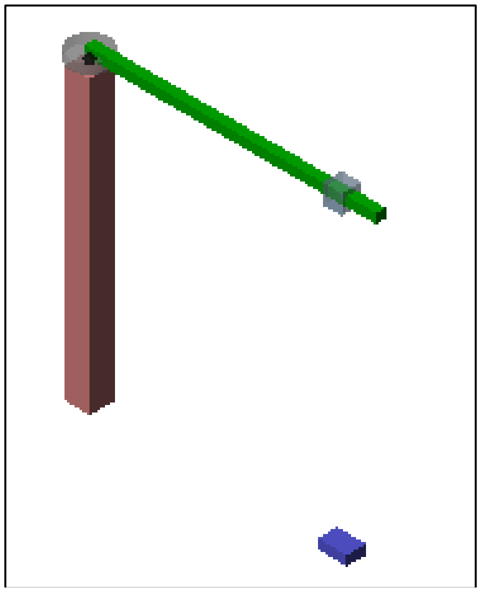
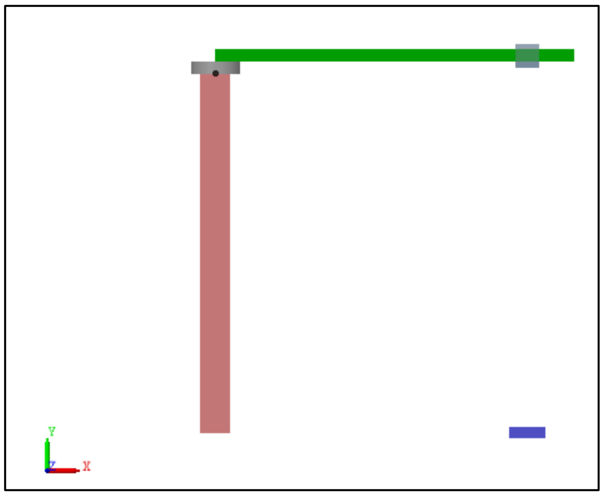
MEEM 4730 Project 4 Report

Simulation & verification of Jib Crane system by mathematical modeling in Simulink & Physical modeling in Simscape





Created & Submitted by: Vrushaketu Mali

24-Apr-2020

Table of Contents

List of figures

Chapter 1. Introduction

The first step in the dynamic system simulation is to develop appropriate mathematical models of the system to be simulated. These models can either be constructed from the physics equations or experimental data. Apart from writing bunch of syntax in the MatLab code, there are two other ways for simulation. First is mathematical modeling in Simulink and second is physical modeling in Simscape.

Simulink is a model-based design environment integrated with MatLab used for modeling, simulating, and analyzing multi-domain dynamic systems. It offers a way to solve the dynamic system equations numerically rather than requiring a code. Models contain blocks which are mathematical functions and signals which are lines connecting different blocks serving as carrying input and output values.

Simscape is an extension in the Simulink library which contains actual physical components, therefore complex multi-domain systems can be modeled without the need to build the mathematical equations.

In this project, we will be simulating the jib and crane system in Simulink by constructing the signal- based math model as well as in Simscape by constructing the component-based model. In order to ensure that both models are built correctly, we are going to compare the results of these simulation with each other and report the errors between them. In order to build the simulink signal based model, we need to have the differential equations of the system ready. The derivation of these equations is a part of this project report and those are derived using Lagrangian method in this project making the use of symbolic toolbox in MatLab.

Chapter 2. Summary

This project aims at creating a signal-based math-model in Simulink and component-based physical model in Simscape for simulating the hanging load system and verifying the results of these simulations by comparing with each other. The verification results, MatLab code description is discussed in detail in this project.

Accomplishments:

1. In this project, we have derived the differential equations of the system using Lagrangian method.
2. We have used those differential equations in signal-based simulink model for simulating the system and calculating the desired outputs.
3. Furthermore, we have also constructed the component-based model which does not require formulation of differential equations.
4. We have compared the results of both models to ensure that the equations we have formed are correctly representing the kinematics of the system.
5. For creating the models, we have made the use of buses, masked sub-systems and also variant sub-system. The information about these is also provided in this project report.
6. We have run and verified the simulation for first scenario of inputs. The verification is justified by providing the numerical values of maximum errors involved between two models.
7. After verification, we have modified the input of jib and simulated the system again.
8. At last, there are certain recommendations included for the future improvement in the model to make it more realistic.

Chapter 3. Project Statement/Problem Statement

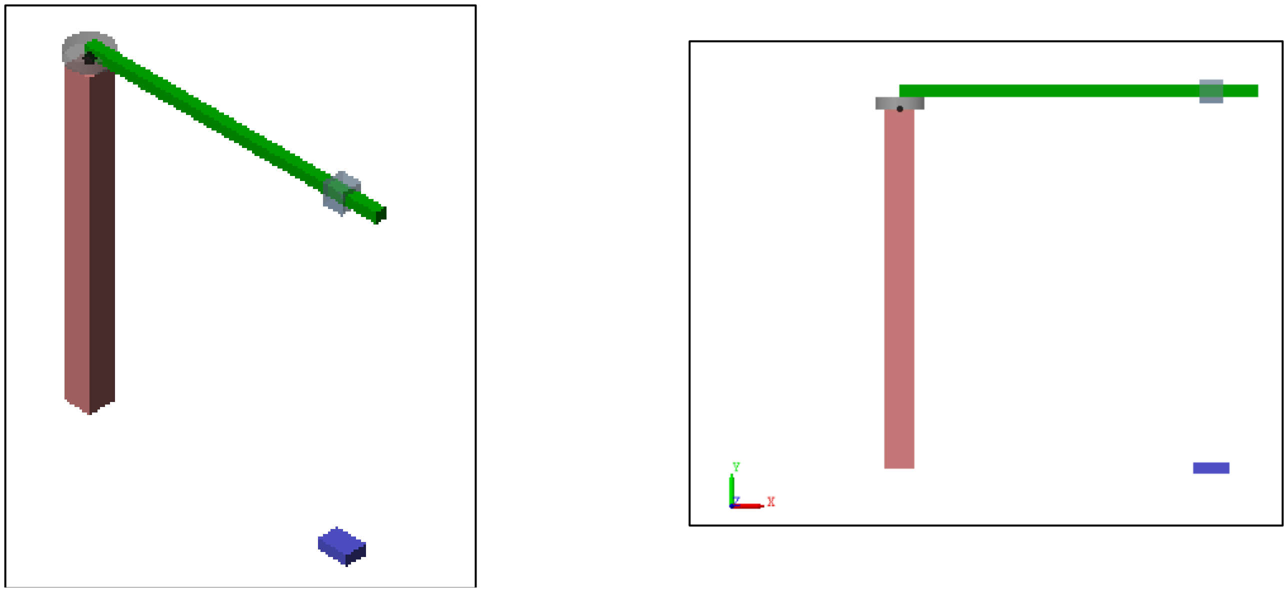


Figure 1: Project/Problem Statement

As shown in figure, Consider the jib crane system that has below parts.

1. Tower (red)
2. Motor disk (grey)
3. Jib (green)
4. Trolley (grey)
5. Load block (blue)

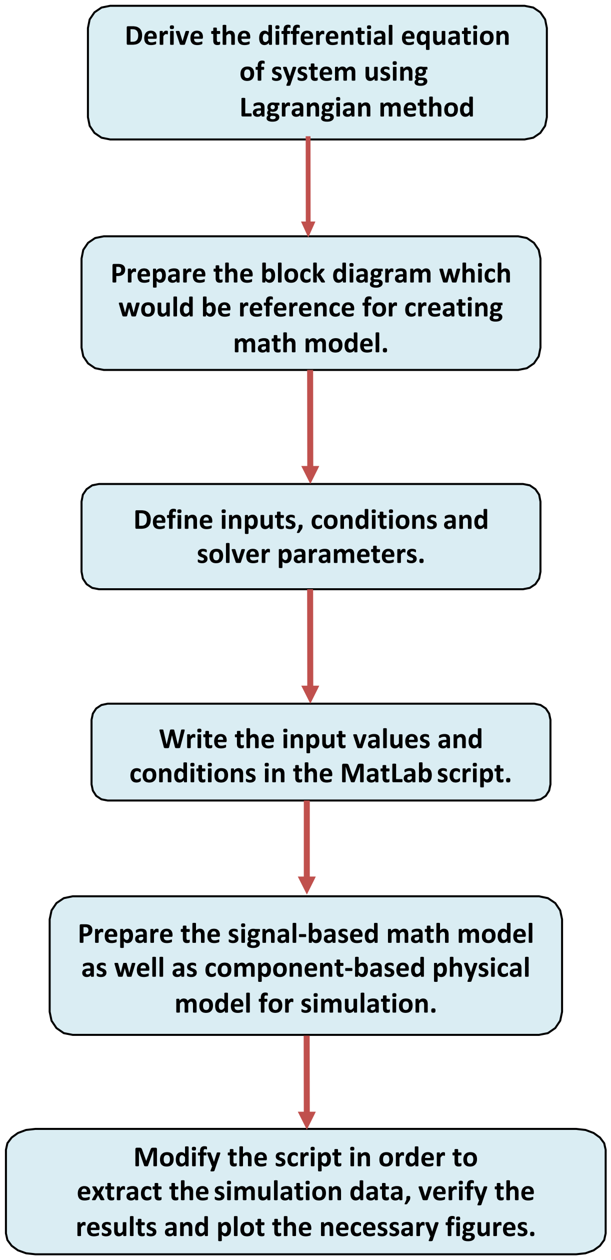
1. Develop a single live script .mlx file to use Lagrange's equation to derive the system's differential equation model using symbolic toolbox in MatLab.

2. Develop a single Simulink .slx file that contains both a signal-based and a component-based model of the system. Use this approach to justify some level of model verification where the trolley, load and jib assembly are actuated with the prescribed motion. Document and quantify the amount of force and torque needed for the prescribed motion.

3. Replace the prescribed motion of the jib assembly with the actuation by motor. Document the behavior of the slew motion for the motor driven case.

Chapter 4. Approach for simulation

Below flowchart shows the approach we would follow for simulating the system.


Derive the differential equation of system using Lagrangian method
Prepare the block diagram which would be reference for creating math model.
Define inputs, conditions and solver parameters.
Write the input values and conditions in the MatLab script.
Prepare the signal-based math model as well as component-based physical model for simulation.
Modify the script in order to extract the simulation data, verify the results and plot the necessary figures.

Chapter 5. Derivation of differential equations of system by Lagrangian Method

As we are going to create a signal based model along with component-based model, we need to have the differential equations of the system ready so that we can enter those equations (in the matrix form) in the signal based model and run the simulation. In this project, we are going to derive the differential equation of the system using Lagrangian approach by making use of the symbolic toolbox in MatLab live script.

In general, following steps are followed while deriving the DEQs with Lagrangian approach.

1. Determine the degrees of freedom & required generalized co-ordinates. (We will take help of body diagram for this).
2. Write down the position and velocity vectors.
3. Write the kinetic, potential energy as well as virtual work.
4. Apply Lagrange's equation and find out the DEQ of system.

Let us follow these steps one by one.

4.1 System diagram & Frames

From the figure shown in the project description, we can see that the tower is fixed. On the top frame of the tower, the jib is attached which rotates around the vertical axis of tower called as slew axis. The trolley is a cubical block with cubical extrusion inside it. The dimensions of this cubical extrusion are same as that the width and heights of the jib, so the trolley slides onto the jib. The load is attached to the jib through invisible cable such that its attachment point on top side is the geometric center of the bottom surface of the trolley and the attachment point on bottom side is the geometric center of the top surface of the load block. We will define certain frames at specific points in the system which are shown below.

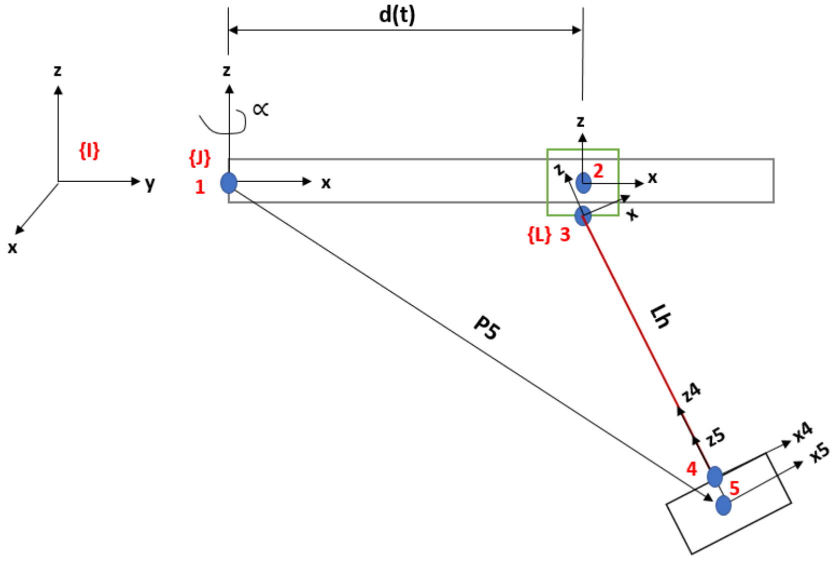


Figure 2: System Diagram

Below are the nomenclatures used in above figure as well as in the derivation.

