Multibody program a MATLAB ***script*** is included here that uses a set of analytical equations to calculate the motion of the system. Detailed derivations of these equations are presented in Example 1 of Unit 8 of this volume. The equations are repeated below for convenience. The angle ϕ is the ***angle*** of the ***disk*** and is calculated based on a ***zero starting angle*** and a ***constant angular acceleration***. The variables $\hat{y}_B$ and y_B (the y coordinate of collar B) are functions of angle ϕ and are calculated using the ***first equation***.

The ***world components*** of ${}^R\underset{\sim}{\omega}_{AB}$ the ***angular velocity*** of AB are calculated using the ***second equation***, and these results are used to calculate $\underset{\sim}{v}_B$ the ***velocity*** of the ***collar*** using the ***third equation***. The ***world components*** of ${}^R\underset{\sim}{\alpha}_{AB}$ the ***angular acceleration*** of AB are calculated using the ***fourth equation***, and these results are used to calculate $\underset{\sim}{a}_B$ the ***acceleration*** of the ***collar*** using the ***fifth equation***. Note the equations for the angular velocity and angular acceleration components are ***matrix equations***. The 3×3 coefficient matrix $[A]$ of these equations depends on the position of the system and is the same in both equations. Note that the ***angular displacement*** of collar B does ***not appear*** in these equations.

$$y_B = a\left[1 - C_\phi + \sqrt{40 - \left(2 + S_\phi\right)^2}\right] = a\,\hat{y}_B$$ (solve for $\hat{y}_B$ and y_B)

$$\begin{bmatrix} 0 & 3 & \left(1-C_\phi\right)-\hat{y}_B \\ \hat{y}_B-\left(1-C_\phi\right) & \left(2+S_\phi\right) & 0 \\ -\left(2+S_\phi\right) & 0 & 3 \end{bmatrix} \begin{Bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{Bmatrix} \triangleq [A]\{\omega\} = \begin{Bmatrix} -\Omega C_\phi \\ 0 \\ 0 \end{Bmatrix}$$ (solve for $\omega_x, \omega_y, \omega_z$)

$$v_B = -3a\omega_x - a\left(2+S_\phi\right)\omega_z + a\,\Omega\,S_\phi$$ (solve for v_B)

$$\begin{bmatrix} 0 & 3 & \left(1-C_\phi\right)-\hat{y}_B \\ \hat{y}_B-\left(1-C_\phi\right) & \left(2+S_\phi\right) & 0 \\ -\left(2+S_\phi\right) & 0 & 3 \end{bmatrix} \begin{Bmatrix} \alpha_x \\ \alpha_y \\ \alpha_z \end{Bmatrix} \triangleq [A] \begin{Bmatrix} \alpha_x \\ \alpha_y \\ \alpha_z \end{Bmatrix} =$$

$$\begin{Bmatrix} -\dot{\Omega}C_\phi + \Omega^2 S_\phi - \omega_y\left(\left(\hat{y}_B-\left(1-C_\phi\right)\right)\omega_x + \left(2+S_\phi\right)\omega_y\right) - \omega_z\left(\left(2+S_\phi\right)\omega_z + 3\omega_x\right) \\ \omega_x\left(\left(2+S_\phi\right)\omega_z + 3\omega_x\right) + \omega_y\left(3\omega_y - \left(\hat{y}_B-\left(1-C_\phi\right)\right)\omega_z\right) \\ \Omega C_\phi \omega_x \end{Bmatrix}$$ (solve for $\alpha_x, \alpha_y, \alpha_z$)

$$a_B = -3a\alpha_x - a\left(2+S_\phi\right)\alpha_z + a\,\dot{\Omega}S_\phi + a\,\Omega^2 C_\phi + \omega_z\left(3a\omega_y - \left(y_B - a\left(1-C_\phi\right)\right)\omega_z\right) - \omega_x\left(\left(y_B - a\left(1-C_\phi\right)\right)\omega_x + a\left(2+S_\phi\right)\omega_y\right)$$ (solve for a_B)

