MMF062 Vehicle Dynamics
Design Task 2 - Simulation of Lateral Dynamics using an electrified SAAB 9-3

Gabriel Wendel
Erik Lydig

Figure 1: SAAB 9-3 (2014). [1]

Second group assignment in the course MMF062 - Vehicle Motion Engineering

MSc Mobility Engineering
Chalmers University of Technology
Sweden

Task 1: Steady State Cornering Characteristics

In this task we are considering three different load cases, for which we calculate various parameters.

$$\begin{bmatrix} \text{Load case 1: } l_f = 0.37L \\ \text{Load case 2: } l_f = 0.63L \\ \text{Load case 3: } l_f = 0.47L \end{bmatrix}$$

Task 1.1: Steady state cornering equations

To derive the Steady-State cornering equations for a one-track vehicle model at high speeds, we start by assuming the following:

1. Small steering (slip) angle:
   - $\cos(\alpha) \approx 1$
   - $\sin(\alpha) \approx \alpha$
   - $\tan(\alpha) \approx \alpha$
2. Path radius >> vehicle. Forces and accelerations are approximately co-directed.
3. Slip in longitudinal direction $s_x = 0$.

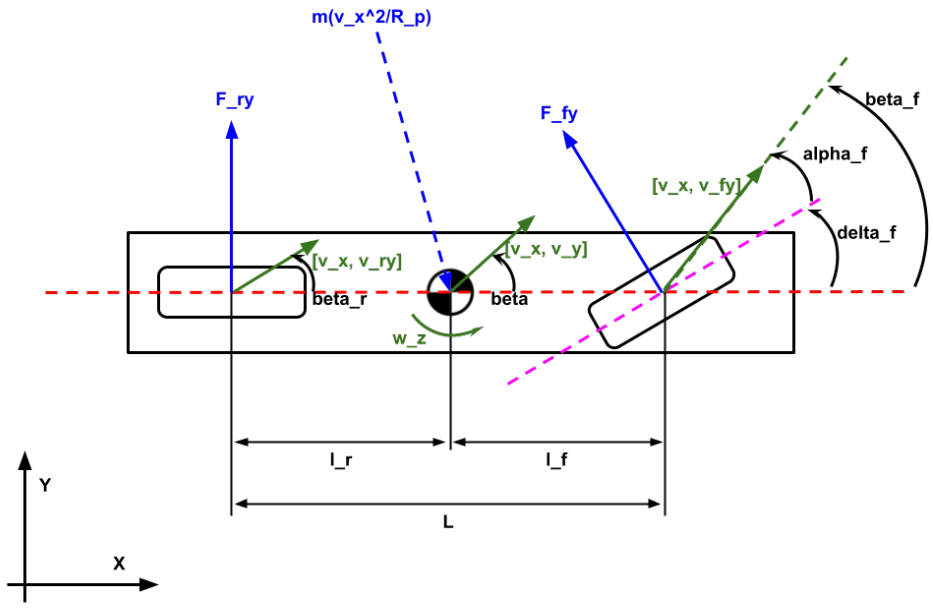


Figure 2: One-track (bicycle model)

From Figure 2, equilibrium in y-direction, together with the assumption of small angles yields:

$$m\frac{v_x^2}{R_p}\cos(\delta_f) = F_{ry} + F_{fy}\cos(\delta_f) \Rightarrow m\frac{v_x^2}{R_p} \approx F_{ry} + F_{fy} \tag{1}$$

The same assumption is used for deriving the moment equilibrium around the CoG:

$$0 = F_{ry}l_r - F_{fy}l_f\cos(\delta_f) \Rightarrow 0 \approx F_{ry}l_r - F_{fy}l_f \tag{2}$$

Constitution relationship between lateral tyre force and lateral slip stiffness:

$$F_{fy} = -C_f s_{fy} \tag{3}$$
$$F_{ry} = -C_r s_{ry} \tag{4}$$

Compatibility between the steering angle δ_f, slip angle α_f and course (body slip) angle β_f:

General compatibility formula:

$$\beta_f = \delta_f + \alpha_f$$

General formula for lateral slip:

$$s_y = \frac{v_y}{R\omega} \approx [s_x = 0] \approx \frac{v_y}{v_x} = \tan(\alpha) \approx \alpha$$