1. Frame {I} : Inertial frame located at the intersection at Tower and Jib.
2. Frame {J} : Rotating Jib frame located at the intersection of Tower and Jib
3. Frame {L} : Rotating frame located at the hoist attachment point on trolley
4. Point 1 : The origin of Inertial {I} as well as Jib {J} frame.
5. Point 2 : The center of mass of trolley
6. Point 3 : The origin of frame L
7. Point 4 : The hoist attachment point on load block
8. Point 5 : The center of mass of load block
9. $\boldsymbol{d(t)}$ = Distance from the slew axis to the center of the trolley.
10. $\propto$ = Rotation angle of jib with respect to slew axis (Z-axis of tower)
11. $\boldsymbol{Lh}$ = Length from point 3 to point 4
12. $\boldsymbol{P}_5$ = Position vector to the center of mass of load block from origin of {I} frame.
13. $\boldsymbol{m}$ = mass of the load block
14. $\boldsymbol{I}_{xx}$ = Mass moment of Inertia of load block along x-x
15. $\boldsymbol{I}_{yy}$ = Mass moment of Inertia of load block along y-y
16. $\boldsymbol{I}_{zz}$ = Mass moment of Inertia of load block along z-z
17. $\boldsymbol{h_t}$ = Height of the trolley block

5.2 Degree of freedom of system and generalized co-ordinates

Let us define what degrees of freedom each joint has.

- Tower and Jib connection – It has one rotational degree of freedom such that jib rotates around vertical axis of the tower (slew axis) by an angle $\propto$. We will use revolute joint to make this rotation.

- Jib and Trolley connection – This has one translational degree of freedom such that trolley can slide on the jib. We will use prismatic joint to make this translation.

- Trolley and load connection – This have four degrees of freedom; One translational which causes the load to move vertically along the axis of the cable and three rotational degrees of freedom which causes load block to rotate along 3 axes. We will use Telescopic joint for this.

The rotation of load around three axis is a rotation sequence for which we will formulate the rotation matrices in the derivation. We will define the rotation sequence as XYZ and the rotation angles as $\emptyset, \boldsymbol{\theta}, \boldsymbol{\varphi}$

These angles will be generalized coordinates for our system.

$\boldsymbol{q_1}$ **=** $\emptyset$ **=** Rotation of load block around x-axis
$\boldsymbol{q_2}$ **=** $\boldsymbol{\theta}$ **=** Rotation of load block around y-axis
$\boldsymbol{q_2}$ **=** $\boldsymbol{\varphi}$ **=** Rotation of load block around z-axis

5.3 Derivation in live script using symbolic toolbox

First of all, we will write down the single axis rotation matrices which will be reference for deriving the equations.

For rotation around X-axis:

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & \sin(\alpha) \\ 0 & -\sin(\alpha) & \cos(\alpha) \end{bmatrix}$$

For rotation around Y-axis:

$$R_y(\alpha) = \begin{bmatrix} \cos(\alpha) & 0 & -\sin(\alpha) \\ 0 & 1 & 0 \\ \sin(\alpha) & 0 & \cos(\alpha) \end{bmatrix}$$

For rotation around Z-axis:

$$R_z(\alpha) = \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

We extract the angular velocity ${}^{b}\vec{\omega}_{b}$ from a rotation matrix using:

$${}^{b}\Omega_{b} = {}^{b}_{a}R\,{}^{a}_{b}\dot{R}$$

and we extract the components of ${}^{b}\vec{\omega}_{b}$ from the spin tensor as

$${}^{b}\Omega_{b} = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}$$

As with implementing Lagrange's Equation using Live Script, we'll need to move back and forth between functions of time and symbols. So, we'll need to create to arrays that contains the quantities that we'll substitute back and forth.

First, we will define all the variables used.

```
 clearvars;
 t = sym('t','real');   % time (s)
 sympref('default');

 % Constant parameters
 syms m g Ixx Iyy Izz ht; % define these constant parameters. ht = height of the trolley.
Ixx, Iyy & Izz are the mass moment of inertia of the load. m is the mass of the load.

 % Generalized coordinate states
 syms phiT(t) thtT(t) psiT(t) ;  % define names of three euler angles in time variable
 syms phiS    thtS    psiS    ;  % define names of three euler angles in symbolic
variable
 syms phidS   thtdS   psidS   ;  % define names of first derivative of three euler angles
in symbolic variable
 syms phiddS  thtddS  psiddS  ;  % define names of second derivative of three euler
angles in symbolic variable


 % Attachment point states
 syms xT(t) yT(t) LhT(t) ;  % define the x y & z motion of attachment point in time
variable
 syms xS    yS    LhS ;     % define the x y & z motion of attachment point in symbolic
variable
 syms xdS   ydS   LhdS ;    % define names of first derivative of motions in symbolic
variable
 syms xddS  yddS  LhddS ;   % define names of second derivative of motions in symbolic
variable

 xdT   = diff(xT,t);    % Calculate velocity of attachment point in x-direction
 ydT   = diff(yT,t);    % Calculate velocity of attachment point in y-direction
 phidT = diff(phiT,t);  % Differentiate rotation angle phi
 thtdT = diff(thtT,t);  % Differentiate rotation angle theta
 psidT = diff(psiT,t);  % Differentiate rotation angle psi
 LhdT = diff(LhT, t);   % Differentiate motion of load in z-axis

 xddT   = diff(xdT,t); % Calculate acceleration of attachment point in x-direction
 yddT   = diff(ydT,t); % calculate accleration of attachment point in y-direction.
 phiddT = diff(phidT,t);
 thtddT = diff(thtdT,t);
 psiddT = diff(psidT,t);
 LhddT = diff(LhdT, t);

 varT = [xT, yT, LhT, phiT, thtT, psiT,  xdT, ydT, LhdT, phidT, thtdT, psidT, xddT, yddT,
LhddT, phiddT, thtddT, psiddT];  % Array of time variables
 varS = [xS, yS, LhS, phiS, thtS, psiS,  xdS, ydS, LhdS, phidS, thtdS, psidS, xddS, yddS,
LhddS, phiddS, thtddS, psiddS];  % Array of symbolic variables
```

Step 1 - Position vector from point 1 to point 3.

Define position vector to point 3 from point 1 represented in {J} frame.

```
r3JS = [xS;yS;-0.5*ht]; % abs pos vec to point 3
r3JT = subs(r3JS,varS,varT) % create t ver
```

r3JT =

$$\begin{pmatrix} \mathrm{xT}(t) \\ \mathrm{yT}(t) \\ -\frac{\mathrm{ht}}{2} \end{pmatrix}$$

```
v3JT = diff(r3JT,t) % differentiate
```

v3JT =

$$\begin{pmatrix} \frac{\partial}{\partial t}\,\mathrm{xT}(t) \\ \frac{\partial}{\partial t}\,\mathrm{yT}(t) \\ 0 \end{pmatrix}$$

```
v3JS = subs(v3JT,varT,varS); % create sym ver
```

Step 2 - Position vector from point 3 to point 5.

Define position vector to point 5 from point 3 and represent it in {J} frame.

Let's create $\frac{d}{dt}\left({}^{J}\vec{r}_{5/3}\right)$

```
r53JS = [0;0;-LhS]; % vec  to 5 rel to 3, rep in {J}
r53JT = subs(r53JS,varS,varT); % create t ver
r53dJT = diff(r53JT,t)  % differentiate
```

r53dJT =

$$\begin{pmatrix} 0 \\ 0 \\ -\frac{\partial}{\partial t}\,\mathrm{LhT}(t) \end{pmatrix}$$

```
r53dJS = subs(r53dJT,varT,varS);  % create s ver
```

Step 3 - Rotation matrices for Euler angle sequences and its multiplication

Now we'll create the rotation matrix for an X-Y-Z Euler angle sequence for rotation from Jib frame {J} to load frame {L} . ${}^{L}_{J}R = R_z(\psi)R_y(\theta)\,R_x(\phi)$ and its transpose. We'll use the notation RJL for this, read as R, J to L.

```
RxS = [1  0         0
       0  cos(phiS) sin(phiS)
       0 -sin(phiS) cos(phiS)]   % Rotation matrix for rotation around x-axis by angle
phi
```

```
RxS =
```

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\mathrm{phiS}) & \sin(\mathrm{phiS}) \\ 0 & -\sin(\mathrm{phiS}) & \cos(\mathrm{phiS}) \end{pmatrix}$$

```
RyS = [cos(thtS) 0 -sin(thtS)
           0     1     0
       sin(thtS) 0  cos(thtS)]   %Mmatrix for rotation around y-axis by angle theta
```

```
RyS =
```

$$\begin{pmatrix} \cos(\mathrm{thtS}) & 0 & -\sin(\mathrm{thtS}) \\ 0 & 1 & 0 \\ \sin(\mathrm{thtS}) & 0 & \cos(\mathrm{thtS}) \end{pmatrix}$$

```
RzS = [cos(psiS)  sin(psiS)  0
       -sin(psiS) cos(psiS)  0
            0         0      1 ] % Matrix for rotation around z-axis by angle psi
```

```
RzS =
```

$$\begin{pmatrix} \cos(\mathrm{psiS}) & \sin(\mathrm{psiS}) & 0 \\ -\sin(\mathrm{psiS}) & \cos(\mathrm{psiS}) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

```
RJLS = RzS * RyS * RxS
```

```
RJLS =
```

$$\begin{pmatrix} \cos(\mathrm{psiS})\cos(\mathrm{thtS}) & \cos(\mathrm{phiS})\sin(\mathrm{psiS}) + \cos(\mathrm{psiS})\sin(\mathrm{phiS})\sin(\mathrm{thtS}) & \sin(\mathrm{phiS})\sin(\mathrm{psiS}) - \cos(\mathrm{phiS})\cos(\mathrm{psiS})\sin(\mathrm{thtS}) \\ -\cos(\mathrm{thtS})\sin(\mathrm{psiS}) & \cos(\mathrm{phiS})\cos(\mathrm{psiS}) - \sin(\mathrm{phiS})\sin(\mathrm{psiS})\sin(\mathrm{thtS}) & \cos(\mathrm{psiS})\sin(\mathrm{phiS}) + \cos(\mathrm{phiS})\sin(\mathrm{psiS})\sin(\mathrm{thtS}) \\ \sin(\mathrm{thtS}) & -\cos(\mathrm{thtS})\sin(\mathrm{phiS}) & \cos(\mathrm{phiS})\cos(\mathrm{thtS}) \end{pmatrix}$$

```
RLJS = RJLS.'
```

```
RLJS =
```

$$\begin{pmatrix} \cos(\mathrm{psiS})\cos(\mathrm{thtS}) & -\cos(\mathrm{thtS})\sin(\mathrm{psiS}) & \sin(\mathrm{thtS}) \\ \cos(\mathrm{phiS})\sin(\mathrm{psiS}) + \cos(\mathrm{psiS})\sin(\mathrm{phiS})\sin(\mathrm{thtS}) & \cos(\mathrm{phiS})\cos(\mathrm{psiS}) - \sin(\mathrm{phiS})\sin(\mathrm{psiS})\sin(\mathrm{thtS}) & -\cos(\mathrm{thtS})\sin(\mathrm{phiS}) \\ \sin(\mathrm{phiS})\sin(\mathrm{psiS}) - \cos(\mathrm{phiS})\cos(\mathrm{psiS})\sin(\mathrm{thtS}) & \cos(\mathrm{psiS})\sin(\mathrm{phiS}) + \cos(\mathrm{phiS})\sin(\mathrm{psiS})\sin(\mathrm{thtS}) & \cos(\mathrm{phiS})\cos(\mathrm{thtS}) \end{pmatrix}$$

Step 4 - Angular Velocity of Load

To create ${}^{L}_{J}\dot{R}$ we need to substitute the symbolic variables with their time dependent cousins

```
RLJT = subs(RLJS,varS,varT); % create t ver
RLJdT = diff(RLJT,t);  % differentiate
RLJdS = subs(RLJdT,varT,varS);  % create s ver
```

Now we'll form ${}^{L}\Omega_{L} = {}^{J}_{L}R\ {}^{J}_{L}\dot{R}$

```
OmgLLS = simplify(RJLS * RLJdS);  % create the matrix of angular velocities
```

Finally, extract the components of ${}^{L}\overrightarrow{\omega}_{L}$. This is the absolute angular velocity of the hoist cable, represented in {J}.

```
omgLLS = [OmgLLS(3,2);OmgLLS(1,3);OmgLLS(2,1)]  % Extract the components of angular
velocities
```

```
omgLLS =
```

$$\begin{pmatrix} \mathrm{thtdS}\sin(\mathrm{psiS}) + \mathrm{phidS}\cos(\mathrm{psiS})\cos(\mathrm{thtS}) \\ \mathrm{thtdS}\cos(\mathrm{psiS}) - \mathrm{phidS}\cos(\mathrm{thtS})\sin(\mathrm{psiS}) \\ \mathrm{psidS} + \mathrm{phidS}\sin(\mathrm{thtS}) \end{pmatrix}$$

Step 5 - Velocity of center of the load (point 5)

The general forms of the absolute position and velocity of the load are:

$\overrightarrow{r}_2 = \overrightarrow{r}_1 + \overrightarrow{r}_{2/1}$ and $\overrightarrow{v}_2 = \overrightarrow{v}_1 + \frac{d}{dt}\left(\overrightarrow{r}_{2/1}\right) + \overrightarrow{\omega} \times \overrightarrow{r}_{2/1}$

where

$${}^{a}\overrightarrow{v}_1 = [\dot{x}, \dot{y}, 0]^T$$

We will eventually create KE using ${}^{a}\overrightarrow{v}_2$. With the following workflow

1. Form ${}^{a}\overrightarrow{v}_1$, though we have that above...
2. Form $\frac{d}{dt}\left({}^{b}\overrightarrow{r}_{2/1}\right)$ and then convert to {a}
3. Form ${}^{b}\overrightarrow{\omega} \times {}^{b}\overrightarrow{r}_{2/1}$ and then also convert to {a}

To create $\overrightarrow{\omega} \times \overrightarrow{r}_{5/3}$ we can use our spin tensor scheme.

```
omCrossTermS = OmgLLS * r53JS;
```

Now combine the last two terms of

$$\vec{v}_5 = \vec{v}_3 + {}^L_J R \times \left[\frac{d}{dt}\left(\vec{r}_{5/3}\right) + \vec{\omega} \times ({}^J\vec{r}_{5/3})\right]$$

that are currently represented in {J} and change their representation to {I}

```
v5IS = v3JS + RLJS * (r53dJS + omCrossTermS)
```

v5IS =

$$\begin{pmatrix} \mathrm{xdS} - \mathrm{LhdS}\sin(\mathrm{thtS}) - \mathrm{LhS}\cos(\mathrm{psiS})\cos(\mathrm{thtS})\,\sigma_1 - \mathrm{LhS}\cos(\mathrm{thtS})\sin(\mathrm{psiS})\,\sigma_2 \\ \mathrm{ydS} + \mathrm{LhdS}\cos(\mathrm{thtS})\sin(\mathrm{phiS}) - \mathrm{LhS}\,\left(\cos(\mathrm{phiS})\sin(\mathrm{psiS}) + \cos(\mathrm{psiS})\sin(\mathrm{phiS})\sin(\mathrm{thtS})\right)\sigma_1 + \mathrm{LhS}\,\left(\cos(\mathrm{phiS})\cos(\mathrm{psiS}) - \sin(\mathrm{phiS})\sin(\mathrm{psiS})\sin(\mathrm{thtS})\right)\sigma_2 \\ \mathrm{LhS}\,\left(\cos(\mathrm{psiS})\sin(\mathrm{phiS}) + \cos(\mathrm{phiS})\sin(\mathrm{psiS})\sin(\mathrm{thtS})\right)\sigma_2 - \mathrm{LhdS}\cos(\mathrm{phiS})\cos(\mathrm{thtS}) - \mathrm{LhS}\,\left(\sin(\mathrm{phiS})\sin(\mathrm{psiS}) - \cos(\mathrm{phiS})\cos(\mathrm{psiS})\sin(\mathrm{thtS})\right)\sigma_1 \end{pmatrix}$$

where

$$\sigma_1 = \mathrm{thtdS}\cos(\mathrm{psiS}) - \mathrm{phidS}\cos(\mathrm{thtS})\sin(\mathrm{psiS})$$

$$\sigma_2 = \mathrm{thtdS}\sin(\mathrm{psiS}) + \mathrm{phidS}\cos(\mathrm{psiS})\cos(\mathrm{thtS})$$

Now we'll move on to finding KE, PE, the Lagrangian and implement Lagrange's Equation.

