MODELLING AND CONTROL OF WHEEL SHIMMY

Modelling and control of multibody mechanical systems

Author:
George Panagiotou

Date: September 26, 2024

1 Equations of Motion

Conversion between Cartesian and polar unit vectors:

$$\mathbf{e_r} = \cos\delta\,\mathbf{i} + \sin\delta\,\mathbf{j}, \tag{1}$$
$$\mathbf{e}_\theta = -\sin\delta\,\mathbf{i} + \cos\delta\,\mathbf{j}, \tag{2}$$
$$\mathbf{i} = \cos\delta\mathbf{e_r} - \sin\delta\mathbf{e}_\theta, \tag{3}$$
$$\mathbf{j} = \cos\delta\mathbf{e}_\theta + \sin\delta\mathbf{e_r}, \tag{4}$$
$$\mathbf{k} = \mathbf{k}. \tag{5}$$

The position vector of the center of the mass is given by:

$$\mathbf{r} = x\mathbf{i} - f\,\mathbf{e_r} + y\mathbf{j} = (x - f\cos\delta)\mathbf{i} + (y + \sin\delta)\mathbf{j}. \tag{6}$$

To find the Velocity vector we need to differentiate the position vector:

$$\dot{\mathbf{r}} = \dot{x}\mathbf{i} - f\dot{\delta}\mathbf{e}_\theta + \dot{y}\mathbf{j} = (\dot{x} + f\dot{\delta}\sin\delta)\mathbf{i} + (\dot{y} - f\dot{\delta}\cos\delta)\mathbf{j}. \tag{7}$$

To find the acceleration vector we differentiate the velocity vector:

$$\ddot{\mathbf{r}} = \ddot{x}\mathbf{i} - f\ddot{\delta}\mathbf{e}_\theta + f\dot{\delta}^2\mathbf{e_r} + \ddot{y}\mathbf{j} = (\ddot{x} + f\ddot{\delta}\sin\delta + f\dot{\delta}^2\cos\delta)\mathbf{i} + (\ddot{y} - f\ddot{\delta}\cos\delta + f\dot{\delta}^2\sin\delta)\mathbf{j}. \tag{8}$$

Motion of the center of mass:

$$\sum F = m\ddot{r} = F_x\mathbf{i} - ky\mathbf{j} + Y\mathbf{e}_\theta = (F_x - Y\sin\delta)\mathbf{i} + (Y\cos\delta - ky)\mathbf{j}, \tag{9}$$
$$m\ddot{r} = (F_x - Y\sin\delta)\mathbf{i} + (Y\cos\delta - ky)\mathbf{j}. \tag{10}$$

We substitute the acceleration vector (equation 8) into the equation 10 and we create 2 different equations by separate them by their unit vector.

$$m(\ddot{x} + f\ddot{\delta}\sin\delta + f\dot{\delta}^2\cos\delta) = F_x - Y\sin\delta, \tag{11}$$
$$m(\ddot{y} - f\ddot{\delta}\cos\delta + f\dot{\delta}^2\sin\delta) = Y\cos\delta - ky. \tag{12}$$

Equations for motion about the center of mass:

$$I_{zz}\ddot{\delta}\mathbf{k} = f\,\mathbf{e_r} \times (F_X\mathbf{i} - ky\mathbf{j}) - (e - f)\mathbf{e_r} \times Y\mathbf{e}_\theta, \tag{13}$$

$$I_{zz}\ddot{\delta}\mathbf{k} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ f\cos\delta & f\sin\delta & 0 \\ F_x & -ky & 0 \end{vmatrix} - (e - f)Y, \tag{14}$$

$$I_{zz}\ddot{\delta} = -fky\cos\delta - F_x f\sin\delta - (e - f)Y. \tag{15}$$

Equations for lateral and longitudinal velocities:

$$\mathbf{v}_{\text{lat}} = \dot{\mathbf{r}_\mathbf{g}} \cdot \mathbf{e}_\theta, \tag{16}$$

$$\mathbf{v}_{\text{long}} = \dot{\mathbf{r}_\mathbf{g}} \cdot \mathbf{e_r}, \tag{17}$$

where $\dot{\mathbf{r_g}}$ is the velocity vector of the contact point with the ground:

$$\dot{\mathbf{r_g}} = \dot{x}\mathbf{i} - e\dot{\delta}\mathbf{e}_\theta + \dot{y}\mathbf{j} = \dot{x}(\cos\delta\mathbf{e_r} - \sin\delta\mathbf{e}_\theta) - e\dot{\delta}\mathbf{e}_\theta + \dot{y}(\cos\delta\mathbf{e}_\theta + \sin\delta\mathbf{e_r}), \tag{18}$$

$$\dot{\mathbf{r_g}} = (-e\dot{\delta} - \dot{x}\sin\delta + \dot{y}\cos\delta)e_\theta + (\dot{x}\cos\delta + \dot{y}\sin\delta)\mathbf{e_r}, \tag{19}$$

$$\mathbf{v}_{\text{lat}} = -\dot{x}\sin\delta - e\dot{\delta} + \dot{y}\cos\delta, \tag{20}$$
$$\mathbf{v}_{\text{long}} = \dot{x}\cos\delta + \dot{y}\sin\delta. \tag{21}$$

Final combined equation:

$$\frac{\sigma}{\dot{x}\cos\delta + \dot{y}\sin\delta}\dot{Y} + Y = -C\frac{-\dot{x}\sin\delta - e\dot{\delta} + \dot{y}\cos\delta}{\dot{x}\cos\delta + \dot{y}\sin\delta}. \tag{22}$$

The following 4 equations represents the equations of motion of the system:

$$m(\ddot{x} + f\ddot{\delta}\sin\delta + f\dot{\delta}^2\cos\delta) = F_x - Y\sin\delta, \tag{23}$$
$$m(\ddot{y} - f\ddot{\delta}\cos\delta + f\dot{\delta}^2\sin\delta) = Y\cos\delta - ky, \tag{24}$$
$$I_{zz}\ddot{\delta} = -fky\cos\delta - F_x f\sin\delta - (e-f)Y, \tag{25}$$
$$\frac{\sigma}{\dot{x}\cos\delta + \dot{y}\sin\delta}\dot{Y} + Y = -C\frac{-\dot{x}\sin\delta - e\dot{\delta} + \dot{y}\cos\delta}{\dot{x}\cos\delta + \dot{y}\sin\delta}. \tag{26}$$