From compatibility we can describe lateral slip in the rear wheel as:

$$s_{ry} = \alpha_r = \beta_r - \delta_r$$

Taking yaw motion into account and that rear steering angle is zero we get:

$$s_{ry} = \frac{v_y - l_r\omega_z}{v_x} \tag{5}$$

Same can be said for the front wheel lateral slip, however the steering angle is not zero in this case:

$$s_{fy} = \frac{v_y + l_f\omega_z}{v_x} - \delta_f \tag{6}$$

The last equation describes the angular velocity ω_z, and once again we assume small angles, in other words, the v_x is parallel with the length of the vehicle and $v_y \approx 0$:

$$\omega_z \approx \frac{v_x}{R_p} \tag{7}$$

Combining Equation 1 - 7 yields the steering angle:

$$\delta_f \approx \frac{L}{R_p} + K_u \frac{m v_x^2}{R_p} \tag{8}$$

Where K_u is the understeering gradient defined as:

$$K_u = \frac{C_r l_r - C_f l_f}{C_f C_r L}$$

Task 1.2: Understeer gradient

In order to determine the Understeering gradients for the three load cases, one needs to calculate longitudinal forces for front- and rear wheels, F_{fz} and F_{rz}, as well as cornering stiffness, C_f and C_r.

$$F_{fz} = mg\frac{(1 - \text{length ratio})}{2} \tag{9}$$
$$F_{rz} = mg\frac{\text{length ratio}}{2} \tag{10}$$

$$C_f = 2(c_0 F_{fz} + c_1 F_{fz}^2) \tag{11}$$

$$C_r = 2(c_0 F_{rz} + c_1 F_{rz}^2) \tag{12}$$

Understeering gradient:

$$K_u = \frac{C_r l_r - C_f l_f}{C_f C_r L} = \begin{bmatrix} 6.99e-7 \\ -6.99e-7 \\ 1.59e-7 \end{bmatrix} \frac{1}{\text{N}} / \frac{\text{rad}}{\text{N}} / \frac{\text{rad}}{\text{m/s}^2} \tag{13}$$

Task 1.3: Critical/Characteristic speed

Critical speed defines the speed at which the vehicle becomes unstable in relation to large yaw velocity, due to small steering angle inputs. Both case 1 and 3 are understeered therefore they theoretically never reach a critical speed.

$$v_{c,crit} = \sqrt{\frac{L}{-K_u m}} = \begin{bmatrix} NaN \\ 47.8 \\ NaN \end{bmatrix} \text{ m/s} \tag{14}$$

Vehicles experiencing understeer lack a distinct critical speed. As a result, an alternative definition is employed, namely characteristic speed. Characteristic speed is determined as the point at which doubling the steering input is necessary to maintain the same cornering radius when driving at lower speeds. Note that the oversteered vehicle does not have a characteristic speed.

$$v_{c,crit} = \sqrt{\frac{L}{K_u m}} = \begin{bmatrix} 47.8 \\ NaN \\ 100.1 \end{bmatrix} \text{ m/s} \tag{15}$$

Task 1.4: Steering wheel angle for one certain operating point

Steering angle required to reach 4m/s^2 of lateral acceleration at 100km/h. From Equation [4.21] in Compendium. We can note that it seems reasonable the the oversteered car required the smallest steering angle.

$$\delta_f = a_y \left(\frac{L + K_u m v_x^2}{v_x^2} \right) = \begin{bmatrix} 0.0186 \\ 0.0092 \\ 0.0149 \end{bmatrix} \text{ rad} = \begin{bmatrix} 1.063 \\ 0.526 \\ 0.856 \end{bmatrix} ^\circ \tag{16}$$

Task 1.5: Steering wheel angle for varying operating point

To plot the required steering angle for different velocities for a given radius ($R_p = 200m$) we used Equation 8 derived earlier. Note the steering angle is normalized ($\frac{\delta_f R}{L}$), see Figure 3.

