Assignment 4: Finding stability for wheelset equation

Khushi Agrawal (210514), Shashank Singh Tomar (210965)

The code implementation is provided at the end.

1. Routh Hurwitz

$$m\ddot{y} + \frac{2f_{22}}{V}\dot{y} + \left(\frac{2f_{23}}{V} - \frac{I_y \kappa V}{r_o l}\right)\dot{\psi} - 2f_{22}\psi + K_y y = Q_y$$

$$I_z\ddot{\psi} + \frac{2f_{11}l^2}{V}\dot{\psi} - \left(\frac{2f_{23}}{V} - \frac{I_y \delta_o V}{r_o l}\right)\dot{y} + \frac{2f_{11}\lambda_o l}{r_o}y + K_\psi \psi = Q_\psi$$

Using these two equations and setting them as:

$a_1 y'' + a_2 y' + a_3 \psi' - a_4 \psi + a_5 y = 0$

$b_1 \psi'' + b_2 \psi' - b_3 y' + b_4 y + b_5 \psi = 0$

We get,

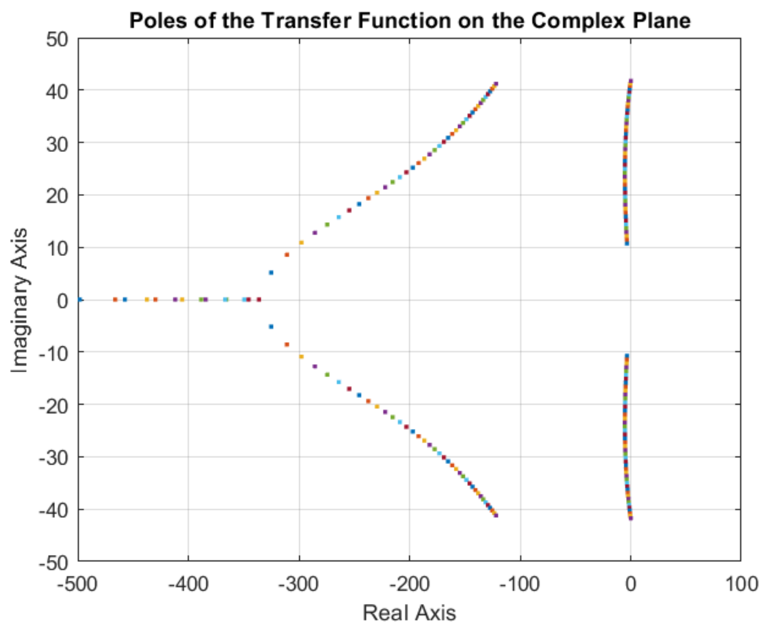
```
a1 = m;
a2 = 2*f22/V;
a3 = (2*f23/V - Iy*kappa*V/(r0*l));
a4 = 2*f22;
a5 = Ky;
b1 = Iz;
b2 = 2*f11*(l^2)/V;
b3 = (2*f23/V - Iy*delta0*V/(r0*l));
b4 = 2*f11*lambda0*l/r0 ;
b5 = K_sai;

p4 = a1*b1;
p3 = (a1*b2 + a2*b1);
p2 = (a1*b5 + a2*b2 + a5*b1 + a3*b3);
p1 = (a2*b5 + a5*b2 - a3*b4 - a4*b3);
p0 = (a5*b5 + a4*b4);
```

Where $p_4 s^4 + p_3 s^3 + p_2 s^2 + p_1 s^1 + p_0 s^0 = 0$

Then we apply the Routh Hurwitz criterion to this characteristic equation to get V_{cr} **= 79.09 m/s**