2 F_x

The forward speed $\dot{x}$ is constraint in order to remain a constant value $\dot{x} = v \in \mathbb{R}$ by applying a specific force F_x. To find F_x you have to set $\ddot{x} = 0$ due to the fact that $\dot{x}$ is constant.

$$\ddot{x} = 0 \rightarrow m(0 + f\ddot{\delta}\sin\delta + f\dot{\delta}^2\cos\delta) = F_x - Y\sin\delta \tag{27}$$
$$F_x = m(f\ddot{\delta}\sin\delta + f\dot{\delta}^2\cos\delta) + Y\sin\delta \tag{28}$$

3 Equations of motion with non-holonomic constraint

In the case of a non-holonomic constraint in tire modeling the equation of $\mathbf{v}_{\text{lat}}$ becomes zero:

$$\mathbf{v}_{\text{lat}} = -\dot{x}\sin\delta - e\dot{\delta} + \dot{y}\cos\delta = 0, \tag{29}$$

, and with the forward speed constrained to be constant, the four equations of motion become:

$$\frac{\sigma}{v\cos\delta + \dot{y}\sin\delta}\dot{Y} + Y = 0, \tag{30}$$
$$F_x = m(f\ddot{\delta}\sin\delta + f\dot{\delta}^2\cos\delta) + Y\sin\delta, \tag{31}$$
$$m(\ddot{y} - f\ddot{\delta}\cos\delta + f\dot{\delta}^2\sin\delta) = Y\cos\delta - ky, \tag{32}$$
$$I_{zz}\ddot{\delta} = -fky\cos\delta - F_x f\sin\delta - (e-f)Y, \tag{33}$$

4 Complete MATLAB template

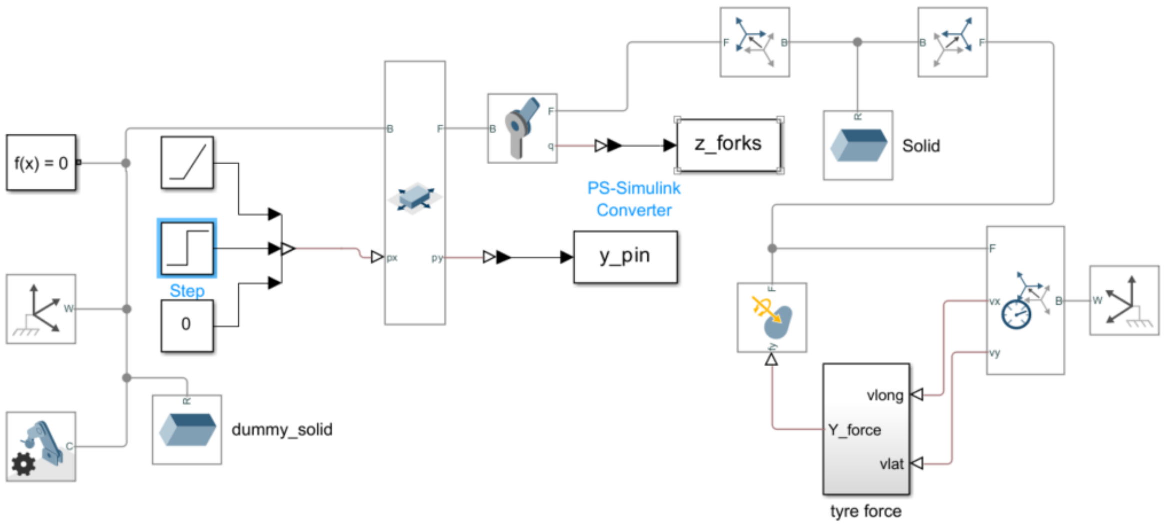
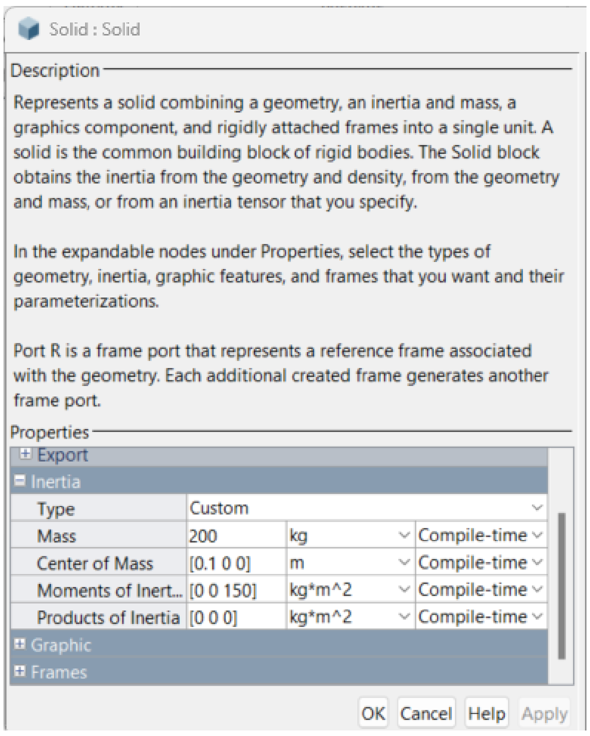


Figure 1: Complete assembly of the given template

5 Model examination



Solid : Solid

Description

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

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

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

Properties

⊞ Export

⊟ Inertia

Type	Custom		
Mass	200	kg	Compile-time
Center of Mass	[0.1 0 0]	m	Compile-time
Moments of Inert...	[0 0 150]	kg*m^2	Compile-time
Products of Inertia	[0 0 0]	kg*m^2	Compile-time

⊞ Graphic

⊞ Frames

Figure 2: Properties of the solid

As we can see from the properties,The total length of the body is 1.4 metre, the mass of the system is set to 200 kg and the inertia is set to 150 $kg * m^2$ in the z-direction.

Furthermore from the properties is is clear that the center of the mass is offset 0.1 metres in the +x-direction from the reference frame of the solid which is exactly at the center.
The following figure show the reference frame of the body:

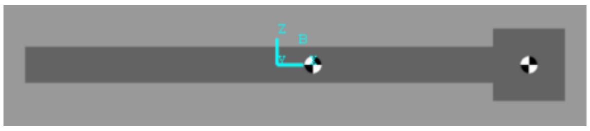


Figure 3: Reference frame of the body

And the following two figures show the rigid transformation between two frames, the two rigid transformations represent the connecting point between the fork and the king pin (Figure 4), and the point at which the Y force is acting at the body (Figure 6).