$$\delta_f = \frac{L}{R_p} + K_u \frac{m v_x^2}{R_p} \tag{17}$$

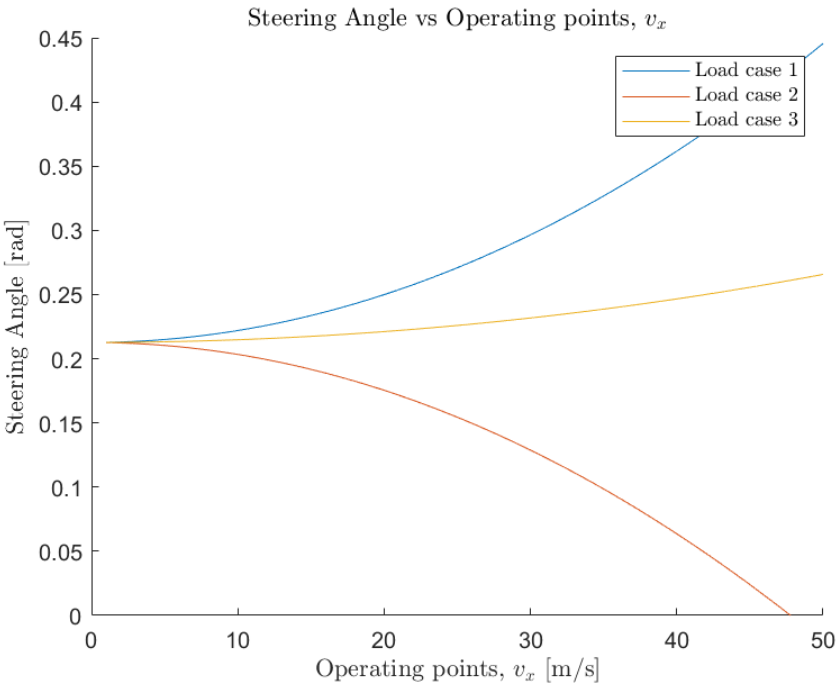


Figure 3: Steering angle vs Velocity for the three Load cases.

Task 2: Simulation with single track model

Task 2.1: Implement model

Equation 18 below describes the lateral slip angle, α, for rear and front axle. No small-angles assumption. Equations below were implemented in the *Lateral slip* - block.

$$\begin{aligned} \alpha_f &= \arctan\left(\frac{v_y + l_f\omega_z}{v_x}\right) - \delta_f \\ \alpha_r (= \beta_r) &= \arctan\left(\frac{v_y - l_r\omega_z}{v_x}\right) \end{aligned} \tag{18}$$

Equation 19 describes cornering stiffness, C, for front and rear. Equations below were implemented in *LatForce* - block within *TireModel*

$$\begin{aligned} C_f &= c_0 F_{zf} + c_1 F_f^2 \\ C_r &= c_0 F_{zr} + c_1 F_r^2 \end{aligned} \tag{19}$$

Equation 20 describes the motion of the vehicle, and are described in equation [4.52] in the Compendium. Equations below were implemented in *Chassis* - block.

$$\begin{aligned} \dot{v}_x &= \frac{F_{fxw}\cos(\delta_f) - F_{fyw}\sin(\delta_f) + F_{rx}}{m} + \omega_z v_y \\ \dot{v}_y &= \frac{F_{fxw}\sin(\delta_f) - F_{fyw}\cos(\delta_f) + F_{ry}}{m} - \omega_z v_x \\ \dot{w}_z &= \frac{(F_{fxw}\sin(\delta_f) + F_{fyw}\cos(\delta_f))l_f - F_{ry}l_r}{J} \end{aligned} \tag{20}$$

Task 2.2: Verify model for moderate a_y

Using the steering angles from Task 1.4 we will use the simulation to verify that the model reaches a lateral acceleration of 4m/s^2 at 100km/h. Figure 4 shows the vehicle path for each case. Figure 5 confirms that the vehicle reaches 4m/s^2 for every load case.

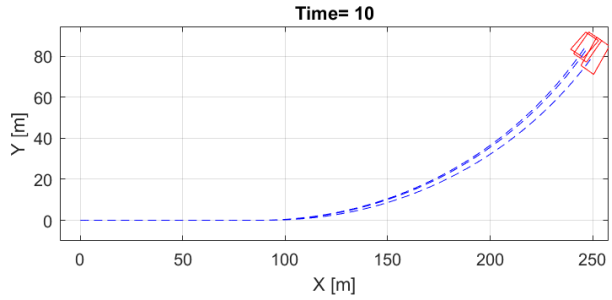
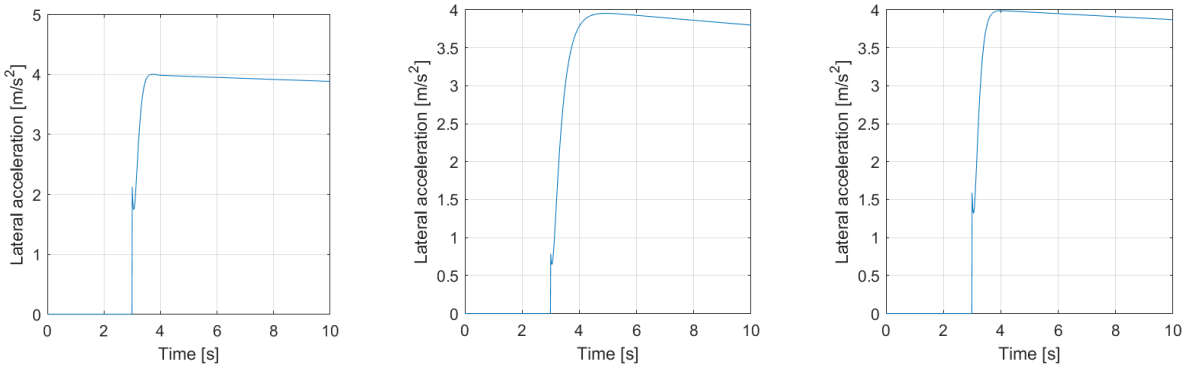