Step 6 - Kinetic Energy

$$T = \frac{1}{2} m \vec{v}_2^{\,T} \vec{v}_2 + \frac{1}{2} \vec{\omega}^{\,T} I_{cm} \vec{\omega}$$

```
Icm = [Ixx 0   0
       0   Iyy 0
       0   0   Izz]; % it's assumed there are no products of inertia

T = m * (v5IS.' * v5IS) / 2 + (omgLLS.' * Icm) * omgLLS / 2 % calculate Kinetic Energy
```

T =

$$\frac{\mathrm{Izz}\,(\mathrm{psidS} + \mathrm{phidS}\sin(\mathrm{thtS}))^2}{2} + \frac{\mathrm{Ixx}\,\sigma_2^2}{2} + \frac{\mathrm{Iyy}\,\sigma_1^2}{2} + \frac{m\,\left((\mathrm{ydS} + \mathrm{LhdS}\cos(\mathrm{thtS})\sin(\mathrm{phiS}) - \mathrm{LhS}\,(\cos(\mathrm{phiS})\sin(\mathrm{psiS}) + \cos(\mathrm{psiS})\sin(\mathrm{phiS})\sin(\mathrm{thtS}))\,\sigma_1 + \mathrm{LhS}\,(\cos(\mathrm{phiS})\cos(\mathrm{psiS}) - \sin(\mathrm{phiS})\sin(\mathrm{psiS})\sin(\mathrm{thtS}))\,\sigma_2)^2 + (\mathrm{LhdS}\sin(\mathrm{thtS}) - \mathrm{xdS} + \mathrm{LhS}\cos(\mathrm{psiS})\cos(\mathrm{thtS})\,\sigma_1 + \mathrm{LhS}\cos(\mathrm{thtS})\sin(\mathrm{psiS})\,\sigma_2)^2 + (\mathrm{LhdS}\cos(\mathrm{phiS})\cos(\mathrm{thtS}) - \mathrm{LhS}\,(\cos(\mathrm{psiS})\sin(\mathrm{phiS}) + \cos(\mathrm{phiS})\sin(\mathrm{psiS})\sin(\mathrm{thtS}))\,\sigma_2 + \mathrm{LhS}\,(\sin(\mathrm{phiS})\sin(\mathrm{psiS}) - \cos(\mathrm{phiS})\cos(\mathrm{psiS})\sin(\mathrm{thtS}))\,\sigma_1)^2\right)}{2}$$

where

$$\sigma_1 = \mathrm{thtdS}\cos(\mathrm{psiS}) - \mathrm{phidS}\cos(\mathrm{thtS})\sin(\mathrm{psiS})$$

$$\sigma_2 = \mathrm{thtdS}\sin(\mathrm{psiS}) + \mathrm{phidS}\cos(\mathrm{psiS})\cos(\mathrm{thtS})$$

Step 7 - Potential Energy

We will use the dot product approach described in class

$$V = mg\,{}^I\vec{r}_{5/3} \cdot \hat{z}_I$$

```
V = m*g* ( (RLJS*r53JS).' * [0;0;1]) % Calculate potential energy
```

V = $-\mathrm{LhS}\,g\,m\cos(\mathrm{phiS})\cos(\mathrm{thtS})$

Step 8 - Formulation of Lagrangian Term (L)

Our three generalized coordinates are ϕ , θ & ψ .

```
L = T-V  % Calculate Lagrangian
```

L =

$$\frac{\mathrm{Izz}\,(\mathrm{psidS}+\mathrm{phidS}\sin(\mathrm{thtS}))^2}{2}+\frac{\mathrm{Ixx}\,\sigma_2^2}{2}+\frac{\mathrm{Iyy}\,\sigma_1^2}{2}+\frac{m\,\left((\mathrm{ydS}+\mathrm{LhdS}\cos(\mathrm{thtS})\sin(\mathrm{phiS})-\mathrm{LhS}\,(\cos(\mathrm{phiS})\sin(\mathrm{psiS})+\cos(\mathrm{psiS})\sin(\mathrm{phiS})\sin(\mathrm{thtS}))\,\sigma_1+\mathrm{LhS}\,(\cos(\mathrm{phiS})\cos(\mathrm{psiS})-\sin(\mathrm{phiS})\sin(\mathrm{psiS})\sin(\mathrm{thtS}))\,\sigma_2)^2+(\mathrm{LhdS}\sin(\mathrm{thtS})-\mathrm{xdS}+\mathrm{LhS}\cos(\mathrm{psiS})\cos(\mathrm{thtS})\,\sigma_1+\mathrm{LhS}\cos(\mathrm{thtS})\sin(\mathrm{psiS})\,\sigma_2)^2+(\mathrm{LhdS}\cos(\mathrm{phiS})\cos(\mathrm{thtS})-\mathrm{LhS}\,(\cos(\mathrm{psiS})\sin(\mathrm{phiS})+\cos(\mathrm{phiS})\sin(\mathrm{psiS})\sin(\mathrm{thtS}))\,\sigma_2+\mathrm{LhS}\,(\sin(\mathrm{phiS})\sin(\mathrm{psiS})-\cos(\mathrm{phiS})\cos(\mathrm{psiS})\sin(\mathrm{thtS}))\,\sigma_1)^2\right)}{2}+\mathrm{LhS}\,g\,m\cos(\mathrm{phiS})\cos(\mathrm{thtS})$$

where

$\sigma_1 = \mathrm{thtdS}\cos(\mathrm{psiS}) - \mathrm{phidS}\cos(\mathrm{thtS})\sin(\mathrm{psiS})$

$\sigma_2 = \mathrm{thtdS}\sin(\mathrm{psiS}) + \mathrm{phidS}\cos(\mathrm{psiS})\cos(\mathrm{thtS})$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \left(\frac{\partial L}{\partial q}\right)$$

Step 9 - Equations of generalized co-ordinates.

ϕ *Equation :*

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\phi}}\right) - \left(\frac{\partial L}{\partial \phi}\right)$$

```
dLdPhiDS = diff(L,phidS);  % differentiate L w.r.t phi Dot
dLdPhiDT = subs(dLdPhiDS,varS,varT); % Create t ver
ddt_dLdPhiDT = diff(dLdPhiDT,t);   % differentiate w.r.t time
ddt_dLdPhiDS = subs(ddt_dLdPhiDT,varT,varS);  % create s ver
dLdPhiS = diff(L,phiS);  % differentiate L w.r.t Phi
phiEqS = expand(ddt_dLdPhiDS-dLdPhiS)  % Form the equation for Phi
```

θ *Equation :*

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

```
dLdThtDS = diff(L,thtdS);   % differentiate L with respect to theta Dot
dLdThtDT = subs(dLdThtDS,varS,varT);  % Create t ver
ddt_dLdThtDT = diff(dLdThtDT,t);   % differentiate w.r.t time
ddt_dLdThtDS = subs(ddt_dLdThtDT,varT,varS);   % create s ver
dLdThtS = diff(L,thtS);  % differentiate L w.r.t theta
thtEqS = expand(ddt_dLdThtDS-dLdThtS)  % Form the equation for theta
```

ψ Equation :

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\psi}}\right) - \left(\frac{\partial L}{\partial \psi}\right)$$

```
dLdpsiDS = diff(L,psidS);  % differentiate L with respect to psi Dot
dLdpsiDT = subs(dLdpsiDS,varS,varT);  % Create t ver
ddt_dLdpsiDT = diff(dLdpsiDT,t);   % differentiate w.r.t time
ddt_dLdpsiDS = subs(ddt_dLdpsiDT,varT,varS);  % create s ver
dLdpsiS = diff(L,psiS);   % differentiate L w.r.t psi
psiEqS = expand(ddt_dLdpsiDS-dLdpsiS)  % Form the equation for psi
```

Step 10 - Forming the mass matrix and force vector

Now we need for form these two equations into something we can simulate,

$$M(\phi,\theta,\psi)\begin{Bmatrix}\ddot{\phi}\\ \ddot{\theta}\\ \ddot{\psi}\end{Bmatrix} = \overrightarrow{F}(\phi,\dot{\phi},\theta,\dot{\theta},\psi,\dot{\psi},\ddot{x},\ddot{y})$$

```
 [MassMatrix,FVector] = equationsToMatrix([phiEqS, thtEqS, psiEqS],
[phiddS,thtddS,psiddS]); % Convert equations into mass matrix and force vector
 MassMatrix = simplify(MassMatrix);  % Extract the mass matrix
 FVector = simplify(FVector);  % Extract the force vector

 % Here, we auto write functions for computing
 % the mass matrix and the main bits of x' = f(x,u,t)
 matlabFunction(MassMatrix,'File','scripts/mMtx');  % Create a matlab function of mass
matrix
 matlabFunction(FVector,'File','scripts/fVec');  % Create a matlab function of Force
vector
```

5.4 State space matrix

Since we have three generalized coordinates, we will have state space matrix of total 6 variables.

$\boldsymbol{x}_1 = \emptyset$

$\boldsymbol{x}_2 = \boldsymbol{\theta}$

$\boldsymbol{x}_3 = \boldsymbol{\varphi}$

$\boldsymbol{x}_4 = \dot{\boldsymbol{x}}_1 = \dot{\emptyset}$

$\boldsymbol{x}_5 = \dot{\boldsymbol{x}}_2 = \dot{\boldsymbol{\theta}}$

$\boldsymbol{x}_6 = \dot{\boldsymbol{x}}_3 = \dot{\boldsymbol{\varphi}}$

$\boldsymbol{x}_7 = \dot{\boldsymbol{x}}_4 = \ddot{\emptyset}$ = Phi Equation from Live script

$\boldsymbol{x}_8 = \dot{\boldsymbol{x}}_5 = \ddot{\boldsymbol{\theta}}$ = Theta Equation from Live script

$\boldsymbol{x}_9 = \dot{\boldsymbol{x}}_6 = \ddot{\boldsymbol{\varphi}}$ = Psi Equation from Live script