The MATLAB script is shown in the ***three panels*** below. Panel 1 shows the ***first four sections*** of the script. These sections describe the ***purpose*** of the script, ***clear*** the workspace variables, define the ***time vector***, define the ***geometric data*** for the crank and connecting rod (*AB*), and calculate the ***motion*** of the crank (angle, angular velocity, and angular acceleration) at each time in the time vector. The ***time vector*** starts at zero (initial time) and finishes at the specified final time "`timeFinal`" using the specified increment "`timeIncrement`". The angular acceleration of the disk "`diskAlpha`" is assumed to be constant with the specified initial angular velocity "`diskOmegaInitial`" and initial angular position "`diskAngleInitial`". It also calculates the sines and cosines associated of all the disk angles.

MATLAB Script – Panel 1: (Define system variables; Calculate applied Motion)

```
%% Volume I, Unit 10, Example 5, 3D Slider Crank Script
%
%  3D Slider Crank - Analytical Solution
%  Equations are as developed in Volume I, Unit 8, Example 1
%
   clear variables

%% Time Data
   timeFinal     = 2.0;    % (sec)
   timeIncrement = 0.01;  % (sec)
   timeVector = 0:timeIncrement:timeFinal; timeLength = length(timeVector);

%% Geometric data
   a                    = 0.1;      % (meters)
   diskRadius           = a;        % (meters)
   connectingRodLength  = 7*a;      % (meters)

%% Applied Motion
   diskAngleInitial = 0;   %  (r)
   diskOmegaInitial = 10;  %  (r/s)
   diskAlpha        =  5;  %  (r/s^2)
%  column angular rate (relative to ground)
   diskOmega = diskOmegaInitial + (diskAlpha*timeVector);
   diskAngle = diskAngleInitial + (diskOmegaInitial*timeVector) +   ...
                           0.5*(diskAlpha*timeVector.*timeVector);
   sinDiskAngle = sin(diskAngle); cosDiskAngle = cos(diskAngle);
                                      ⋮
```

Panel 2 below shows the details of the calculations of the ***angular velocity*** and ***angular acceleration*** components of *AB* and the ***position***, ***velocity***, and ***acceleration*** of the collar *B* at each time step. The script follows closely the procedure described above. Before making the calculations, the arrays are ***sized*** and ***initialized*** using the "`zeros`" function. The ***complete arrays*** "`yBHat`" and "`positionBinR`" are calculated directly using the ***arrays*** containing the ***sines*** and ***cosines*** of the ***disk angles*** for the ***entire duration*** of the motion. The rest of the results are calculated at ***each individual time step*** using a "`for`" loop.

Panel 3 shows the portion of the script that ***plots*** the ***calculated results***. The results are displayed in figure windows 6-9 for easy comparison with the Simscape Multibody results. These analytical results are identical to those generated by the Simscape Multibody model but are not presented here.