Figure 4: Vehicle path for each case in the XY-plane.

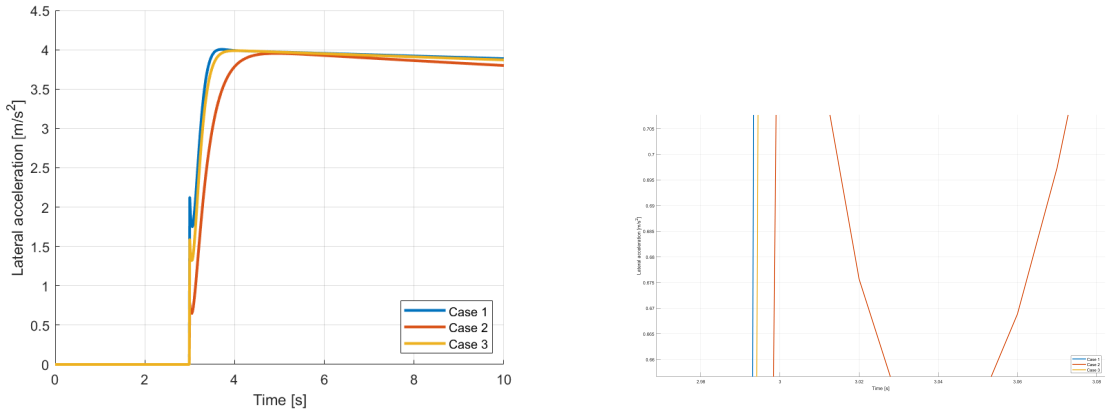


(a) Load case 1 (b) Load case 2 (c) Load case 3

Figure 5: Lateral acceleration for the three load cases

Task 2.3: Transient response

By plotting the lateral acceleration in the same Figure 6a we can observe that for Load Case 1 the lateral acceleration is the greatest compared to the other Load Cases. The angle of the line is steper if you zoom in, see Figure 6b. This means that that the vehicle configuration for Load Case 1 responds the quickest to steering input.



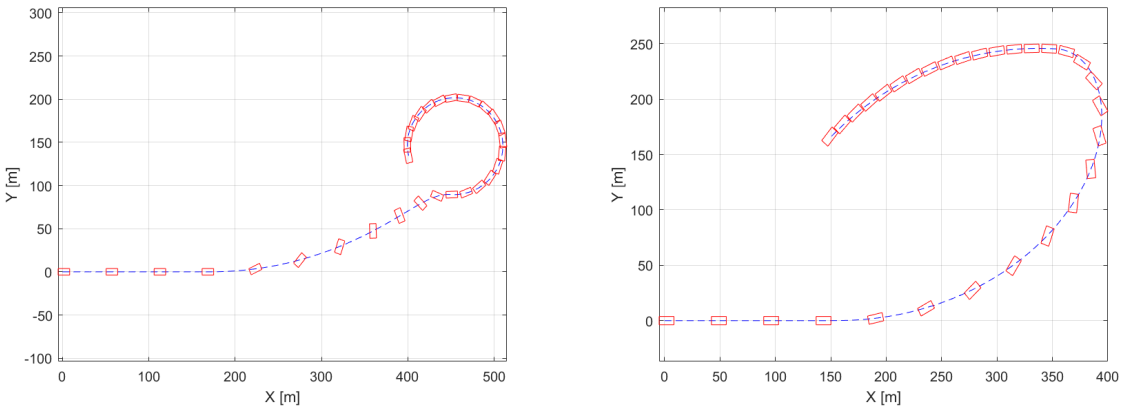
(a) (b)

Figure 6: Transient response.

Task 2.4: Try model at over critical-speed

Using the oversteer load case (Load case 2), above critical speed (200km/h) and an steering angle of 3°, we can observe the vehicle spinning and losing control, see Figure 7a. This is what we would expect since both speed and steering angle is greater than calcuated in Task 1.