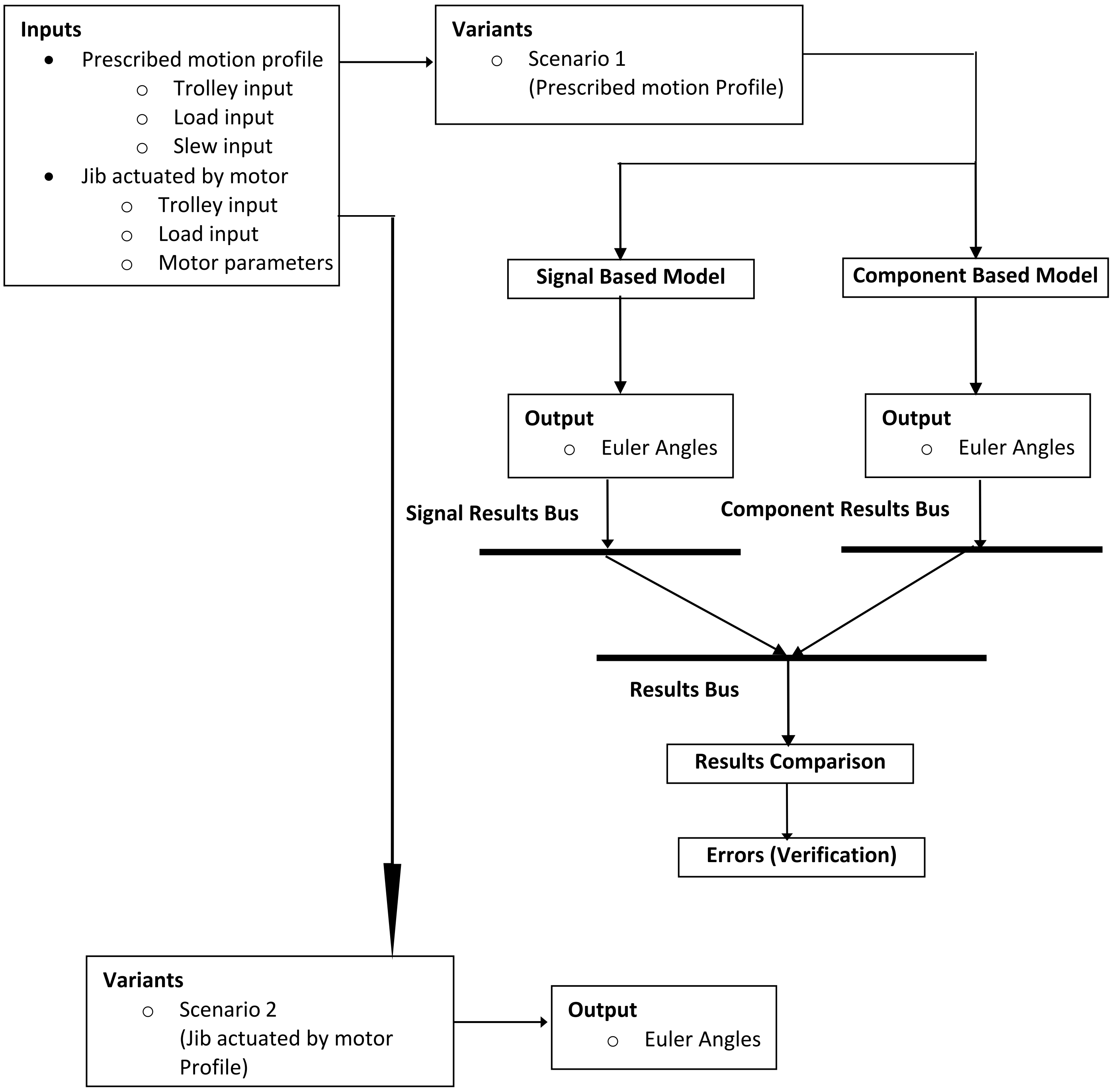
So, state space matrix will be

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\boldsymbol{x}}_1 \\ \dot{\boldsymbol{x}}_2 \\ \dot{\boldsymbol{x}}_3 \\ \dot{\boldsymbol{x}}_4 \\ \dot{\boldsymbol{x}}_5 \\ \dot{\boldsymbol{x}}_6 \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}_4 \\ \boldsymbol{x}_5 \\ \boldsymbol{x}_6 \\ \boldsymbol{x}_7 \\ \boldsymbol{x}_8 \\ \boldsymbol{x}_9 \end{bmatrix}$$

Now, we will enter this state space matrix, force matrix and mass matrix in MatLab function block. The output of MatLab function block will be integrated once so that we will get the values for six variables from $\boldsymbol{x}_1$ to $\boldsymbol{x}_6$.

Chapter 6. Creating a block diagram

In this project, we are going to make the use of variant subsystems in order to jump from scenario 1 to scenario 2. Also, as there are different actuation methods of jib for two scenarios, it will lead to different motions for those two scenarios, so we are also going to make the use of buses in Simulink for carrying those signals.
Let us create the block diagram starting from the inputs to outputs.



Chapter 7. Code description

7.1 Define Inputs, conditions & solver parameters

First of all, let us write the script for clearing the workspace, closing all open figures and opening the relevant Simulink model.
After that, start writing the parameters in the script.

```
Clean up
clearvars;  % Clear workspace
close all % Close all open figures
clc  % Clear command window
Simulink.sdi.clear;  % Clear the simulation data inspector
```

Open the simulink model

```
load_system('project4Model');  % load the simulink model
```

Define the physical constants

```
planet.g = 10; % This is the value of gravitational acceleration (m/s^2)
```

Jib Parameters

```
jib.l   = 3.0;  % length of jib (m)
jib.w   = 0.1;  % width of jib  (m)
jib.rho = 2700; % density of the jib material (kg/m^3)
```

Trolley Parameters

```
trol.l   = 0.2;     % length of trolley (m)
trol.w   = 0.2;     % width of trolley (m)
trol.h   = 0.2;     % height of trolley (m)
trol.rho = 2700;    % density of trolley material (kg/m^3)
trol.clr = [147 174 200]/255; % light blue
```

Load Parameters

```
 lod.l = 0.3;    % length of load block (m)
 lod.w = 0.1;    % width of load block (m)
 lod.h = 0.2;    % height of load block (m)
 lod.rho = 8000; % density load block material (kg/m3);
 lod.v = lod.l*lod.w*lod.h; % volume of the load block
 lod.m = lod.rho*lod.v;  % mass of the load

 lod.Ixx = (1/12)*lod.m*((lod.h^2)+(lod.w^2));  % Mass moment of Inertia of load block
along X-X
 lod.Iyy = (1/12)*lod.m*((lod.l^2)+(lod.w^2));  % Mass moment of Inertia of load block
along Y-Y
 lod.Izz = (1/12)*lod.m*((lod.l^2)+(lod.h^2));  % Mass moment of Inertia of load block
along Z-Z
```

Base Parameters

```
 bas.h = 3.0;  % height (m)
 bas.w = 0.25; % width (m)
```

Motor Parameters

```
 motor.radius = 0.2;  % slew motor disk radius (m)
 motor.height = 0.1;  % slew motor disk height (m)
 motor.rho    = 2700; % density of disk material (kg/m^3)
 motor.m      = pi*motor.radius^2 * motor.height * motor.rho; % mass of the motor
disk(kg)

 % mass moments of inertia (kg-m^2)
 motor.Ixx    = motor.m * ( (motor.radius^2)/4 + (motor.height^2)/12 );  % Mass moment of
Inertia of motor block along X-X
 motor.Iyy    = motor.Ixx;  % Mass moment of Inertia of motor block along Y-Y
 motor.Izz    = motor.m * motor.radius^2 / 2;  % Mass moment of Inertia of motor block
along Z-Z
 motor.Iarm   = 1E-1*100;  % armature inertia in spin axis (kg-m^2)

 motor.L      = 0.1;       % armature inductance (H)
 motor.R      = 1.0;       % armature resistance (ohm)
 motor.b      = 1.0;       % damping (N/m/(rad/s))
 motor.kt     = 7E-1;      % torque constant (N-m/A)
 motor.kb     = 7E-1;      % back emf constant (V/(rad/s))

 motor.V = 50;
```

Initial conditions of trolley and load

```
trol.ics = [2.5 0]; % Initial position and speed of trolley (m, m/s)
lod.ic = -3.0;  % Initial position of the load block (m)
```

Solver Parameters

```
simPrm.h      = 1E-3; % Integration time step (s)
simPrm.solTyp = 'Fixed-step';  % Solver type
simPrm.sol    = 'ode4';  % Integration method
simPrm.tEnd   = 50; % Simulation end time (s)
```

Set Simulink Configuration

```
set_param('project4Model','SolverType',simPrm.solTyp);  % set this solver type in
simulink parameters
set_param('project4Model','Solver'    ,simPrm.sol);     % set this integration method in
simulink solver
```

7.2 Define the buses

Buses are used in Simulink to carry multiple signals which can be utilized for simulating more than one models. In our project, as we need input motions for both the models, we will create the bus for carrying the appropriate signals which are input motions for both scenarios.

Below is the structure of the buses which we have defined in our model

Bus name: Input ← Main top-level bus
Bus elements:
Trolley…………………(Prescribed motion for trolley)
Hoist…………………(Prescribed motion for load)
Slew…………………(Prescribed motion for jib)

Bus name: Results ← Main top-level bus
Bus elements: Bus name : Signal_Results ← 2nd level bus inside top bus
Bus elements:
Phi_S…………………(Angle Phi)
Tht_S…………………(Angle Theta)
Psi_S…………………(Angle Psi)
Bus name : Component_Results ← 2nd level bus inside top bus
Bus elements:
Phi_C…………………(Angle Phi)
Tht_C…………………(Angle Theta)
Psi_C…………………(Angle Psi)

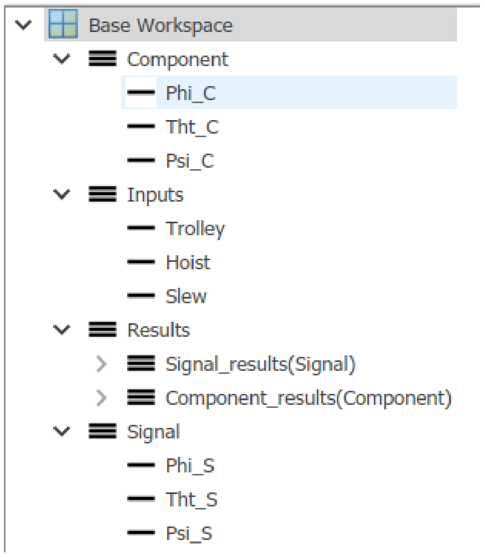

```
Base Workspace
  Component
    Phi_C
    Tht_C
    Psi_C
  Inputs
    Trolley
    Hoist
    Slew
  Results
    Signal_results(Signal)
    Component_results(Component)
  Signal
    Phi_S
    Tht_S
    Psi_S
```

Figure 3: Bus definitions in workspace

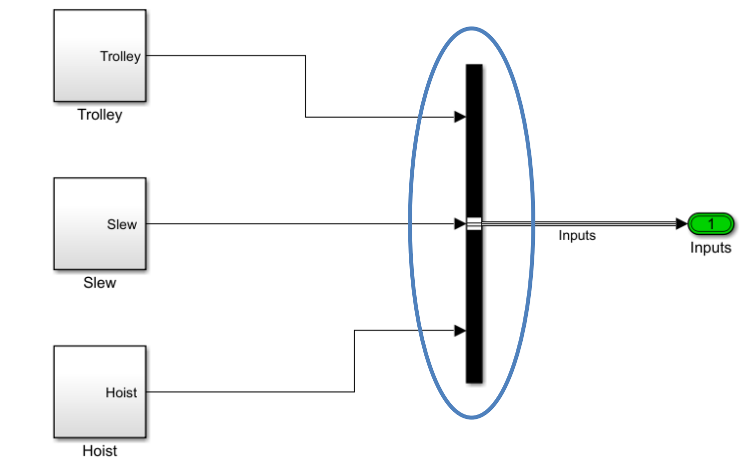


Figure 4: Inputs bus in the model

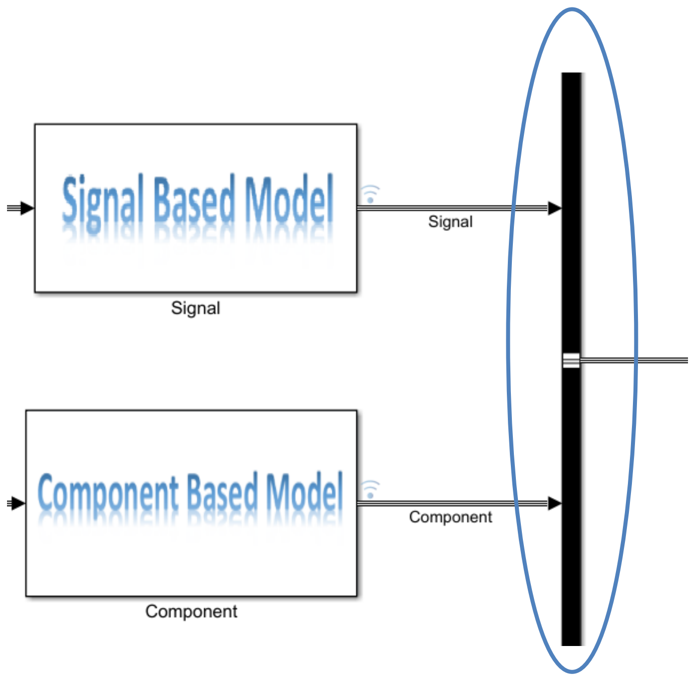


Figure 5: Results/Outputs bus in the model

We will save these bus definitions in .mat format and then load it before running the simulation by entering the command in MatLab script.

Load Bus Definitions

```
load busDefs  % load the buses definitions
```

7.3 Create the variant sub-systems

As earlier said, since there are two scenarios, we need to compute Euler angles for both scenarios The best possible way in this case is two create variant subsystems and switch them accordingly the command given in MatLab script.

7.3.1 What is variant sub-system?

Variant sub-system is simply a block which is used to create a single model that caters the multiple simulation requirements. For example, if we want to simulate a system for two or more types of inputs while allowing to switch between them then we create a variant sub- system which includes the blocks of all those inputs and are executed as per the specific command or scenario defined in the MatLab script.

In our project, we have two types of input profile which we are named as **Prescribed motion** and **Jib actuated by motor**. We are simulating the system for both of these profiles. So, we will create the variant subsystem involving both scenario inputs. Below figure shows the variant subsystem created.