MATLAB Script – Panel 2: (Analytical Solution)

```
                                        ⋮
%% Analytical Solution
%  initialize arrays
   omegaABinR   = zeros(3,timeLength); alphaABinR       = zeros(3,timeLength);
   velocityBinR = zeros(1,timeLength); accelerationBinR = zeros(1,timeLength);
   coefficientMatrix = zeros(3); rhsOmega = zeros(3,1); rhsAlpha = zeros(3,1);

%  Position of Slider B
   temp01  = (2 + sinDiskAngle).*(2 + sinDiskAngle); temp01  = sqrt(40 - temp01);
   yBHat   = (1 - cosDiskAngle + temp01);  positionBinR = diskRadius*yBHat;

%  Angular Velocity and Acceleration of AB in R and
%  Velocity and Acceleration of B in R
   for itime = 1:timeLength
      coefficientMatrix(1,1) = 0; coefficientMatrix(1,2) = 3;
      coefficientMatrix(1,3) = (1 - cosDiskAngle(1,itime)) - yBHat(1,itime);
      coefficientMatrix(2,1) = -coefficientMatrix(1,3);
      coefficientMatrix(2,2) = 2 + sinDiskAngle(1,itime); coefficientMatrix(2,3) = 0;
      coefficientMatrix(3,1) = -coefficientMatrix(2,2);
      coefficientMatrix(3,2) = 0; coefficientMatrix(3,3) = 3;
      temp01 = yBHat(1,itime)- (1 - cosDiskAngle(1,itime));
      temp02 = 2 + sinDiskAngle(1,itime);

   %  angular velocity of AB in R
      rhsOmega = [-diskOmega(1,itime)*cosDiskAngle(1,itime); 0; 0];
      omegaABinR(:,itime) = coefficientMatrix\rhsOmega;

   %  velocity of B in R
      velocityBinR(1,itime) = (diskOmega(1,itime)*sinDiskAngle(1,itime)) - ...
           (3*omegaABinR(1,itime)) - (temp02*omegaABinR(3,itime));
      velocityBinR(1,itime) = diskRadius*velocityBinR(1,itime);

   %  angular acceleration of B in R
      rhsAlpha(1,1) = -(diskAlpha*cosDiskAngle(1,itime)) + ...
                       ((diskOmega(1,itime)^2)*sinDiskAngle(1,itime)) - ...
                       (temp01*omegaABinR(1,itime)*omegaABinR(2,itime)) - ...
                       (temp02*(omegaABinR(2,itime)^2)) - ...
                       (3*omegaABinR(1,itime)*omegaABinR(3,itime)) - ...
                       (temp02*(omegaABinR(3,itime)^2));
      rhsAlpha(2,1) = (3*(omegaABinR(1,itime)^2)) + ...
                      (temp02*omegaABinR(1,itime)*omegaABinR(3,itime)) + ...
                      (3*(omegaABinR(2,itime)^2)) - ...
                      (temp01*omegaABinR(2,itime)*omegaABinR(3,itime));
      rhsAlpha(3,1) = diskOmega(1,itime)*cosDiskAngle(1,itime)*omegaABinR(1,itime);
      alphaABinR(:,itime) = coefficientMatrix\rhsAlpha;

   %  acceleration of B in R
      accelerationBinR(1,itime) = (-3*alphaABinR(1,itime)) - ...
          (temp02*alphaABinR(3,itime)) + (diskAlpha*sinDiskAngle(1,itime)) + ...
          ((diskOmega(1,itime)^2)*cosDiskAngle(1,itime)) + ...
          (3*omegaABinR(2,itime)*omegaABinR(3,itime)) - ...
          (temp01*(omegaABinR(3,itime)^2)) - (temp01*(omegaABinR(1,itime)^2)) - ...
          (temp02*omegaABinR(1,itime)*omegaABinR(2,itime));
      accelerationBinR(1,itime) = diskRadius*accelerationBinR(1,itime);
   end
                                        ⋮
```