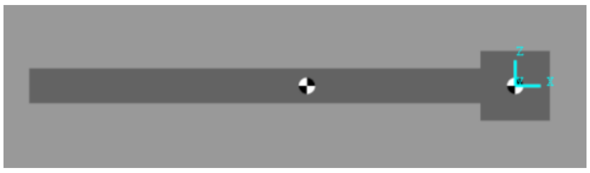


Figure 4: Frame offset +0.7 meters

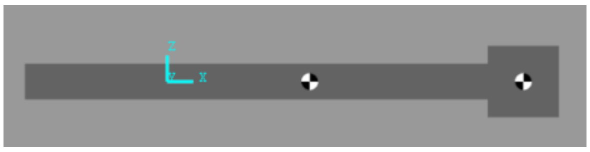


Figure 5: Frame offset -0.3 meters

From the figures we can extract the length of f and e which are 0.6 and 1 metre respectively. To find the value of C we had to extract the differential equation of the following subsystem.

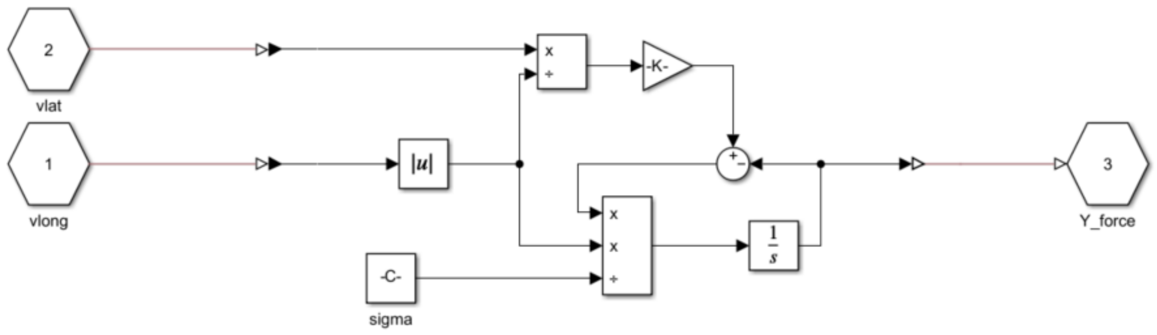


Figure 6: Tyre force subsystem

We compare the following equation (subsystem equation) with the tyre equation in the question.

$$\frac{\sigma}{|\mathbf{v}_{\text{long}}|}\dot{Y} + Y = k\frac{\mathbf{v}_{\text{lat}}}{|\mathbf{v}_{\text{long}}|} => \frac{\sigma}{|\mathbf{v}_{\text{long}}|}\dot{Y} + Y = -C\alpha.$$

$$k = -C = -40000 => C = 40000$$

Answers summary:

e	1 m
f	0.6 m
m	200 kg
I_z	150 $kg * m^2$
C	40000

Table 1: Model properties.

6 Varying stiffness value

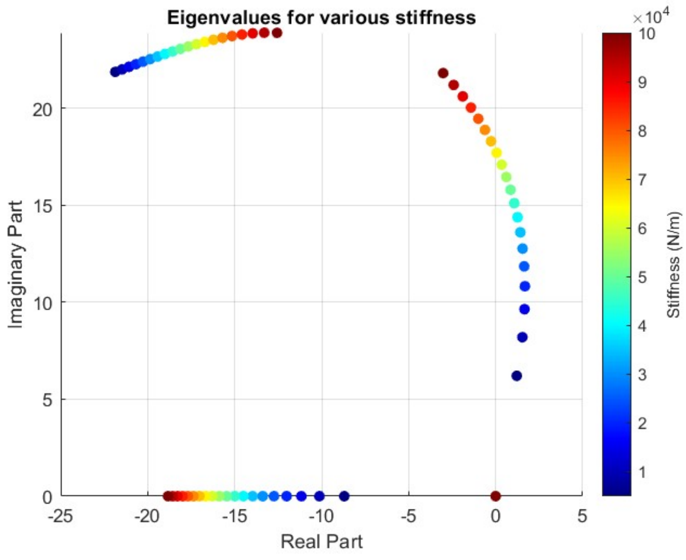


Figure 7: Eigenvalues for various stiffness values

According to the graph, for stiffness values below 65,000, the eigenvalues of the system are on the real positive axis. Eigenvalues on the real positive axis are considered unstable, and as a result, the system also becomes unstable. However, as the stiffness value increases, the eigenvalues move closer to the imaginary axis. When the stiffness value reaches 65,000, the system becomes marginally stable.