If we instead simulate just below the critical speed ($v_{crit} = 47.8\text{m/s} \approx 170\text{km/h}$) and steering angle as calculated in Task 1.4 ($\delta_f = 0.526°$), we can see that the vehicle is just on the limit of becoming unstable, see Figure 7b. This is reasonable since we are just below the critical speed of this vehicle configuration.



(a) Vehicle path for Load Case 2 with: $v_x = 200\text{km/h}$, $\delta_f = 3°$

(b) Vehicle path for Load Case 2 with: $v_x = 170\text{km/h}$, $\delta_f = 0.526°$

Figure 7: Over critical-speed paths.

Task 3: Load transfer

Task 3.1: Add load transfer to model

As described in the instructions the individual tyre stiffnesses will be evaluated which is later used to determine the combined axle stiffness, see Equation 21.

$$
\begin{aligned}
C_f &= c_0 F_{zfl} + c_1 F_{zfl}^2 + c_0 F_{zfr} + c_1 F_{zfr}^2 \\
C_r &= c_0 F_{zrl} + c_1 F_{zrl}^2 + c_0 F_{zrr} + c_1 F_{zrr}^2
\end{aligned} \tag{21}
$$

The Longitudinal load transfer equations can be found in the compendium equation [3.20], see Equation 22. Note that the equations in the Compendium include the static load which we will not include, since we only want the force Δ when load is transferred. The equations are for front and rear axle, and since we want to express the longitudinal forces per wheel, we divide by two.

$$
\begin{aligned}
\Delta F_{flz} &= -ma_x \left(\frac{h}{L}\right)/2 \\
\Delta F_{frz} &= -ma_x \left(\frac{h}{L}\right)/2 \\
\Delta F_{rlz} &= ma_x \left(\frac{h}{L}\right)/2 \\
\Delta F_{rrz} &= ma_x \left(\frac{h}{L}\right)/2
\end{aligned} \tag{22}
$$

The Lateral load transfer equations follow the equations in the compendium [4.43], see Equation 23.

$$
\begin{aligned}
\Delta F_{flz} &= -ma_y \left(\frac{h_{RCf} l_r}{Lw} + \frac{\Delta h}{w} \frac{c_{f,roll}}{c_{vehicle,roll}}\right) \\
\Delta F_{frz} &= ma_y \left(\frac{h_{RCf} l_r}{Lw} + \frac{\Delta h}{w} \frac{c_{f,roll}}{c_{vehicle,roll}}\right) \\
\Delta F_{rlz} &= -ma_y \left(\frac{h_{RCr} l_f}{Lw} + \frac{\Delta h}{w} \frac{c_{r,roll}}{c_{vehicle,roll}}\right) \\
\Delta F_{rrz} &= ma_y \left(\frac{h_{RCr} l_f}{Lw} + \frac{\Delta h}{w} \frac{c_{r,roll}}{c_{vehicle,roll}}\right)
\end{aligned} \tag{23}
$$

Figure 8 shows that the normal loads on the two outer wheels in a left-hand corner increases, which is what we would expect when the load transfers.

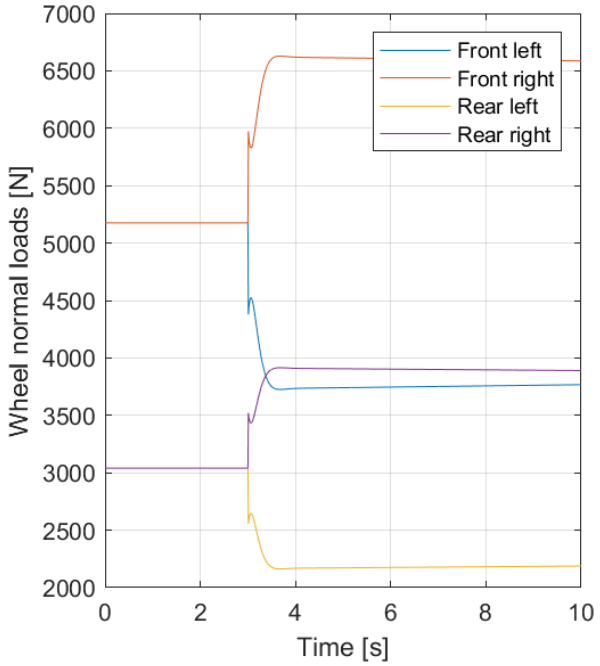


Figure 8: Wheel normal loads [N] for each wheel with load transfer equations for load case 1, using steering angle from Task 1.4

Task 3.2: Influence from load transfer on steering response

After implementing the load transfer model in Simulink, the lateral acceleration for the three load cases was recorded and plotted, see Figure 9.