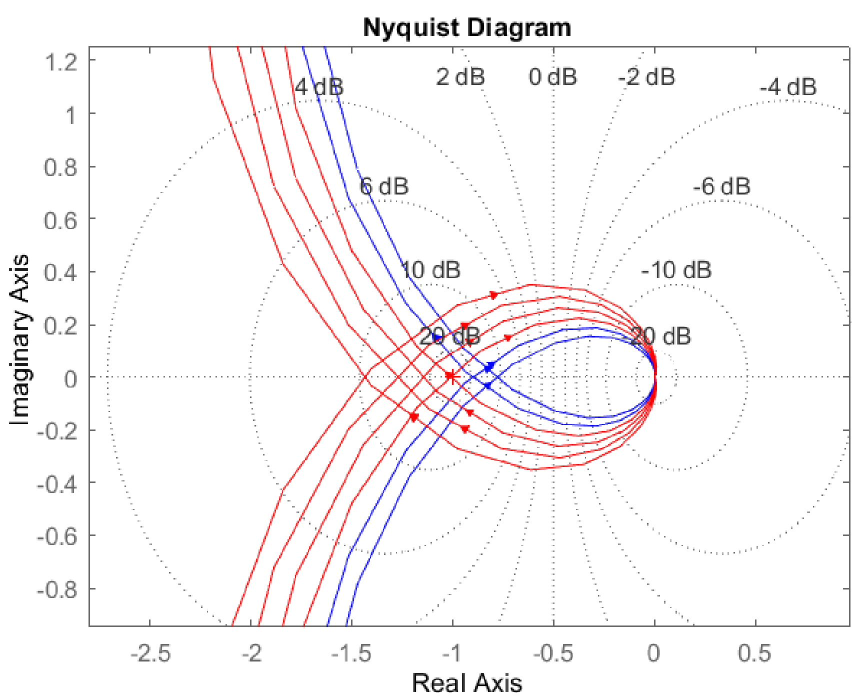
2. Root Locus

We vary the velocity from 20 to 150 m/s and calculate the roots of the characteristic equation. Plotting these roots gives us the Root Locus. The Roots in the LHP denote the unstable velocities. Here, also we get V_{cr} **= 79.09 m/s.**



2. Nyquist Criterion

We plot the Nyquist plot of the open loop transfer function for various different velocities. The velocities which encircle the -1 point denote the unstable velocities.



Codes

1. Routh_Hurwitz.m

```
%Stability determined by Routh Hurwitz criterion:
%values
m=1486;
Iz=1034;
Iy=166;
W=25*1000;
ky=1.561*10^6;
k_sai = 2.12*10^6;
cy=0;
c_sai = 0;
r0=0.5;
l=0.835;
lambda0=0.1174;
eps0=6.423;
delta0=0.02754;
sigma=0.0508;
f11=7.44*10^6;
f22=6.79*10^6;
f23=13.7*10^3;
g=9.8;
N0= W*g/4;
kappa = delta0*(1-f23/(N0*r0));
Ky = ky + (2*N0*eps0/l)*(1-f23/(N0*r0));
K_sai = k_sai + (2*N0*l)*(-delta0+f23/(N0*l));
% Define a range of V values to analyze
V_values = linspace(20, 150, 100);
% Loop over all V values and compute the maximum real part of eigenvalues
for V = V_values
   a1 = m;
   a2 = 2*f22/V;
   a3 = (2*f23/V - Iy*kappa*V/(r0*l));
   a4 = 2*f22;
   a5 = Ky;
   b1 = Iz;
   b2 = 2*f11*(l^2)/V;
   b3 = (2*f23/V - Iy*delta0*V/(r0*l));
   b4 = 2*f11*lambda0*l/r0;
   b5 = K_sai;

   p4 = a1*b1;
   p3 = (a1*b2 + a2*b1);
   p2 = (a1*b5 + a2*b2 + a5*b1 + a3*b3);
   p1 = (a2*b5 + a5*b2 - a3*b4 - a4*b3);
```

```
    p0 = (a5*b5 + a4*b4);
    coeffVector = [p4,p3,p2,p1,p0];
    stability = routh_hurwitz(coeffVector);
    if stability == 0
        disp(V)
        break;
    end
end
function [stability] = routh_hurwitz(coeffVector)
    ceoffLength = length(coeffVector);
    rhTableColumn = round(ceoffLength/2);

    rhTable = zeros(ceoffLength,rhTableColumn);

    rhTable(1,:) = coeffVector(1,1:2:ceoffLength);

    if (rem(ceoffLength,2) ~= 0)
        rhTable(2,1:rhTableColumn - 1) = coeffVector(1,2:2:ceoffLength);
    else
        rhTable(2,:) = coeffVector(1,2:2:ceoffLength);
    end
    epss = 0.01;
    %  Calculate other elements of the table
    for i = 3:ceoffLength

        %  special case: row of all zeros
        if rhTable(i-1,:) == 0
            order = (ceoffLength - i);
            cnt1 = 0;
            cnt2 = 1;
            for j = 1:rhTableColumn - 1
                rhTable(i-1,j) = (order - cnt1) * rhTable(i-2,cnt2);
                cnt2 = cnt2 + 1;
                cnt1 = cnt1 + 2;
            end
        end

        for j = 1:rhTableColumn - 1
            %  first element of upper row
            firstElemUpperRow = rhTable(i-1,1);

            %  compute each element of the table
            rhTable(i,j) = ((rhTable(i-1,1) * rhTable(i-2,j+1)) - ....
                (rhTable(i-2,1) * rhTable(i-1,j+1))) / firstElemUpperRow;
        end


        %  special case: zero in the first column
        if rhTable(i,1) == 0
```

```
            rhTable(i,1) = epss;
        end
    end
    unstablePoles = 0;
    %   Check change in signs
    for i = 1:ceoffLength - 1
        if sign(rhTable(i,1)) * sign(rhTable(i+1,1)) == -1
            unstablePoles = unstablePoles + 1;
        end
    end

    if unstablePoles == 0
        stability = 1;
    else
        stability = 0;
    end
end
```