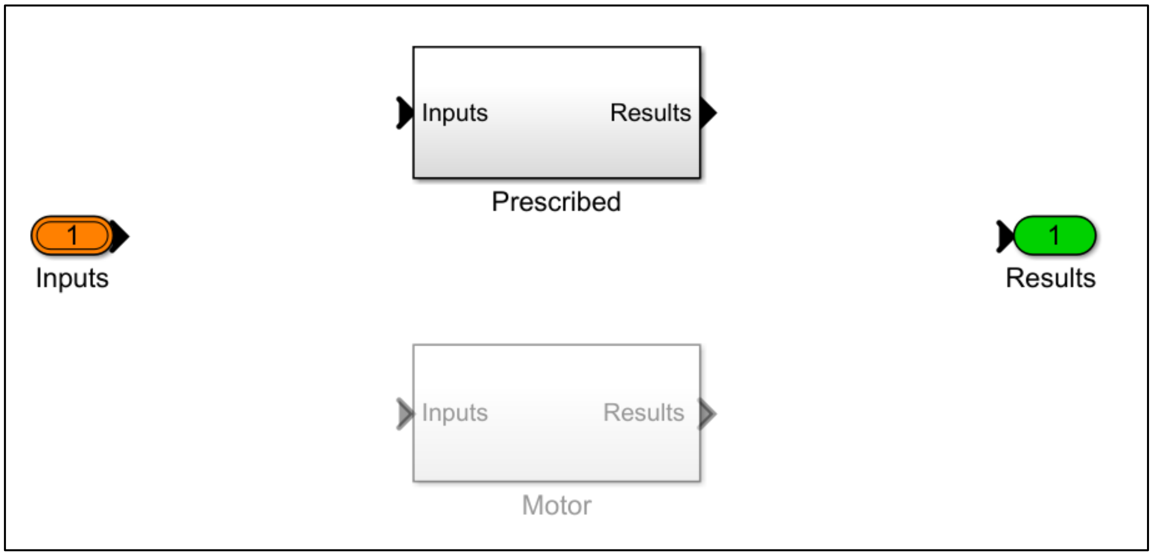


Figure 6: Variant sub-system in the model

Variant subsystems for different thrust scenarios

```
%% Maneuver Variant Subsystem Setup
% SCENARIO = 1, Prescribed motion of jib assembly
% SCENARIO = 2, Jib assembly actuated by motor
Prescribed_results = Simulink.Variant('SCENARIO==1'); % Define the first scenario
Motor_results = Simulink.Variant('SCENARIO==2'); % Define the second scenario
```

7.4 Create the masked sub-system

7.4.1 What is masked sub-system?

A masked subsystem is an interface for a simulink block which hides the logic inside the block and provides the controlled access to the block parameters. Generally, the masked subsystems are created when we intend user not to go in the detail of the logic inside subsystem and can update the output of the block by just changing the few parameters. In our case all three prescribed motions depend on three parameters which are amplitude, pulse duration and coast time. Thus, every time updating the motion does not necessarily require jumping inside the subsystem. If we provide the parameters on the interface of the block, we can easily update the model. Thus, in our case, we are going to create the masked subsystem for both scenarios. Below snapshot shows those subsystems. The downside arrow on the bottom left corner indicates that this is a masked subsystem.

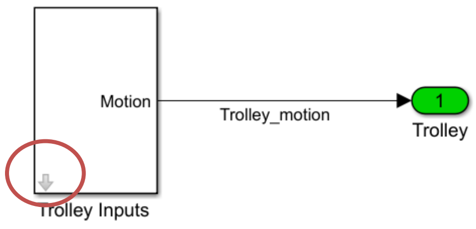


Figure 7A: Masked sub-system for Prescribed motion of trolley

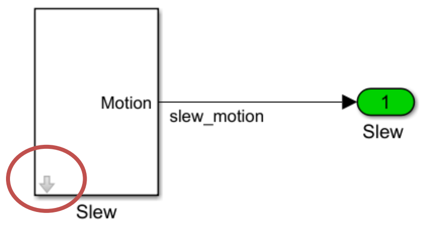


Figure 7B: Masked sub-system for Prescribed motion of jib

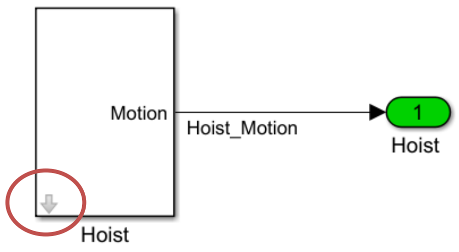


Figure 7C: Masked sub-system for Prescribed motion of Hoist (load)

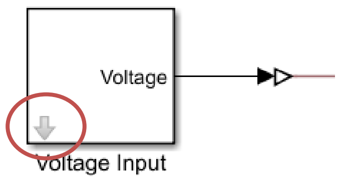


Figure 7D: Masked sub-system for voltage input to DC motor

Figure 7: Masked sub-systems

7.5 Signal-based math model

Below figure shows the signal-based model. It has two MatLab functions.

First MatLab function – for x, y & z states. It takes the inputs of prescribed motions of trolley, hoist and slew and converts it into x, y & z states.

Second MatLab function – for state-space matrix. It takes the outputs from first function as inputs and converts it into the state space matrix. We will see below this MatLab function how the state space matrix is defined.

It gives the output as state space matrix which we discussed earlier.
Thus, there are six outputs $\emptyset, \boldsymbol{\theta}, \boldsymbol{\varphi}\ \dot{\emptyset},\ \dot{\boldsymbol{\theta}}, \dot{\boldsymbol{\varphi}}$

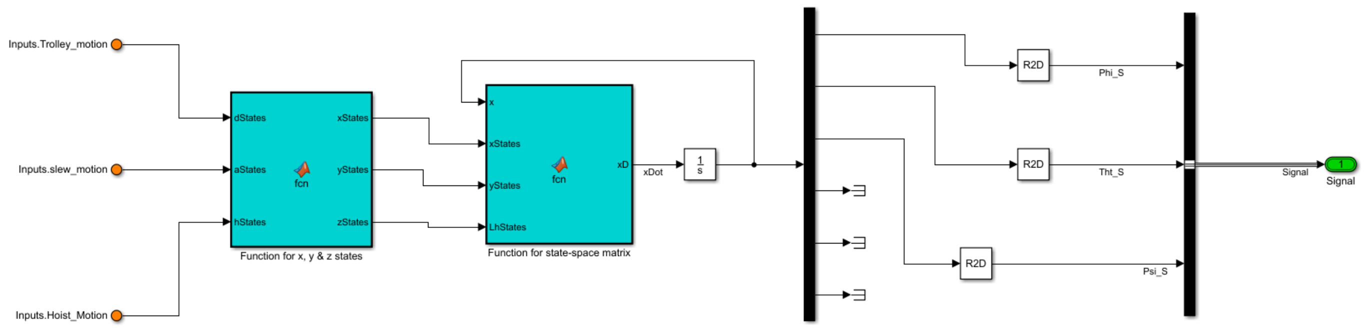
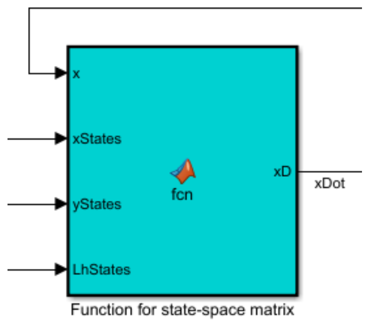
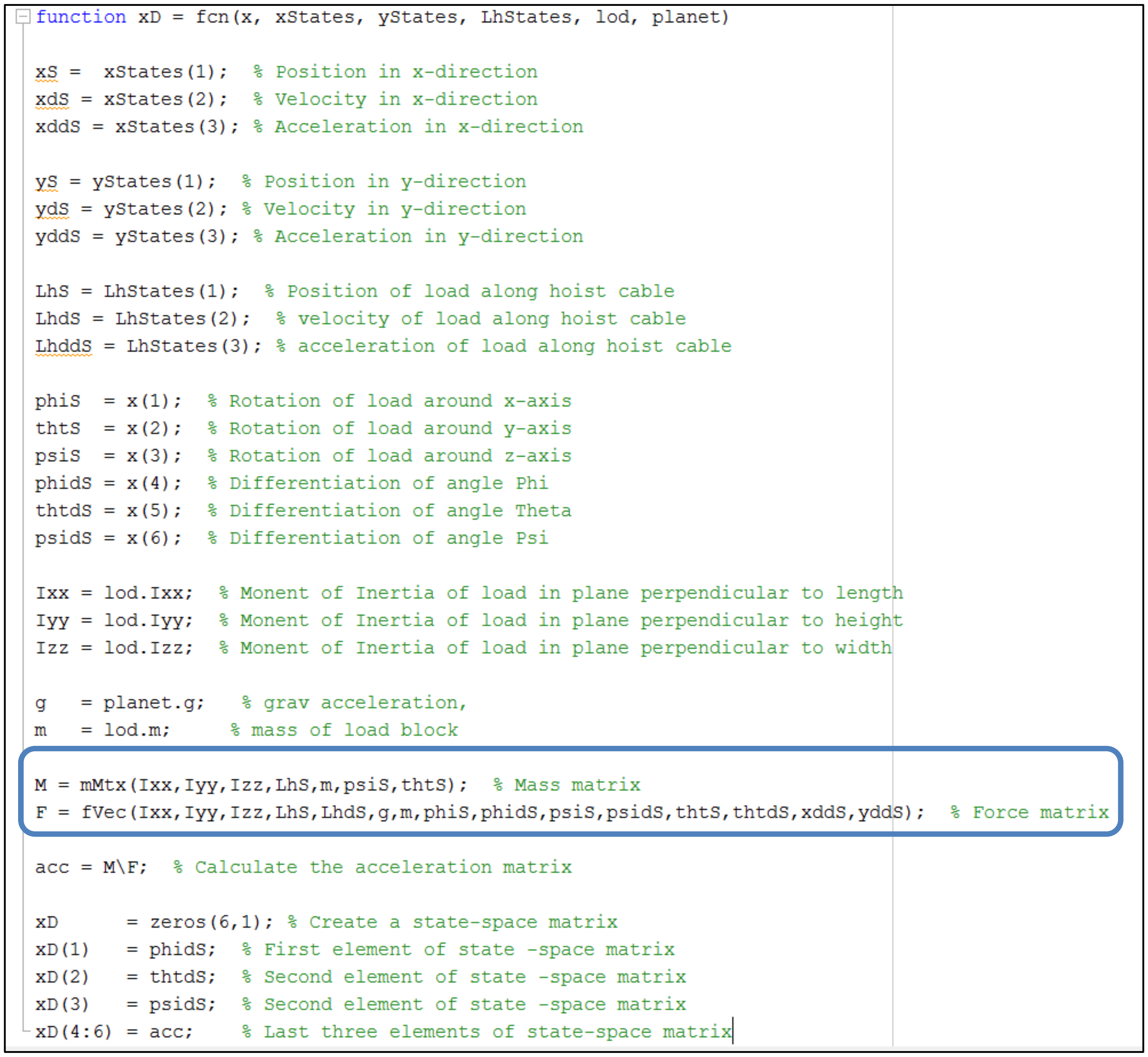


Figure 8: Signals based Simulink model

7.5.1 Inside MatLab function block



As already said in chapter 5.5, we have created a separate mass and force matrix .m file in the MatLab using symbolic toolbox. Also, we have defined several parameters which are computed in main script such as Moment of Inertia of the load, moment of inertial of motor and so on. Below snapshot shows the code written inside the MatLab function block. **We can see that the Force and mass matrices are exactly same which we derived in chapter 5.5.**



```
function xD = fcn(x, xStates, yStates, LhStates, lod, planet)

xS =  xStates(1);  % Position in x-direction
xdS = xStates(2);  % Velocity in x-direction
xddS = xStates(3); % Acceleration in x-direction

yS = yStates(1);  % Position in y-direction
ydS = yStates(2); % Velocity in y-direction
yddS = yStates(3); % Acceleration in y-direction

LhS = LhStates(1);  % Position of load along hoist cable
LhdS = LhStates(2);  % velocity of load along hoist cable
LhddS = LhStates(3); % acceleration of load along hoist cable

phiS  = x(1);  % Rotation of load around x-axis
thtS  = x(2);  % Rotation of load around y-axis
psiS  = x(3);  % Rotation of load around z-axis
phidS = x(4);  % Differentiation of angle Phi
thtdS = x(5);  % Differentiation of angle Theta
psidS = x(6);  % Differentiation of angle Psi

Ixx = lod.Ixx;  % Monent of Inertia of load in plane perpendicular to length
Iyy = lod.Iyy;  % Monent of Inertia of load in plane perpendicular to height
Izz = lod.Izz;  % Monent of Inertia of load in plane perpendicular to width

g   = planet.g;   % grav acceleration,
m   = lod.m;     % mass of load block

M = mMtx(Ixx,Iyy,Izz,LhS,m,psiS,thtS);  % Mass matrix
F = fVec(Ixx,Iyy,Izz,LhS,LhdS,g,m,phiS,phidS,psiS,psidS,thtS,thtdS,xddS,yddS);  % Force matrix

acc = M\F;  % Calculate the acceleration matrix

xD      = zeros(6,1); % Create a state-space matrix
xD(1)   = phidS;  % First element of state -space matrix
xD(2)   = thtdS;  % Second element of state -space matrix
xD(3)   = psidS;  % Second element of state -space matrix
xD(4:6) = acc;    % Last three elements of state-space matrix
```

7.6 Components-based physical model

Below figure shows the component-based model takes the input as prescribed motions of Trolley, load and jib assembly. It gives the output as state space matrix which we discussed earlier.
Thus, there are three outputs $\emptyset, \boldsymbol{\theta}, \boldsymbol{\varphi}$

7.6.1 Description of the model

Joints used:

1. Two Prismatic Joints – To enable the sliding motion of trolley on jib and vertical movement of load
2. One revolute joint – To rotate the jib mounted on tower
3. One Gimbal joint – For three axis rotation of the hanging load

Solids used:

1. Cuboid – To create a tower on which jib is mounted
2. Cuboid – To create a jib
3. Cuboid – To create a load block
4. Cube – To create a trolley
5. Cylinder – To create motor disk

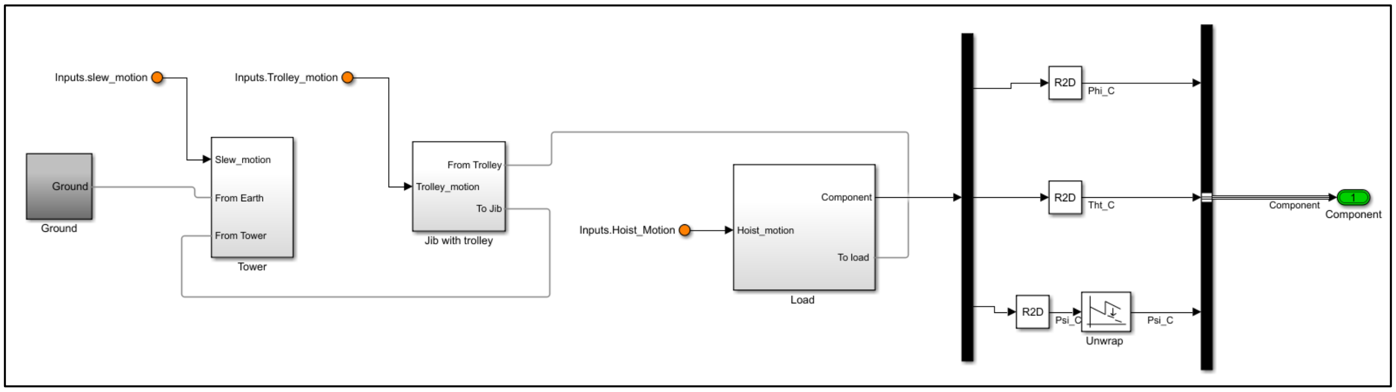


Figure 9: Components based Simscape model

7.7 Running the simulation scenarios & extracting output data

Now, we will run the simulation for both scenarios and then extract the output data which will be useful for plotting the figures and verification purpose.