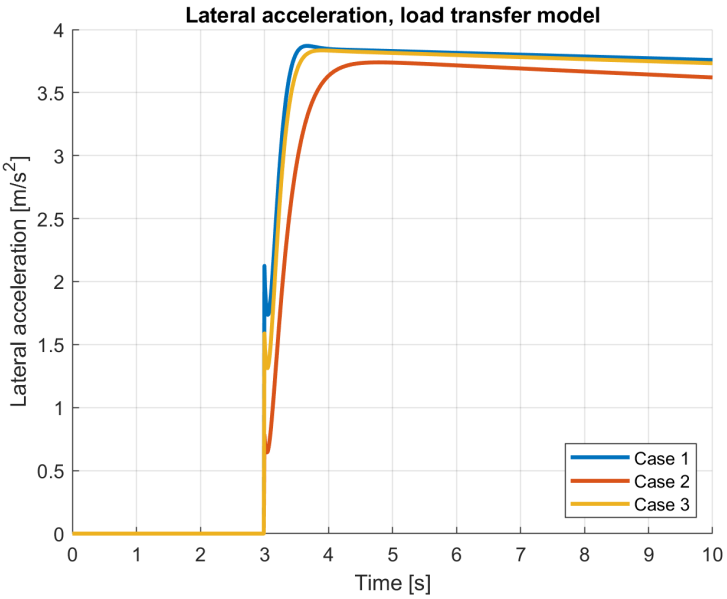


Figure 9: Lateral Acceleration with load transfer model for the three load cases.

From Figure 9 one can see that when load transfer is implemented the lateral acceleration is decreased. With load transfer implemented there is a redistribution of load among the vehicle's wheels. When the vehicle turns, weight is shifted from the inner- to the outer wheels due to centrifugal force and moment of the CoG relative to the roll centre.

The shift in load causes change in vertical tyre force between inner and outer tyres. Due to the non-linear relation between vertical and lateral forces, the total lateral forces will be reduced, compared to a scenario where load transfer is not taken into account. The reduction of lateral forces leads to the car not able to generate the same amount of lateral acceleration.

The ratio of peak lateral acceleration between Task 3.2 and Task 2.2, as per Equation 24, is presented in Table 1. Lateral acceleration was decreased by approximately 5 % for all cases.

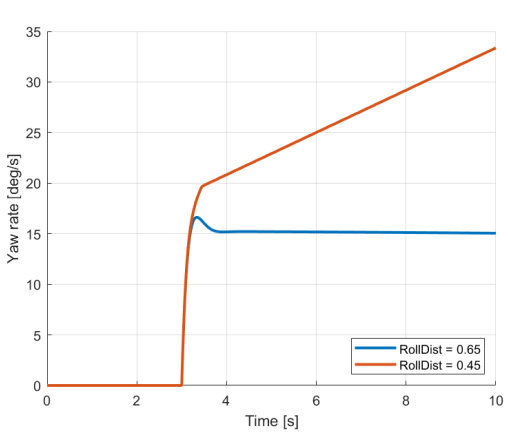
$$\text{Peak lateral acceleration ratio} = \frac{a_{y,max,Task3.2}}{a_{y,max,Task2.2}} \tag{24}$$

Table 1: Change in lateral acceleration between Task 3.2 and Task 2.2 expressed as a ratio between the peak values of acceleration a_y.

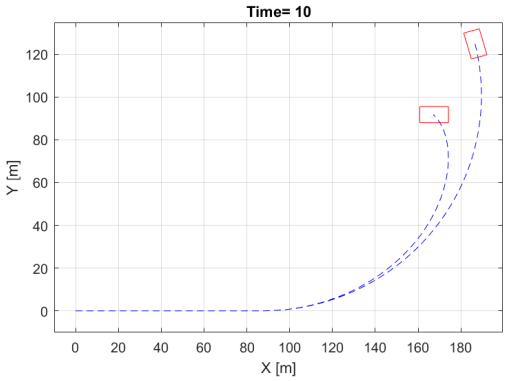
Load case	**Peak lateral acceleration ratio**
1	0.9666
2	0.9460
3	0.9616

Task 3.3: Influence from roll stiffness distribution on steering response

Using Load case 3 with steering wheel angle of 30°, we compare the yaw rate when roll distribution is changed, see Figure 10a. RollDist is defined as $\frac{Front_{rollstiffness}}{Total_{rollstiffness}}$, which means that if RollDist = 1, the front axle is rigid and rear axle has no roll stiffness. Which would result in an understeered car due to reduction in lateral forces on front wheels. This can be confirmed in the graphs, where the larger RollDist (=0.65) results in an understeered car, lower yaw rate and therefore a larger turning radius.