7 Validation of stability for k=30000 and k=100000

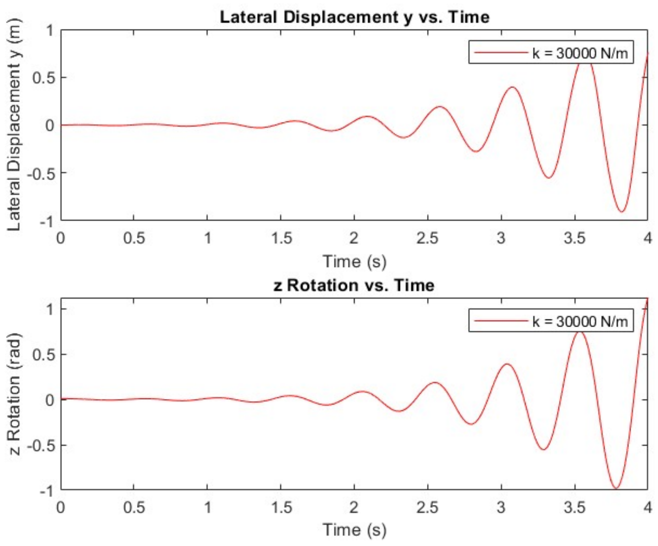


Figure 8: Lateral displacement and rotation for k=30000

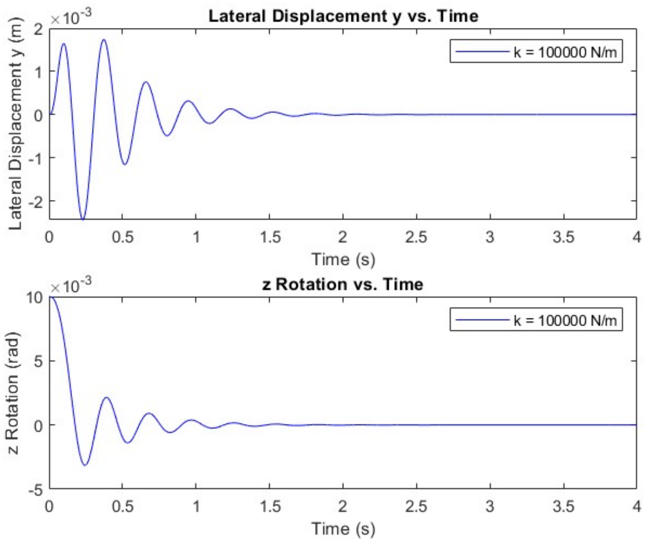


Figure 9: Lateral displacement and rotation for k=100000

The stability of the system for $k = 30000$ and $k = 100000$ is consistent with the results from the eigenvalues plot. For $k = 30000$, there are eigenvalues on the real positive axis, making the system unstable. In contrast, for $k = 100000$, all the eigenvalues are on the negative real axis, causing the system to stabilize.

Frequency response:

To determine the frequency response of the system, we select eigenvalues that indicate the system's transition between stable and unstable behavior. These are typically the eigenvalues with real parts closest to the imaginary axis. The imaginary part of an eigenvalue is directly related to the system's oscillatory response frequency. Specifically, the frequency f in hertz (cycles per second) can be calculated from the imaginary part $\mathrm{Im}(\lambda)$ of the eigenvalue λ using the formula:

$$f = \frac{\mathrm{Im}(\lambda)}{2\pi} \Rightarrow T = \frac{1}{f}$$

The figure below shows the eigenvalues selected for calculating the frequency response of the system for stiffness values of 100,000 and 30,000. The stiffness value is represented by the color in the graph:

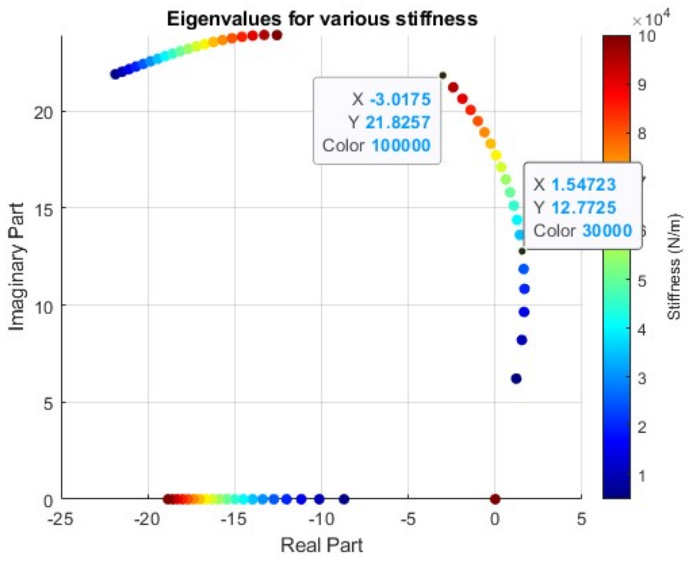


Figure 10: Chosen eigenvalues for validating the frequency response of the system with the previous results

For $k = 30000$:
From the response graph, the period of the response (T_r) is 0.49 sec. The period of the eigenvalues is given by:

$$\lambda = \sigma - j\omega \Rightarrow \omega = \frac{2\pi}{T} \Rightarrow T = \frac{2\pi}{\omega}$$

$$\omega_e = 12.77 \Rightarrow T_e = 0.49 = T_r$$

For $k = 100000$:
From the response graph, the period of the response (T_r) is 0.29 sec. The period of the eigenvalues is given by:

$$\lambda = \sigma - j\omega \Rightarrow \omega = \frac{2\pi}{T} \Rightarrow T = \frac{2\pi}{\omega}$$

$$\omega_e = 21.82 \Rightarrow T_e = 0.29 = T_r$$

8 Varying forward speed

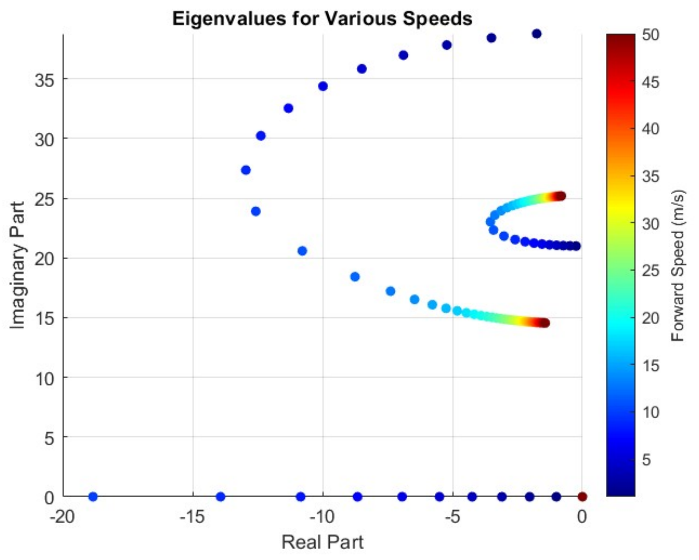


Figure 11: Eigenvalues for various stiffness values

According to the graph, all the eigenvalues have their real parts in the negative region. This implies that the system remains stable for any forward speed. There appears to be no correlation between the forward speed of the system and the occurrence of shimmy oscillations. Additionally, the system was tested for values significantly higher than those mentioned in the question, and it maintained stability across all these tests.