Simulate Both Scenarios

```
 SCENARIO = 1; % Prescribed motion for jib rotation
 Prescribed_results = sim('project4Model','SignalLoggingName','sdat'); % Simulate first
scenario and save results in prescribed_results

 SCENARIO = 2; % Jib rotation by motor
 Motor_results = sim('project4Model','SignalLoggingName','sdat'); % Simulate second
scenario and save results in Motor_results
```

Extract Data for plotting and printing the results

```
 Component = 1;  % Variable for extracting the results from component based model
 Force = 2;  % Variable for extracting the force results
 Torque = 3; % Variable for extracting the Torque results
 Jib_motion_1 = 4; % Variable for extracting the ang displacement of jib in scenario 1
 Signal = 5;  % Variable for extracting the results from signal based model
 Errors = 6;  % Variable for extracting the error between results from signal based and
component based models
 Angles = 1; % Variable for extracting euler angles in scenario 2
 Jib_motion_2 = 2; % Variable for extracting the ang displacement of jib in scenario 2

 % Scenario 1 Results (Prescribed Motion)
 s1.t = Prescribed_results.tout;  % Save simulation time values in s1.t array
 s1.Phi_S = Prescribed_results.sdat{Signal}.Values.Phi_S.Data(:,1); % Save phi values
from signal based model in s1.Phi_S
 s1.Tht_S = Prescribed_results.sdat{Signal}.Values.Tht_S.Data(:,1); % Save theta values
from signal based model in s1.Tht_S
 s1.Psi_S = Prescribed_results.sdat{Signal}.Values.Psi_S.Data(:,1); % Save psi values
from signal based model in s1.Psi_S
 s1.Phi_C = Prescribed_results.sdat{Component}.Values.Phi_C.Data(:,1); % Save phi values
from component based model in s1.Phi_C
 s1.Tht_C = Prescribed_results.sdat{Component}.Values.Tht_C.Data(:,1); % Save theta
values from component based model in s1.Tht_C
 s1.ThtDE = Baseline_Result.sdat{Errors}.Values.Data(:,4);  % Save error in ang velocity
of rod
 s1.f = Baseline_Result.sdat{Thrust}.Values.Data(:,1);  % Save thrust magnitude of
baseline profile
 s1.Psi_C = Prescribed_results.sdat{Component}.Values.Psi_C.Data(:,1);  % Save psi values
from component based model in s1.Psi_C
 s1.Phi_E = Prescribed_results.sdat{Errors}.Values.Data(:,1); % Save error in Phi in
s1.Phi_E
```

```
 s1.Tht_E = Prescribed_results.sdat{Errors}.Values.Data(:,1); % Save error in Phi in
s1.Tht_E
 s1.Psi_E = Prescribed_results.sdat{Errors}.Values.Data(:,1); % Save error in Phi in
s1.Psi_E
 s1.Force = Prescribed_results.sdat{Force}.Values.Data(:,1); % Save trolley force in
s1.Force
 s1.Torque = Prescribed_results.sdat{Torque}.Values.Data(:,1); % Save torque on jib in
s1.Torque
 s1.motion = Prescribed_results.sdat{Jib_motion_1}.Values.Data(:,1); % Save ang
displacement of jib of scneario 1 in s1.motion

 % % Scenario 2 Results (Motor)
 s2.t = Motor_results.tout;  % Save simulation time values in s2.t array
 s2.Phi = Motor_results.sdat{Angles}.Values.Data(:,1); % Save phi values in s2.Phi
 s2.Tht = Motor_results.sdat{Angles}.Values.Data(:,2); % Save theta values in s2.Tht
 s2.Psi = Motor_results.sdat{Angles}.Values.Data(:,3); % Save psi values in s2.Psi
 s2.motion = Motor_results.sdat{Jib_motion_2}.Values.Data(:,1); % Save ang displacement
of jib of scneario 2 in s2.motion
```

Chapter 8. Results and verification

8.1 Prescribed motion profile results

After, we run the simulation and extract the data with the help of the indices, we will make several plots which are –

- Output results of Euler angles from signal based and component-based models for prescribed motion profile
- Errors between signal based and component-based models for prescribed motion case
- Output results of Euler angles from signal based and component-based models for second scenario

Plot the values of all parameters vs time for scenario 1

```
% % Plot the all parameters vs time for first scenarios
figure(1);
subplot(3,1,1);  % Plot first figure in a subplot of 3 rows and 1 column
plot(s1.t,s1.Phi_S,'k'); % Plot time vs Phi-signal for first scenario
hold on
plot(s1.t,s1.Phi_C,'r'); % Plot time vs Phi-component for first scenario
axis([0 50 -60 60]);  % Give the limits of both axis
legend('\phi signal based','\phi Component based'); % Give the legends
title("Angle \phi vs Time"); % Give the title
xlabel('Time (sec)');  % Give X label as Time in seconds
ylabel('Angle \phi (degrees)');  % Give Y label as angle in degrees
subplot(3,1,2); % Plot second figure in a subplot of 3 rows and 1 column
plot(s1.t,s1.Tht_S,'k'); % Plot time vs Theta-signal for first scenario
hold on
plot(s1.t,s1.Tht_C,'r');  % Plot time vs Theta-component for first scenario
axis([0 50 -80 80]);  % Give the limits of both axis
legend('\theta signal based','\theta Component based'); % Give the legends
title("Angle \theta vs Time"); % Give the title
xlabel('Time (sec)');  % Give X label as Time in seconds
ylabel('Angle \theta (degrees)');  % Give Y label as angle in degrees
subplot(3,1,3); % Plot third figure in a subplot of 3 rows and 1 column
plot(s1.t,s1.Psi_S,'k'); % Plot time vs Psi-signal for first scenario
hold on
plot(s1.t,s1.Psi_C,'r'); % Plot time vs Psi-component for first scenario
axis([0 50 -60 360]);  % Give the limits of both axis
legend('\psi signal based','\psi Component based'); % Give the legends
title("Angle \psi vs Time"); % Give the title
xlabel('Time (sec)');  % Give X label as Time in seconds
ylabel('Angle \psi (degrees)');  % Give Y label as angle in degrees

sgtitle('Euler angles vs Time for prescribed motion scneario from singal and component
models'); % Create the main title
```

<u>Results</u>:

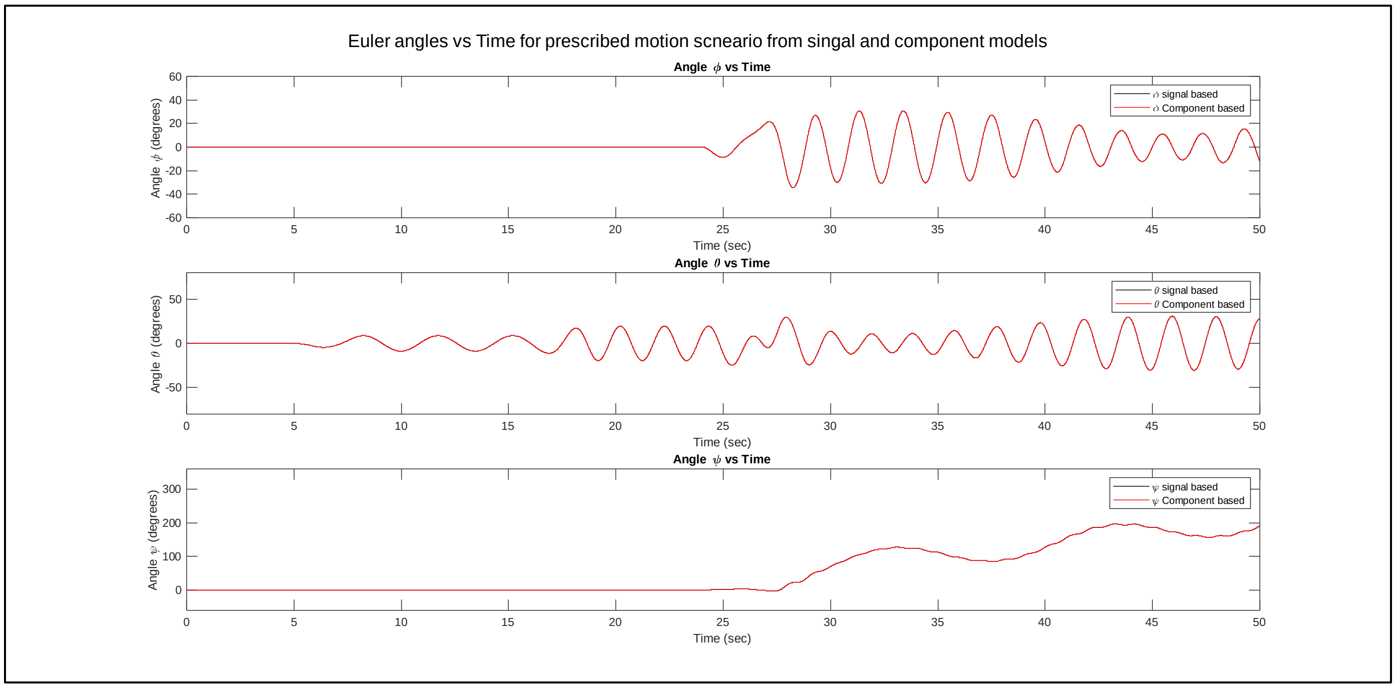


Figure 10: Results for Prescribed motion (Scenario 1)

The above subplots show the values of all three Euler angles at different instance of time from signal based as well as component-based models for prescribed motion profile (scenario 1).

<u>Conclusion</u>:

From the above plot, it can be seen that the output values from signal based and component-based models overlap on each other and thus there is very minimum error between them. ***<u>We will have the error values table shown in section 8.6 - printing the results.</u>***

8.2 Computation of force on trolley and torque on jib for scenario 1

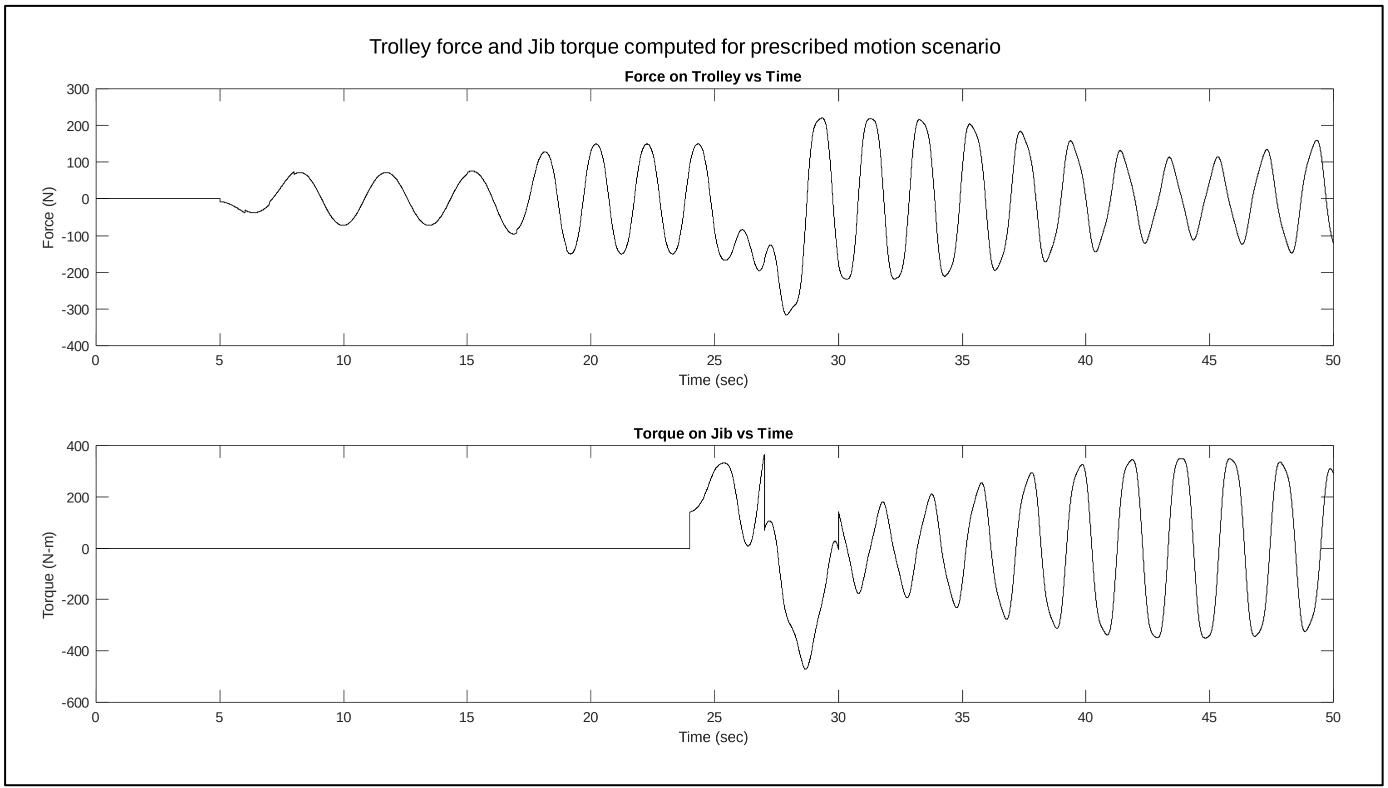


Figure 11: Computation of Force and Torque (Scenario 1)

Conclusion:

The first figure in the above plot shows the force on the trolley which has prescribed translational motion with respect to time. This force is automatically computed by Simscape after we check the box for sensing of force in prismatic joint used for trolley. The first jump on the negative scale is because the positive-x direction of the trolley is away from the tower and as we saw in the simulation, the trolley moves in the direction towards the tower so the force being in negative x direction has negative magnitude at the start.