(a) Yaw rate for different roll distributions.



(b) Vehicle path for Load Case 3 with different roll distributions.

Figure 10: Influence from roll stiffness distribution.

Task 3.4: Propose roll stiffness distribution

As determined in Task 3.3 the roll stiffness distribution influences the understeering gradient. To achieve a neutral steer vehicle the understeering gradient $K_u = 0$. Using a RollDist = 0.25 we can see in Figure 11 that the Actual and Neutral steering angle follow each other.

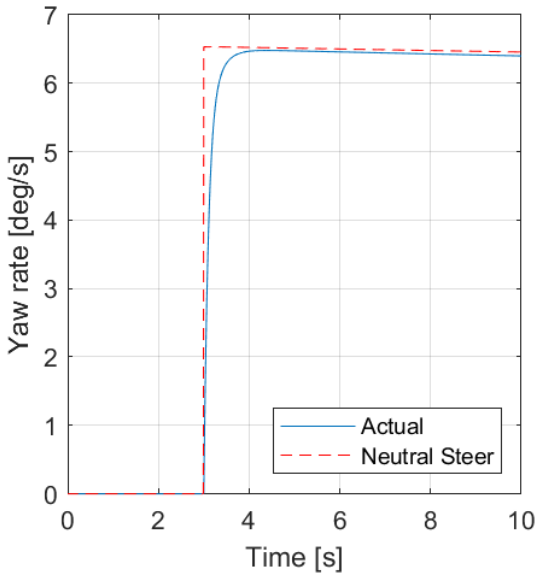


Figure 11: Yaw rate for actual- and neutral steer with roll stiffness distribution = 0.25.

Setting the roll stiffness distribution precisely for neutral steer is deemed unsafe due to the limited flexibility to accommodate changes in weight distribution and road conditions. Loading the car heavily in the rear, for instance, raises the risk of oversteer. Opting for an understeer configuration is generally regarded as safer, as it provides a safety margin and is preferable in potential collision scenarios, favoring a head-on crash over a rear-end impact. A roll stiffness distribution around 0.2 is suggested as a reasonable starting point to achieve this desired understeer characteristic.

Task 4: Combined tyre slip

Task 4.1: Add combined slip to model

Equation 25 was implemented in Simulink block *LatForce* in order to account for combined slip.

$$\begin{aligned} C_{f,corr} &= \left(\frac{\sqrt{(\mu F_{z,front})^2 - F_{x,front}^2}}{\mu F_{z,front}} \right) C_f \\ C_{r,corr} &= \left(\frac{\sqrt{(\mu F_{z,rear})^2 - F_{x,rear}^2}}{\mu F_{z,rear}} \right) C_r \end{aligned} \tag{25}$$

Combined slip is handled on axle level and normal load is the sum of left and right wheels normal forces for front- and rear axle, see Equation 26.

$$\begin{aligned} F_{z,front} &= F_{z,front,left} + F_{z,front,right} \\ F_{z,rear} &= F_{z,rear,left} + F_{z,rear,right} \end{aligned} \tag{26}$$

Observing the yaw rate seen in Figure 12, we can see that the Actual Steer is greater than the Neutral Steer. This means that for the same steering input we "get" more yaw rate than in a neutral steer vehicle, in other words the vehicle is oversteered.

Adding slip in the x-direction will reduce the force component in the Y direction, since the total force must remain the same. Less force in lateral direction will lead to a oversteer vehicle.