9 Varying relaxation length

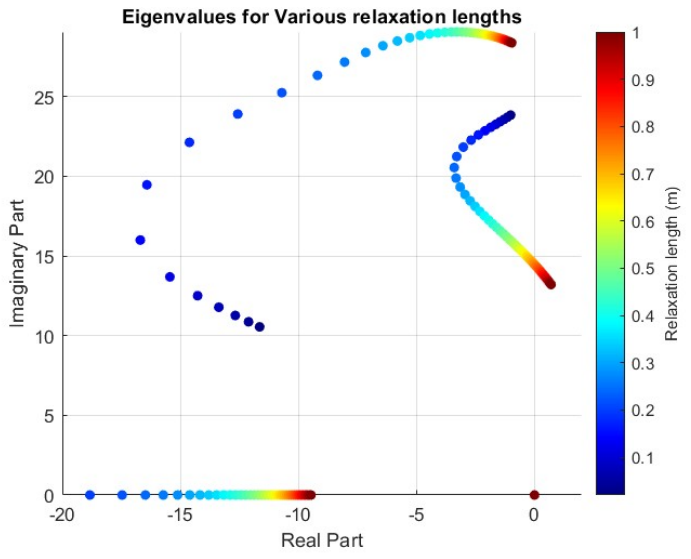


Figure 12: Eigenvalues for various relaxation length values

The graph shows that if the relaxation length is more than 0.74 meters, the system gets unstable. We see this in the lateral displacement and rotation plot, where the system is unstable when the relaxation length is 1 meter.

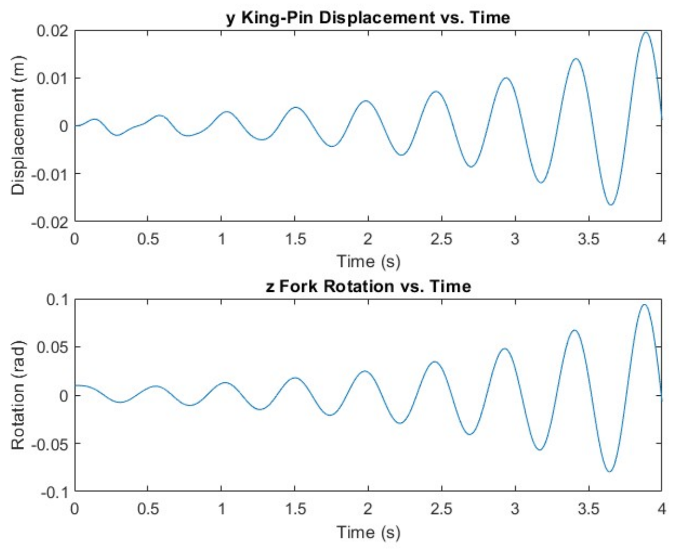


Figure 13: Lateral displacement and rotation for relaxation length = 1m

10 Feedback control $T = -\lambda\dot{\delta}$

Eigenvalues for k=60000:

$$
\begin{aligned}
\lambda_1 &= -17.2009 + 23.3165i, \\
\lambda_2 &= -17.2009 - 23.3165i, \\
\lambda_3 &= 0.3462 + 17.0988i, \\
\lambda_4 &= 0.3462 - 17.0988i, \\
\lambda_5 &= -16.2880 + 0.0000i, \\
\lambda_6 &= 0.0000 + 0.0000i, \\
\lambda_7 &= 0.0000 + 0.0000i, \\
\lambda_8 &= 0.0000 + 0.0000i, \\
\lambda_9 &= 0.0000 + 0.0000i.
\end{aligned}
$$

The system is unstable because two of its eigenvalues have a positive real part. Due to that, there are two unstable poles. The following plots validate that the system is unstable for k=60000.

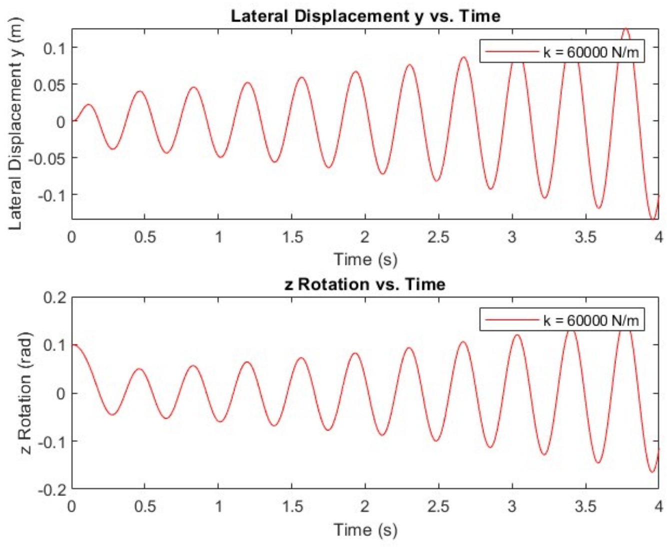


Figure 14: Lateral displacement and rotation for k=60000.

In order to stabilize the system we have to apply a feedback control law ($T = -\lambda\dot{\delta}$)as a torque applied on the wheel assembly. The feedback control is shown at the figure below.

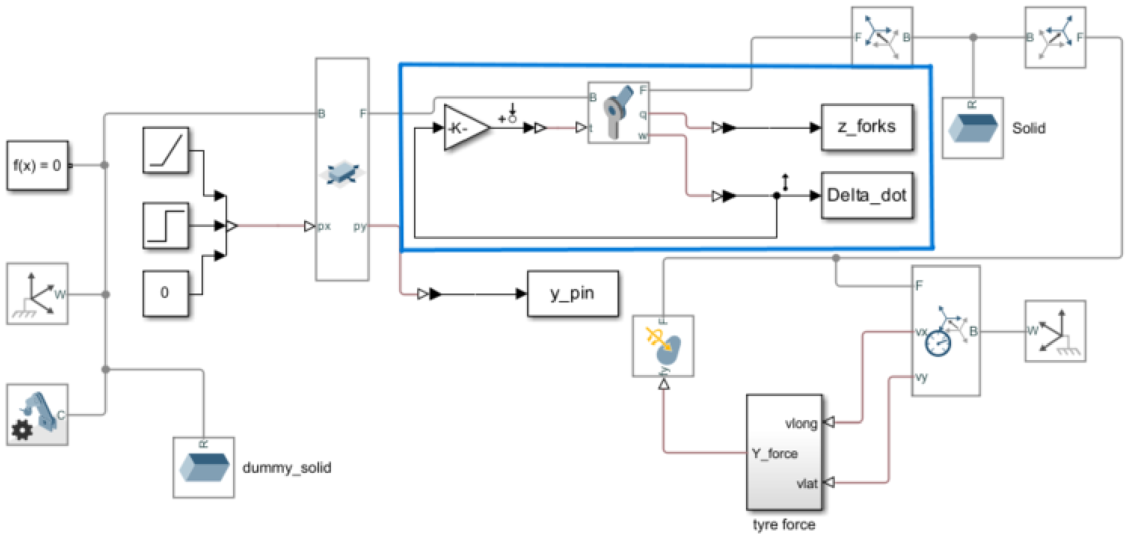


Figure 15: Modified model

To stabilize the system, it's essential to identify the appropriate feedback gain that will ensure stability. This feedback gain, denoted as λ, can be determined by plotting the Nyquist diagram for the open-loop system, with respect on the input T and the output $\dot{\delta}$.