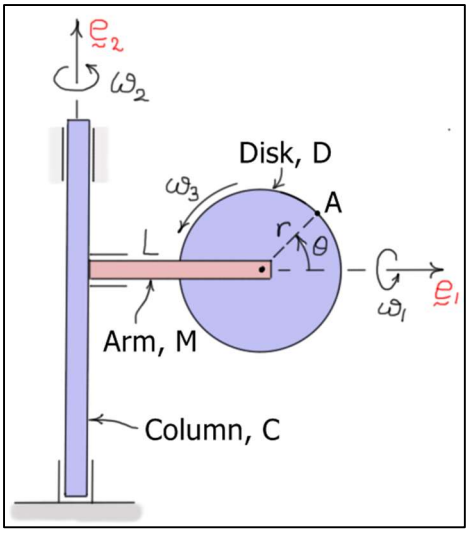
MATLAB Script – Panel 3: (Plot the calculated results)

```
                                                    ⋮
%% Plot the Calculated Results
%  Disk Motion
   figure(6); clf;
   subplot(2,1,1); plot(timeVector,(180/pi)*diskAngle,'b'); grid on;
   xlabel('Time (sec)'); ylabel('Angle (deg)');
   subplot(2,1,2); plot(timeVector,diskOmega,'b'); grid on;
   xlabel('Time (sec)'); ylabel('Angular Rate (r/s)');
   tempstr = ['Disk Angle and Angular Rate. Angular Acceleration = ',num2str(diskAlpha),' r/s^2.'];
   sgtitle(tempstr)

%  Angular Velocity
   figure(7); clf;
   subplot(3,1,1); plot(timeVector,omegaABinR(1,:),'b'); grid on;
   xlabel('Time (sec)'); ylabel('OmegaABinR_x (r/s)');
   subplot(3,1,2); plot(timeVector,omegaABinR(2,:),'b'); grid on;
   xlabel('Time (sec)'); ylabel('OmegaABinR_y (r/s)');
   subplot(3,1,3); plot(timeVector,omegaABinR(3,:),'b'); grid on;
   xlabel('Time (sec)'); ylabel('OmegaABinR_z (r/s)');
   sgtitle('Angular Velocity Components of AB in R')

%  Angular Acceleration
   figure(8); clf;
   subplot(3,1,1); plot(timeVector,alphaABinR(1,:),'b'); grid on;
   xlabel('Time (sec)'); ylabel('AlphaBinR_x (r/s^2)');
   subplot(3,1,2); plot(timeVector,alphaABinR(2,:),'b'); grid on;
   xlabel('Time (sec)'); ylabel('AlphaABinR_y (r/s^2)');
   subplot(3,1,3); plot(timeVector,alphaABinR(3,:),'b'); grid on;
   xlabel('Time (sec)'); ylabel('AlphaABinR_z (r/s^2)');
   sgtitle('Angular Acceleration Components of AB in R')

%  Position, Velocity, and Acceleration of Slider B
   figure(9); clf;
   subplot(3,1,1); plot(timeVector,positionBinR,'b'); grid on;
   xlabel('Time (sec)'); ylabel('Position (m)');
   subplot(3,1,2); plot(timeVector,velocityBinR,'b'); grid on;
   xlabel('Time (sec)'); ylabel('Velocity (m/s)');
   subplot(3,1,3); plot(timeVector,accelerationBinR,'b'); grid on;
   xlabel('Time (sec)'); ylabel('Acceleration (m/s^2)');
   sgtitle('Position, Velocity, and Acceleration of Slider B');
```

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) $\qquad r = 0.25$ (m)

$\dot{\omega}_2 = 3$ (rad/s^2) = constant $\qquad \omega_2(t=0) = 2$ (rad/s)

$\dot{\omega}_1 = 4$ (rad/s^2) = constant $\qquad \omega_1(t=0) = 3$ (rad/s)

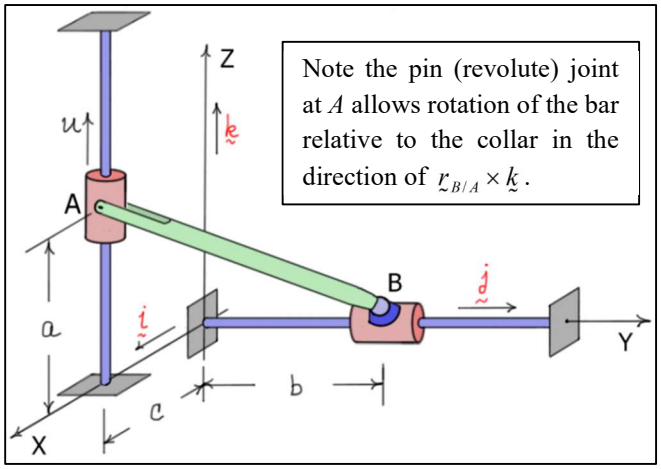
$\dot{\omega}_3 = 5$ (rad/s^2) = constant $\qquad \omega_3(t=0) = 4$ (rad/s)

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 Simscape Multibody 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 Simscape Multibody 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:



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

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 Simscape Multibody 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 Simscape Multibody 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