The second figure in the above plot shows the torque on the jib which has prescribed rotating motion with respect to time. The torque value jumps to positive value at 24 seconds as we have prescribed motion starting at that time. This torque is automatically computed by Simscape after we check the box for sensing of torque in revolute joint used for jib.

8.3 Verification of signal-based model with component-based model

Plot the values of errors in Euler angles for scenario 1

```
% Plot the errrors in all parameters vs time for first scenario
figure(2);

subplot(3,1,1); % Plot first figure in a subplot of 3 rows and 1 column
plot(s1.t,s1.Phi_E,'b');  % Plot time vs Error in Phi
title("Error in Angle \phi vs Time"); % Give the title
xlabel('Time (sec)');  % Give X label as Time in seconds
ylabel('Error Angle \phi (degrees)');  % Give Y label as error in degrees

subplot(3,1,2); % Plot second figure in a subplot of 3 rows and 1 column
plot(s1.t,s1.Tht_E,'b'); % Plot time vs Error in Theta
title("Error in Angle \theta vs Time"); % Give the title
xlabel('Time (sec)');  % Give X label as Time in seconds
ylabel('Error Angle \theta (degrees)');  % Give Y label as error in degrees

subplot(3,1,3);  % Plot Third figure in a subplot of 3 rows and 1 column
plot(s1.t,s1.Psi_E,'b');   % Plot time vs Error in Psi
title("Error in Angle \psi vs Time"); % Give the title
xlabel('Time (sec)');  % Give X label as Time in seconds
ylabel('Error Angle \psi (degrees)');  % Give Y label as error in degrees

sgtitle('Verification - Error in Euler angles computed from signal and component
models'); % Create the main title
```

Results:

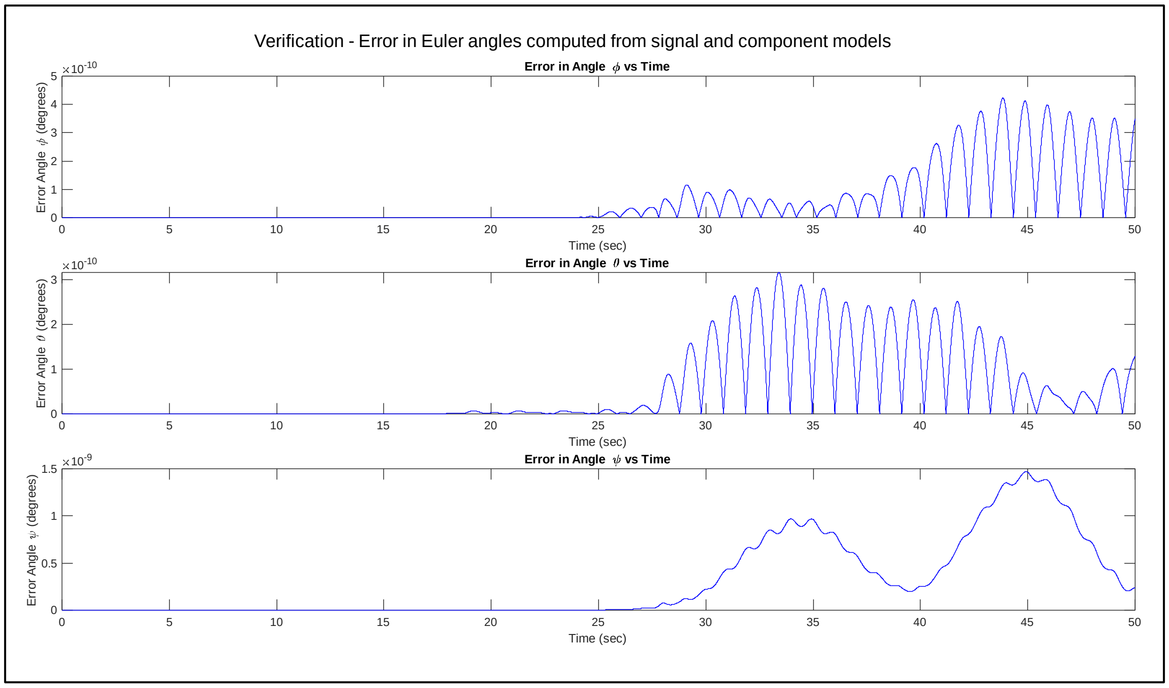


Figure 12: Errors in Euler angles computation between Signal based and component-based model for scenario 1

The above subplots show the differences calculated for Euler angles computed from signal based and component-based models for scenario 1.

Conclusion:

From the above plots, it can be seen that there is very minimum error between the outputs from signal based and component-based models. We can say that both the models are implemented correctly ***We will have the error values table shown in section 8.6 - printing the results.***

8.4 Jib actuated by motor scenario-2 results

As a part of this project requirements, we have to replace the actuation of jib assembly with motor Thus, the voltage input given to motor will cause the motor shaft to rotate which will cause jib assembly to rotate. Our intention is to have the same motion of jib assembly as we observed in prescribed motion case. In order to achieve the same motion profile, we have started with step command of voltage input and thus tuned the parameters until we achieve the motion profile closer to what we have in prescribed motion profile. Below are the final tuned parameters.

Parameters of Input profile of voltage		
Parameter	**Value**	**Units**
Positive Voltage amplitude	**120.0**	V
Start Time	**24.0**	sec
Duration of pulse	**2.1**	sec
Negative Voltage amplitude	**-13.8**	V
Stop time	**44.0**	sec

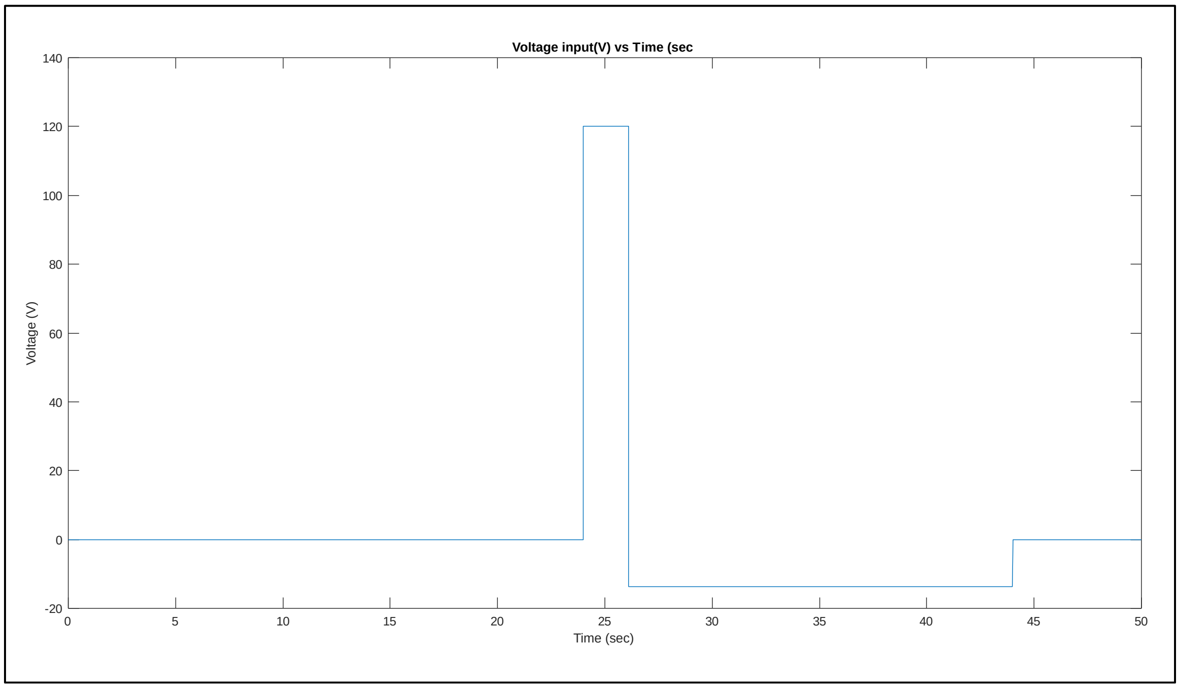


Figure 13: Voltage profile for DC motor Input

```
% % Plot the euler angles vs time for second scenario
figure(4);
subplot(3,1,1); % Plot first figure in a subplot of 3 rows and 1 column
plot(s2.t,s2.Phi,'k');  % Plot time vs Phi for second scenario
title("Angle Phi vs Time"); % Give the title
xlabel('Time (sec)');  % Give X label as Time in seconds
ylabel('Angle Phi (degrees)');  % Give Y label as angles in degrees
subplot(3,1,2);  % Plot second figure in a subplot of 3 rows and 1 column
plot(s2.t,s2.Tht,'k');  % Plot time vs Theta for second scenario
title("Angle Theta vs Time"); % Give the title
xlabel('Time (sec)');  % Give X label as Time in seconds
ylabel('Angle Theta (degrees)');  % Give Y label as angles in degrees
subplot(3,1,3);  % Plot third figure in a subplot of 3 rows and 1 column
plot(s2.t,s2.Psi,'k');  % Plot time vs Psi for second scenario
title("Angle Psi vs Time"); % Give the title
xlabel('Time (sec)');  % Give X label as Time in seconds
ylabel('Psi (degrees)');  % Give Y label as angles in degrees
sgtitle('Euler angles vs Time for jib actuated by motor - Scenario 2'); % main title
```

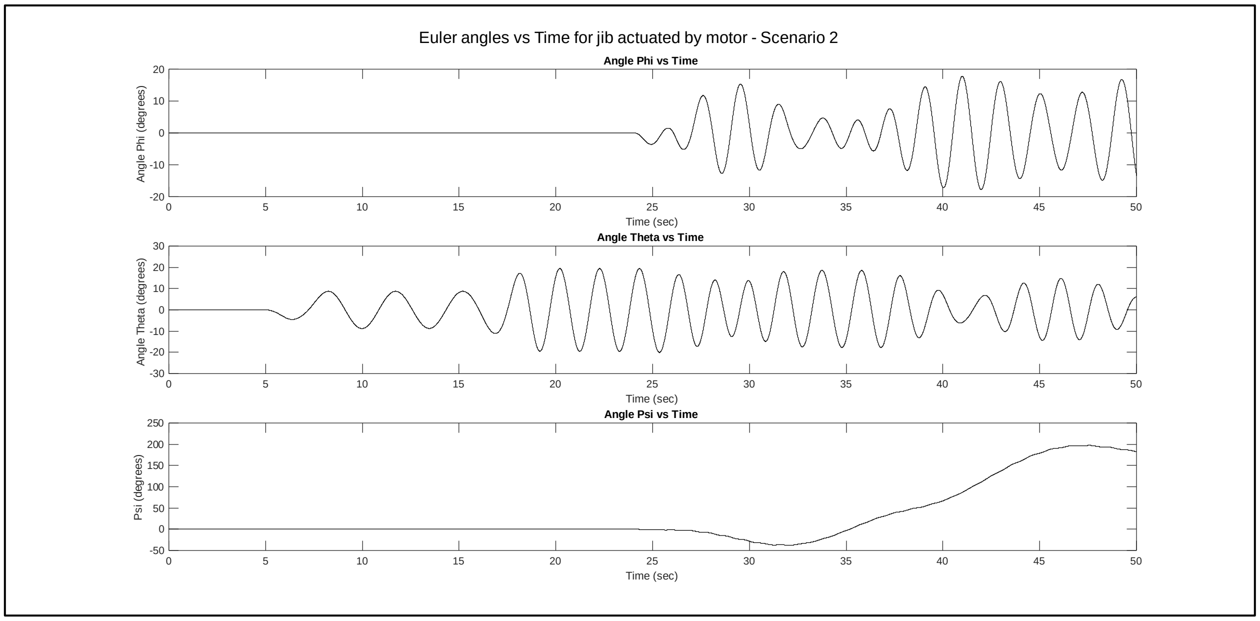


Figure 14: Computation of Euler angles for scenario 2- Jib actuated by motor

<u>Conclusion</u>:

The above plot shows the Euler angles $\emptyset, \boldsymbol{\theta}, \boldsymbol{\varphi}$ computed for scenario 2 where jib assembly is actuated by the motor. If we compare this plot with the scenario 1 outputs, we can say that Euler angles in scenario 2 are close to what we have in scenario 1. Thus, we can infer that the motor actuation is replacing the prescribed motion of jib successfully.