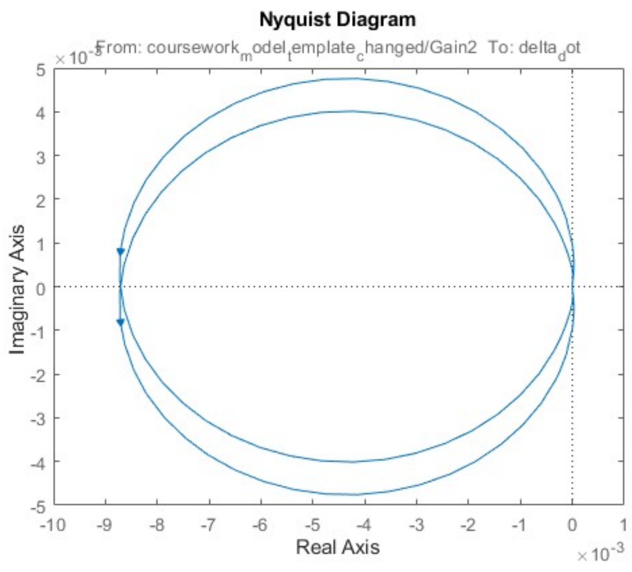


Figure 16: Nyquist diagram for the open-loop system with respect to the input T and output $\dot{\delta}$.

Based on the Nyquist stability criterion, we can assess system stability using the Nyquist diagram. This criterion specifies that for a system with two poles in the right half of the imaginary plane to be stable, it must encircle the critical point, which is -1 + 0j, two times in a counterclockwise direction. However, the Nyquist diagram of our open-loop system does not encircle the -1 point, despite having unstable poles, and due to that our open-loop system is unstable.

For the system to achieve marginal stability, it must intersect at the critical point, -1 + j0. The Nyquist diagram of the open-loop system shows the point where it crosses the real axis, as illustrated in the figure below. To ensure stability we need to shift the point shown below to cross from the point -1.

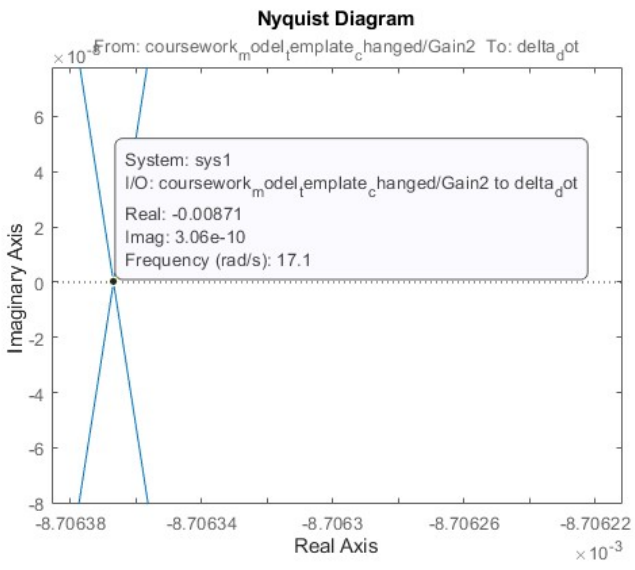


Figure 17: Point of Nyquist for which the Imaginary part becomes zero.

To shift this point to intersect at -1 + 0j, we need to set $\lambda = \frac{1}{a}$, where 'a' represents the point on the real axis. We calculate $\lambda = \frac{1}{0.00871} = 114.81$. To determine this point more accurately, we can obtain the GM (Gain Margin) of the open-loop system, which is also GM $= \frac{1}{a}$, using the MATLAB code `[GM, PM, ~, ~] = margin(sys1)`, which yields $\lambda = 114.7755$.

To verify the marginal stability of the system, we can compute the zeros and poles of the system and compare them with those of the open-loop system.

Open loop poles:

$$\begin{aligned}
P_1 &= 0.3462 + 17.0988i\\
P_2 &= 0.3462 - 17.0988i\\
P_3 &= -17.2014 + 23.3163i\\
P_4 &= -17.2014 - 23.3163i\\
P_5 &= -16.2896 + 0.0000i
\end{aligned}$$

Close-loop poles:

$$\begin{aligned}
P_1 &= -17.2945 + 23.3132i\\
P_2 &= -17.2945 - 23.3132i
\end{aligned}$$

$$P_3 = -0.0000 + 17.1309i$$
$$P_4 = -0.0000 - 17.1309i$$
$$P_5 = -16.1761 + 0.0000i$$

We can see that the poles of the system have been shifted in order to lie on the imaginary axis. From the graph bellow we can see that the lateral displacement and the rotational displacement will be oscillating forever maintaining the so called marginal stability.

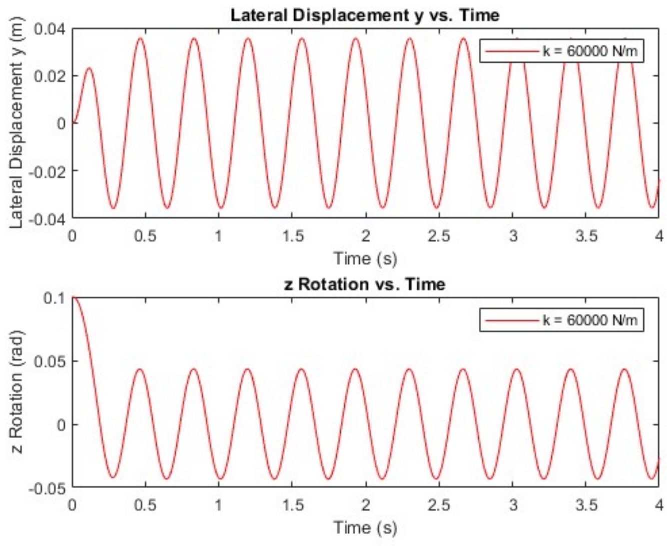


Figure 18: Asymptotic stability.