2. Root_Locus.m

```
% Given parameters
m = 1486;                   % mass (kg)
g = 9.81;                   % gravitational acceleration (m/s^2)
Iy = 166;                   % moment of inertia about the y-axis (kg.m^2)
Iz = 1034;                  % moment of inertia about the z-axis (kg.m^2)
W = 25000;                  % Mass
ky = 1.561 * 10^6;          % lateral stiffness (N/m)
k_sai = 2.12 * 10^6;        % yaw stiffness (Nm/rad)
f11 = 7.44 * 10^6;          % lateral damping coefficient (N)
f22 = 6.79 * 10^6;          % yaw damping coefficient (N)
f23 = 13.7 * 10^3;          % coupling damping coefficient (N)
r0 = 0.5;                   % reference radius (m)
l = 0.838;                  % wheelbase (m)
lambda0 = 0.1174;           % model parameter (unitless)
delta0 = 0.02754;           % model parameter (unitless)
eps0 = 6.423;               % model parameter (unitless)
% N0, kappa, Ky, and K_sai calculations
N0 = W * g /4; % Normal force
kappa = delta0 * (1 - f23 / (N0 * r0));
Ky = ky + (2 * N0 * eps0 / l) * (1 - f23 / (N0 * r0));
K_sai = k_sai + (2 * N0 * l) * (-delta0 + f23 / (N0 * l));
% Set velocity range to analyze
V_range = linspace(20, 150, 100);  % Velocity range (1 to 150 m/s)
critical_velocity = 0;             % To store the critical velocity
unstable_found = false;            % Flag to stop when critical velocity is
found
pole_array = [];
% Loop over the velocity range and calculate the transfer function
```

```
figure;
for V = V_range

   %calculations
   a1 = m;
   a2 = 2*f22/V;
   a3 = (2*f23/V - Iy*kappa*V/(r0*l));
   a4 = 2*f22;
   a5 = Ky;
   b1 = Iz;
   b2 = 2*f11*(l^2)/V;
   b3 = (2*f23/V - Iy*delta0*V/(r0*l));
   b4 = 2*f11*lambda0*l/r0 ;
   b5 = K_sai;

   p4 = a1*b1;
   p3 = (a1*b2 + a2*b1);
   p2 = (a1*b5 + a2*b2 + a5*b1 + a3*b3);
   p1 = (a2*b5 + a5*b2 - a3*b4 - a4*b3);
   p0 = (a5*b5 + a4*b4);
   din = [p4,p3,p2,p1,p0];
   transfer_function = tf(1,din);
   % Get the poles of the transfer function
   poles = pole(transfer_function);

   pole_array = [pole_array poles];

   % Check if any pole has a positive real part (instability condition)
   if any(real(poles) > 0)
       critical_velocity = V;  % Store the velocity at which instability occurs
       unstable_found = true;
       break;  % Exit the loop once instability is found
   end
end
% Plot the poles on the complex plane (real and imaginary axes)
plot(real(pole_array), imag(pole_array),'.');  %for poles
hold on
xlabel('Real Axis');
   ylabel('Imaginary Axis');
   title('Poles of the Transfer Function on the Complex Plane');
   grid on;
% Display the critical velocity
if unstable_found
   fprintf('The system becomes unstable at a critical velocity of: %.2f m/s\n',
critical_velocity);
else
   fprintf('The system remains stable within the analyzed velocity range.\n');
end
```

2. Nyquist_Criterion.m

```
%values
m=1486;
Iz=1034;
Iy=166;
W=25*1000;
ky=1.561*10^6;
k_sai = 2.12*10^6;
cy=0;
c_sai = 0;
r0=0.5;
l=0.835;
lambda0=0.1174;
eps0=6.423;
delta0=0.02754;
sigma=0.0508;
f11=7.44*10^6;
f22=6.79*10^6;
f23=13.7*10^3;
g=9.8;
N0= W*g/4;
kappa = delta0*(1-f23/(N0*r0));
Ky = ky + (2*N0*eps0/l)*(1-f23/(N0*r0));
K_sai = k_sai + (2*N0*l)*(-delta0+f23/(N0*l));
V_values = [70, 75, 80, 85, 90, 95];
for V = V_values
   term1_num = [(-2*f23/V + Iy*kappa*V/(r0*l)), 2*f22];
   term1_denum = [m, 2*f22/V, Ky];
   term2_num = [-2*f23/V + Iy*delta0*V/(r0*l) ,2*f11*lambda0*l/r0];
   term2_denum = [Iz, 2*f11*(l^2)/V, K_sai];

   term1 = tf(term1_num, term1_denum);
   term2 = tf(term2_num, term2_denum);
   transfer_function = term1*term2;
   poles = pole(transfer_function);
   poles_RHP = poles(real(poles)>0);
   n_poles_RHP = length(poles_RHP);

   if V>79
       color = 'r';
   else
       color = 'b';
   end
   nyquist(transfer_function, color);
   grid on;
   hold on
end
```