8.5 Motion of jib actuated by motor

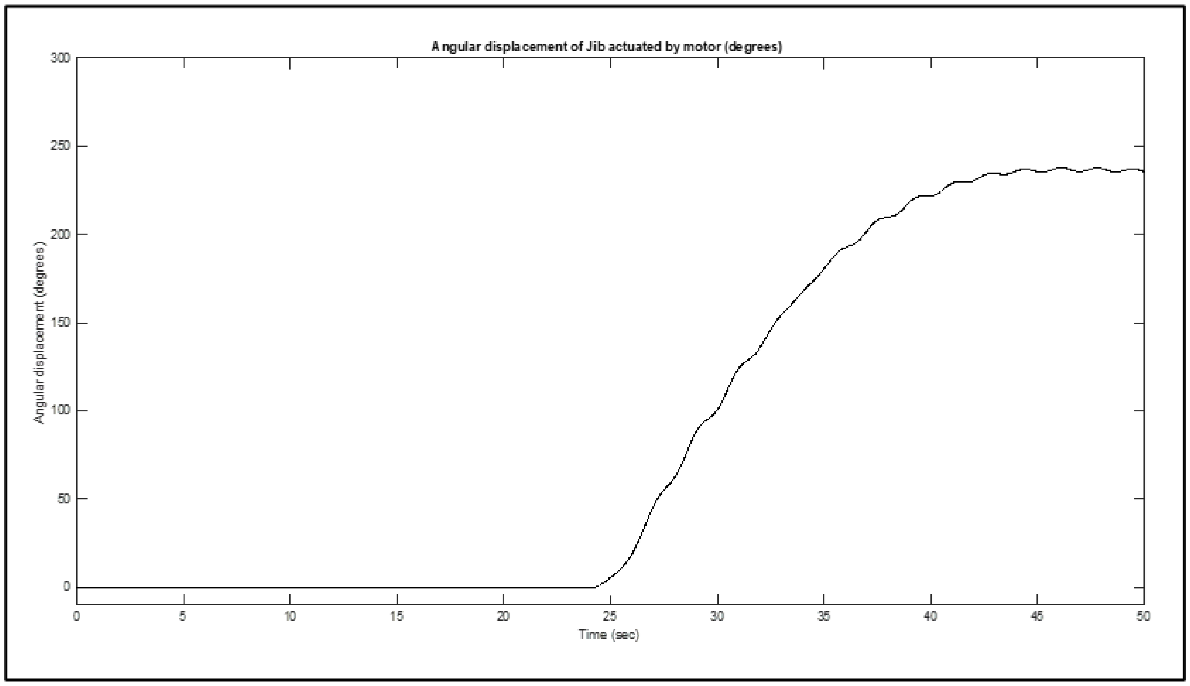


Figure 15: Motion of jib actuated by motor

Conclusion:

The above plot shows motion of jib (angular displacement in degrees) when it is actuated by the motor. Some of the inferences that can be drawn from this plot are

- The motion of the jib starts at the same time (24 secs) as we have in prescribed motion.
- The total angular displacement is near to what we have in prescribed motion scenario.
- After 44 seconds, there is no further angular displacement of the jib.
- The oscillations in the graph at steady state are because the hanging load keeps on moving so it makes jib to vibrate.

8.6 Printing the results & conclusion

Once we run the simulation, we will get the maximum error for all parameters and we will print those values. Accordingly, we will compare those values with the threshold value we have provided in table above (also included in the MatLab script) and then print the corresponding conclusion. Below code is written for printing those errors and conclusion.

```
 fprintf('<strong>
</strong>\n');
 fprintf('<strong>Results of Prescribed - Refer Figure 1 subplots for parameters
</strong>\n');
 fprintf('<strong>---------------------------------------------------------------------------
-----------------------------------------</strong>\n');
 fprintf('<strong>Verification of model accuracy for Prescribed motion - Refer Figure 2
subplots for errors </strong>\n');
 fprintf('Max error in Phi for component based vs signal based model = %f
\n',max(s1.Phi_E)); % Print maximum error in Phi for first scenario
 fprintf('Max error in Theta for component based vs signal based model = %f
\n',max(s1.Tht_E)); % Print maximum error in Theta for first scenario
 fprintf('Max error in Psi for component based vs signal based model  = %f
\n',max(s1.Psi_E)); % Print maximum error in Psi for first scenario
 fprintf('<strong>
</strong>\n');
 if max(s1.Phi_E) < 10e-10  && max(s1.Tht_E) < 10e-10  && max(s1.Psi_E) < 10e-10    %
Print the conclusion
     fprintf('<strong>Both model results are identical so both models are implemented
correctly for prescribed motion; Refer figure 2</strong>\n');
 else
     fprintf('<strong>Both model results are not identical</strong>\n');
 end
 fprintf('<strong>
</strong>\n');
 fprintf('<strong>---------------------------------------------------------------------------
-----------------------------------------</strong>\n');
 fprintf('<strong>Trolley force and Jib torque for Prescribed motion - Refer Figure 3
subplots </strong>\n');

 fprintf('<strong>---------------------------------------------------------------------------
-----------------------------------------</strong>\n');
 fprintf('<strong>Results of Motor actuated jib - Refer Figure 4 subplots for parameters
</strong>\n');
 fprintf('<strong>---------------------------------------------------------------------------
-----------------------------------------</strong>\n');

 fprintf('<strong>Motion of Jib actuated by motor - Refer Figure 5 </strong>\n');
 fprintf('<strong>---------------------------------------------------------------------------
-----------------------------------------</strong>\n');
```

Results:

Maximum errors between signal based and component-based results			
Parameter	**Value**	**Threshold**	**Conclusion**
Error in angle phi $\emptyset$	4.2278e-10	1e-08	**Pass**
Error in angle theta $\boldsymbol{\theta}$	3.1644e-10	1e-08	**Pass**
Error in angle phi $\boldsymbol{\varphi}$	1.4697e-09	1e-08	**Pass**

Conclusion:

We have seen that errors are plotted as a function of time for all three Euler angles of scenario 1 in previous section. We have listed the maximum error which is occurring at particular instance of time for every Euler angle in above table, compared with the threshold value and then drawn the conclusion either 'Pass' or 'Fail'. ***Since all the values are less than the threshold value, we have passed the accuracy level criteria of verification of the model.***

Chapter 9. Comparison of different time steps and integration methods

Why did we select the 0.001 integration step size and ode4 solver?

One of the more critical challenges in solving the system of ordinary differential equations lies in determining the optimum integration step size and integration method. As we go on reducing the step size and high order integration method, with no doubt improve the accuracy of the solution, however that is achieved by compromising the computational time of the solution. Thus, there has to be trade-off between accuracy of the solution and the computational time. Since, the system we are dealing with is non-linear, we will start with 0.01 s integration step size and Ode3 as solver. We will examine the accuracy of simulation in terms of the errors in Euler angles and computational time and will select best option. Below table shows our analysis for different time step size and different solvers.

- We are getting least accuracy in option 1 so we will not consider that. When we choose Ode4 solver with same step size then the accuracy is less than our threshold, computational time is not significantly increased, so we cannot consider this option.
- When we reduce the step size to 10% of original and switch back to Ode3 solver, the errors are more than the threshold value, however computational time is almost doubled. If we compare this with last option where we again select Ode4 as solver, then computational time is not much changed as compared to option 3 but the accuracy has significantly improved. Thus, we consider option 4 (highlighted) as the best option for combination of integration step size and the solver.

Maximum errors between signal based and component-based results				
Sr. No	**Integration step size**	**Solver**	**Maximum error $\emptyset, \theta, \varphi$**	**Computational Time**
1	0.01 s	Ode3	0.0033	2.896 s
			0.0017	
			0.0109	
2	0.01 s	Ode4	5.9973e-06	3.603 s
			5.3655e-06	
			2.5195e-05	
3	0.001 s	Ode3	3.3686e-06	6.633 s
			1.6921e-06	
			1.1009e-05	
4	**0.001 s**	**Ode4**	**4.2278e-10**	**7.921 s**
			3.1644e-10	
			1.4697e-09	

Chapter 10. Future Improvements

So far, we have simulated the jib and crane system with two different types of the inputs and also have verified the simulation with component-based model. Now, let's discuss what could be the potential future improvements we can think of.
The improvements or scope which will be listed here will provide an essence of promoting the thinking how far can we go deep in our simulation and overall learning. Below are the points which are summarized for the future improvements in the model.

Scaling the physics behind the simulation

1) Since we have not captured the frictional forces (between sliding trolley and the jib) in the model, we can include those in the simulation.

2) We can have actual gearbox implemented inside the motor disk and optimize the design for achieving the desired results.

Digging deeper in the integration of multi-physics components with multi-body

1) In the scenario 2 of the project, we have actuated the jib with the electrical motor, where we saw that DC motor is used for actuation. Now, as we have integrated electrical component with our multi-body component, we can follow similar approach for using other multi-physics components such as hydraulic motors, mechanical rotational elements.

2) We then can compare different parameters of this different types of actuation methods for study purpose. For example, the parameters may include the power consumption for different actuators for achieving same output motion etc.

Developing the controls

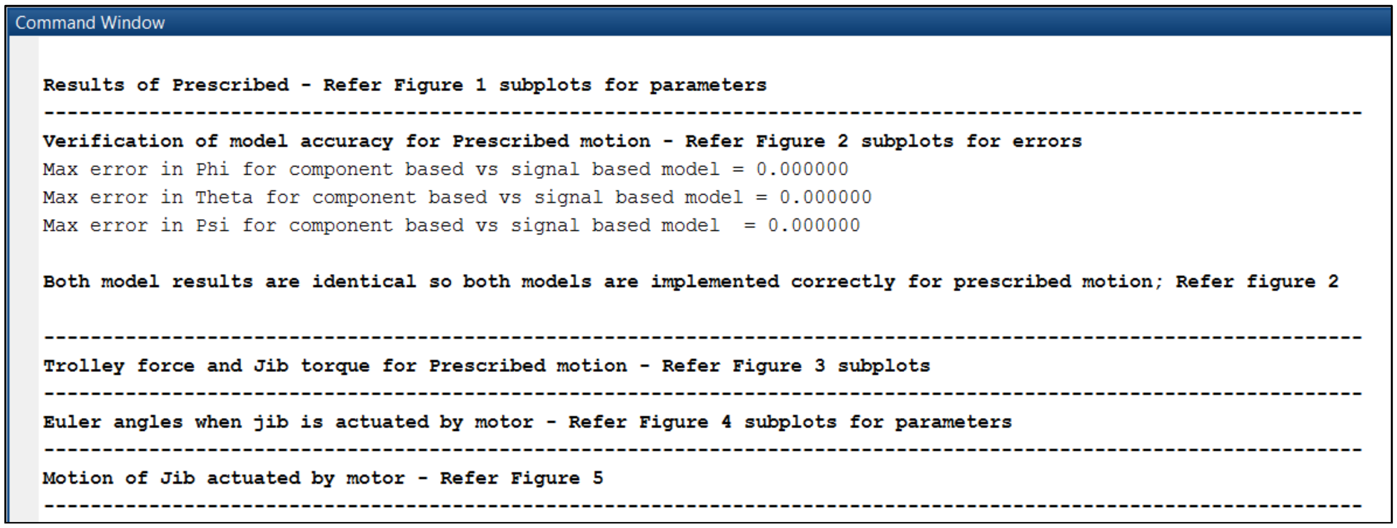
1) We can think of implementing a control (let say PID) for controlling the motions when they are not provided with prescribed profile. This way would provide us more insights on how PID control works, what are its different parameters and how can it be designed for optimum desired response.

Chapter 11. What will be there in command window after running the simulation

Once the simulation run is completed, we will see following data in the command window which is printed from the MatLab script.
The statements in bold letters are the decisions made in the script based upon the comparison of magnitude of errors with the threshold values.

- Errors between all three Euler angles for scenario 1 (Verification)

- Inference drawn regarding the model verification from the observed errors



```
Results of Prescribed - Refer Figure 1 subplots for parameters
---------------------------------------------------------------------------------------------------------------
Verification of model accuracy for Prescribed motion - Refer Figure 2 subplots for errors
Max error in Phi for component based vs signal based model = 0.000000
Max error in Theta for component based vs signal based model = 0.000000
Max error in Psi for component based vs signal based model  = 0.000000

Both model results are identical so both models are implemented correctly for prescribed motion; Refer figure 2

---------------------------------------------------------------------------------------------------------------
Trolley force and Jib torque for Prescribed motion - Refer Figure 3 subplots
---------------------------------------------------------------------------------------------------------------
Euler angles when jib is actuated by motor - Refer Figure 4 subplots for parameters
---------------------------------------------------------------------------------------------------------------
Motion of Jib actuated by motor - Refer Figure 5
---------------------------------------------------------------------------------------------------------------
```

Chapter 12. Conclusion

From all the results shown above, we can infer that both signal-based model and component-based model are implemented correctly. However, what are those aspects we can learn from overall execution of this project is also important affair to discuss. ***Let us discuss those main aspects***.

Learning from Differential equation derivation using symbolic toolbox:

In this project, we approached another way of deriving the differential equations of system which is using symbolic toolbox. It can be said that it is fairly easy to derive those equations once we determine our inputs, generalized coordinates and the position, velocity vectors as MatLab does the rest.

The important point to be emphasized here that this way is much helpful for system involving the bigger complex calculations.
We also learnt how we can formulate the matrices from the equations and how can we create a MatLab function for those matrices which can be used later in our simulation.

Learning from Simscape multibody (Component based model):

Let us take a look at the comparison of the results between signal-based model and component-based model again. We have very minimum error between them. What made it possible? Well, yes, the proper formulation of the differential equations of the system made it possible but the way we have constructed the model in Simscape is also of equal importance. Because we do not require any differential equation for Simscape model, so it gives the results wholly based on the way we construct it.

The use of several joints and the transform sensor has provided us the vast information about the parameters of those joints and transform block. For instance, we learnt that there is more than one possible way of doing rigid transform either by axis alignment or rotation sequence or rotation matrix. We also learnt the types of joints, joints primitives and directionality.
We came to know about the computation of quaternions, direction cosine matrix and its relation to the Euler angles.

In this project, we had an opportunity to study on the integration of the multi-physics block with the multi-body component.

Overall Inference:

At the end, we can conclude that the results of building the models in Simscape with joints and links are itself evidence of correct identification of the kinematics involved in the given problem statement of the project. This and the learning mentioned in above points can act as a reference in the execution of future projects.

References:

1. **Canvas Notes - MEEM 4730 – Dr. Gordon Parker, Michigan Technological University.**

2. Analytical Dynamics by Haim Baruh.

3. Control Tutorials for MatLab and Simulink by Carnegie Mellon University and University of Michigan.

4. Simscape Multibody documentation – MathWorks.