To verify that the system will converge quickly to stability it was necessary to increase λ and run another simulation.

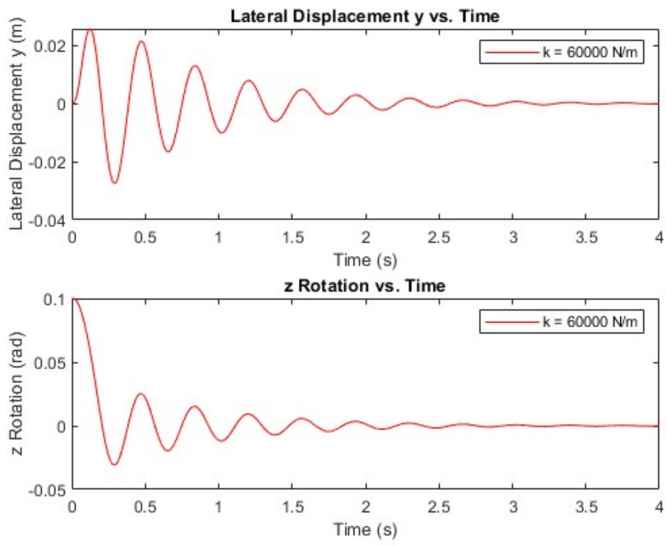


Figure 19: Simulation with λ five times higher than before.

A Appendix

A.1 MATLAB Script

```
% Exercise 4.6
    mdl = 'coursework_model_template';
% Initialize parameters:
    sigma_rel = 0.2;
    k_values = 5000:5000:100000;% values for loop (counter)
    z_initial=0; %e linearisation should always be performed
       at z_in=0;
    eigenvaluesEx6 = [];
    k_for_eigenvalues = [];
    speed=10; %constant speed

% Range the stiffness value k:
    for k = k_values
        [A, B, C, D] = linmod(mdl);% Linearize the model
        eigs_current = eig(A);% Compute eigenvalues of A
        eigenvaluesEx6 = [eigenvaluesEx6; eigs_current];%
           Store the eigenvalues in diff column
        % of the matrix eigenvaluesEX6 for every interation.
        % Store k for each eigenvalue
        k_for_eigenvalues = [k_for_eigenvalues; k *
           ones(length(eigs_current), 1)];
    end

% Plot eigenvalues on complex plane
    figure;
    scatter(real(eigenvaluesEx6), imag(eigenvaluesEx6), 36,
       k_for_eigenvalues, 'filled');
    cb = colorbar; % colorbar to identify the value of k for
       every eig.
    colormap('hot');
    ylabel(cb, 'Stiffness␣(N/m)');
    ylim([0 inf]);
    xlabel('Real␣Part');
    ylabel('Imaginary␣Part');
    title('Eigenvalues␣for␣various␣stiffness');
    grid on;
    clim([min(k_values) max(k_values)]); % Color axis range
       for k values
    colormap jet;

%%
%Exercise 4.7
% Initialize parameters
    sim_time = 4; % 4 seconds of simulation
    set_param(mdl, 'StopTime', num2str(sim_time));
    z_initial = 0.01;

% Simulation k = 30000
    k=30000;
    [A, B, C, D] = linmod(mdl);
    sim_out1 = sim(mdl);
% Extract output data for the 1st simulation
    y_pin1_data = y_pin.signals.values;
    time1 = y_pin.time;
    z_forks1_data = z_forks.signals.values;

% Simulation with k = 100000
    k=100000;
    [A, B, C, D] = linmod(mdl);
    sim_out2 = sim(mdl); % Run

% Extract output data for the 2nd simulation
    y_pin2_data = y_pin.signals.values;
    time2 = y_pin.time;
    z_forks2_data = z_forks.signals.values;

% Lateral displacement k = 30000
    figure;
    subplot(2, 1, 1);
    plot(time1, y_pin1_data, 'r-');
    xlabel('Time␣(s)');
    ylabel('Lateral␣Displacement␣y␣(m)');
    legend('k␣=␣30000␣N/m');
    title('Lateral␣Displacement␣y␣vs.␣Time');

% Rotation k = 30000
    subplot(2, 1, 2);
    plot(time1, z_forks1_data, 'r-');
    xlabel('Time␣(s)');
    ylabel('z␣Rotation␣(rad)');
    legend('k␣=␣30000␣N/m');
    title('z␣Rotation␣vs.␣Time');

% Lateral displacement k=100000
    figure;
    subplot(2, 1, 1);
    plot(time2, y_pin2_data, 'b-');
    xlabel('Time␣(s)');
    ylabel('Lateral␣Displacement␣y␣(m)');
    legend('k␣=␣100000␣N/m');
    title('Lateral␣Displacement␣y␣vs.␣Time');
```

```
% Rotation k=100000
    subplot(2, 1, 2);
    plot(time2,z_forks2_data, 'b-');
    xlabel('Time␣(s)');
    ylabel('z␣Rotation␣(rad)');
    legend('k␣=␣100000␣N/m');
    title('z␣Rotation␣vs.␣Time');
%%
% Exercise 4.8

% Initialize parameters
    k = 100000;
    z_initial=0;
    speed_values = 1:1:50; %Range speed (loop counter)
    eigenvalues_all = [];
    speeds_for_eigenvalues = []; % To keep track of which
       speed corresponds to which eigenvalue

% Loop for range the speed values
    for speed = speed_values
        [A, B, C, D] = linmod(mdl);
        eigs_current = eig(A);
        eigenvalues_all = [eigenvalues_all; eigs_current];
        speeds_for_eigenvalues = [speeds_for_eigenvalues;
           repmat(speed, length(eigs_current), 1)];
    end

% Plot eigenvalues
    figure;
    scatter(real(eigenvalues_all), imag(eigenvalues_all), 30,
       speeds_for_eigenvalues, 'filled');
    colorbar;
    cb = colorbar; % Create a handle for the colorbar
    ylabel(cb, 'Forward␣Speed␣(m/s)'); % Add label to the
       colorbar
    xlabel('Real␣Part');
    ylabel('Imaginary␣Part');
    title('Eigenvalues␣for␣Various␣Speeds');
    xlim([-20 inf]); % Limit for the real part
    ylim([0 inf]);   % Limit for the imaginary part
    grid on;
    clim([min(speed_values) max(speed_values)]); % Color axis
       range for speed values
    colormap jet; % Set the colormap

 %% Exercise 4.9
    speed = 10;
    k = 100000;
    sigma_values = 0.02:0.02:1;
    eigenvalues_all = [];
    sigma_for_eigenvalues = [];

    % Loop to range sigma values
    for sigma_rel = sigma_values
        [A, B, C, D] = linmod(mdl);
        eigs_current = eig(A);
        eigenvalues_all = [eigenvalues_all; eigs_current];
        sigma_for_eigenvalues = [sigma_for_eigenvalues;
           sigma_rel * ones(length(eigs_current), 1)];
    end

% Plot eigenvalues on the complex plane
    figure;
    scatter(real(eigenvalues_all), imag(eigenvalues_all), 30,
       sigma_for_eigenvalues, 'filled');
    xlabel('Real␣Part');
    ylabel('Imaginary␣Part');
    title('Eigenvalues␣for␣Various␣relaxation␣lengths');
    xlim([-20, 2]);
    ylim([0 inf])
    grid on;
    cb = colorbar;
    ylabel(cb, 'Relaxation␣length␣(m)');
    clim([min(sigma_values) max(sigma_values)]);
    colormap jet;

% Initialize parameters
    sigma_rel = 1;
    z_initial = 0.01;
    [A, B, C, D] = linmod(mdl);
    sim(mdl); %run

% Extract output
    y_pin3_data = y_pin.signals.values;
    time3 = y_pin.time;
    z_forks3_data = z_forks.signals.values;

% Plot Lat dis and rotation
    figure;
    subplot(2, 1, 1);
    plot(time3, y_pin3_data);
    title('y␣King-Pin␣Displacement␣vs.␣Time');
    xlabel('Time␣(s)');
    ylabel('Displacement␣(m)');

    subplot(2, 1, 2);
    plot(time3, z_forks3_data);
    title('z␣Fork␣Rotation␣vs.␣Time');
    xlabel('Time␣(s)');
    ylabel('Rotation␣(rad)');

%%
% 4.10
% Intialize parameters
    mdl = 'coursework_model_template';
    sigma_rel = 0.2;
    k=60000;
    speed=10;
    z_initial=0;
    [A, B, C, D] = linmod(mdl);
    eigs_current = eig(A) %print eigvalues

 % simulate with delta=0
    z_initial = 0.1;
    [A, B, C, D] = linmod(mdl);
    sim_time = 4; % 4 seconds of simulation
    set_param(mdl, 'StopTime', num2str(sim_time));
    sim_out3 = sim(mdl);

% Extract output
    y_pin1_data = y_pin.signals.values;
    time1 = y_pin.time;
    z_forks1_data = z_forks.signals.values;

% Lateral displacement y
    figure;
    subplot(2, 1, 1);
    plot(time1, y_pin1_data, 'r-');
    xlabel('Time␣(s)');
    ylabel('Lateral␣Displacement␣y␣(m)');
    legend('k␣=␣60000␣N/m');
    title('Lateral␣Displacement␣y␣vs.␣Time');

% z rotation of forks for both simulations
    subplot(2, 1, 2);
    plot(time1, z_forks1_data, 'r-');
    xlabel('Time␣(s)');
    ylabel('z␣Rotation␣(rad)');
    legend('k␣=␣60000␣N/m');
    title('z␣Rotation␣vs.␣Time');
    hold off;

% open the new changed template which has the control feedback
   loop
```

```
    mdl = 'coursework_model_template_changed';
% Initialize parameters
    sigma_rel = 0.2;
    k=60000;
    speed=10;
    z_initial=0;
    lamda=0;

 % Specify the input and output points for linearization
    in = linio('coursework_model_template_changed/Gain2', 1,
       'input'); % path of input
    out = linio('coursework_model_template_changed/PS-Simulink␣
       Converter1', 1, 'output'); % path of out
    io = [in, out];
    sys1 = linearize(mdl, io);%linirization with respect to a
       specific in and out
    Open_loop_P=pole(sys1) %Validation of marginal stability
    Open_loop_Z=zero(sys1) %Validation of marginal stability
    [GM,PM,~,~] = margin(sys1);

    f1=figure;
    nyquist(sys1);
    xlim([-0.01 0.001]);%focus
    hold off;

    lamda=-1*GM %Gm for marginal stability and 5*GM quick
       stabilization
    in2 = linio('coursework_model_template_changed/Gain2', 1,
       'input');
    out2 =
       linio('coursework_model_template_changed/PS-Simulink␣
       Converter1', 1, 'output');
    io2 = [in2, out2];
    sys2= linearize(mdl, io2);
    Close_loop_P=pole(sys2)%print poles to validate stability
    Close_loop_Z=zero(sys2)%print zeroes to validate stability
    [GM,PM,~,~] = margin(sys2)
    f2=figure;
    nyquist(sys2);
    hold off;

% Plots
    z_initial = 0.1;
    sim_time = 4; % 4 seconds of simulation
    set_param(mdl, 'StopTime', num2str(sim_time));
    sim_out4 = sim(mdl);

% Extract output data
```

```
    y_pin4_data = y_pin.signals.values;
    time4 = y_pin.time;
    z_forks4_data = z_forks.signals.values;

% Lateral displacement
    figure;
    subplot(2, 1, 1);
    plot(time4, y_pin4_data, 'r-');
    xlabel('Time␣(s)');
    ylabel('Lateral␣Displacement␣y␣(m)');
    legend('k␣=␣60000␣N/m');
    title('Lateral␣Displacement␣y␣vs.␣Time');

% Rotation
    subplot(2, 1, 2);
    plot(time4, z_forks4_data, 'r-');
    xlabel('Time␣(s)');
    ylabel('z␣Rotation␣(rad)');
    legend('k␣=␣60000␣N/m');
    title('z␣Rotation␣vs.␣Time');
    hold off;
```