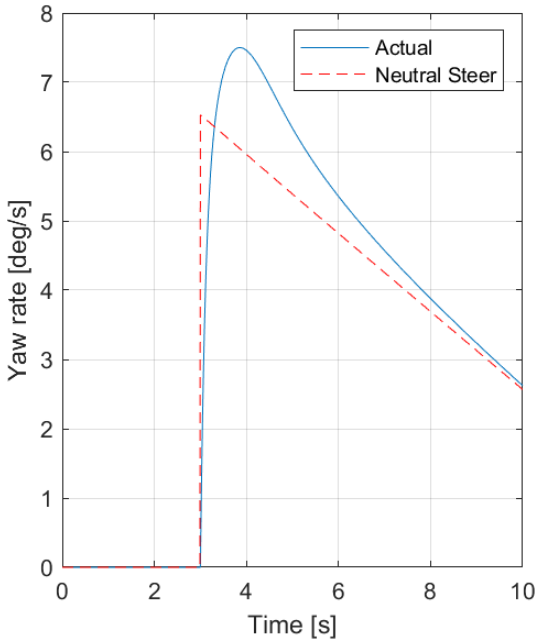
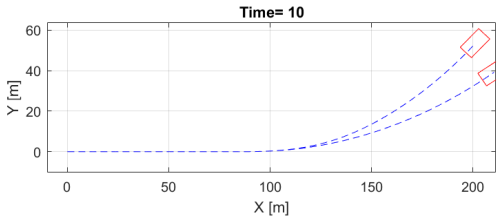


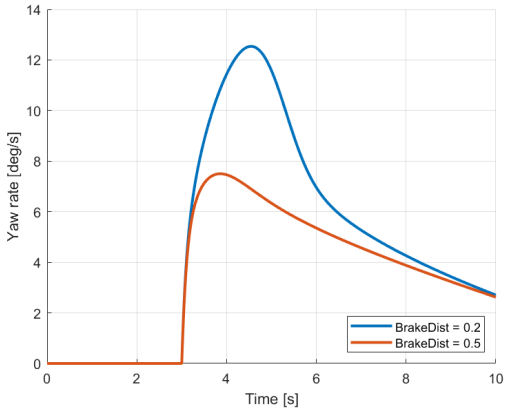
Figure 12: Yaw rate for actual- and neutral steer with combined slip model.

Task 4.2: Influence of brake force distribution

Brake force distribution changes how much braking force is applied for front wheels and rear wheels. BrakeDist = 0.2 means that 20% braking force is on the front wheels and 80% is on the rear wheels. We can conclude from Figure 13a, that increasing BrakeDist will lead to a more understeering vehicle since lateral forces on the front wheels decrease because an increase in vertical and longitudinal forces. Similar to changing the roll stiffness distribution, changing the brake force distribution affects the vehicles understeering gradient. We can therefore expect that an increase in BrakeDist will decrease the yaw rate, see Figure 13b.



(a) Vehicle path for different brake force distributions.



(b) Yaw rate for different brake force distributions.

Figure 13: Influence of brake force distribution.

Task 4.3: Propose brake force distribution.

To achieve neutral steer during braking brake distribution, `brakeDist`, was tuned. To confirm that the vehicle achieved neutral steering, the actual steer curve should exhibit a similar curvature to the neutral steer curve in the yaw rate plot, as illustrated in Figure 14. Neutral steer was achieved with

brake distribution 0.85. This was done by iteratively testing various values of the `brakeDist` parameter until a satisfactory result was obtained.

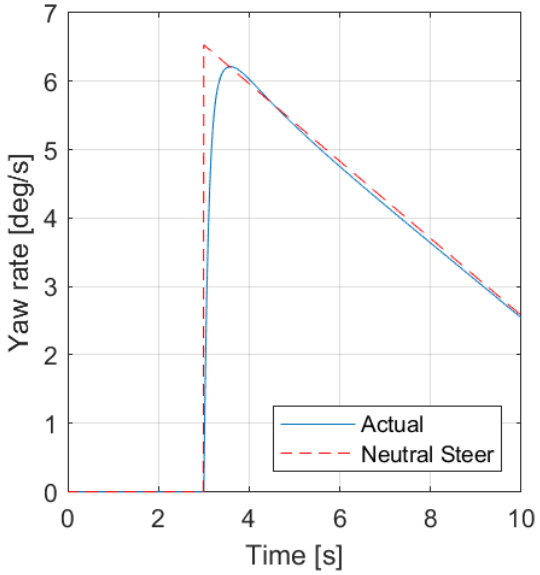


Figure 14: Yaw rate for actual- and neutral steer, brake distribution = 0.85.

Tuning the brake distribution precisely for neutral steer is considered unsafe, as having a margin before the car transitions into oversteer is desirable, aligning with the same reasoning applied to roll distribution. An understeer configuration is preferred for safety, and a brake distribution around 0.9 is recommended to provide a safety buffer before the car reaches a state of oversteer.

Additionally, weight transfer to the front axle is a beneficial mechanism that enhances grip, stabilizes the vehicle, and prevents the rear from sliding out. This is achieved by maintaining lateral forces on the rear wheels, contributing to improved overall stability during various driving conditions.

Task 5: Driving experience in simulator

Task 5.1: Participate in model validation session

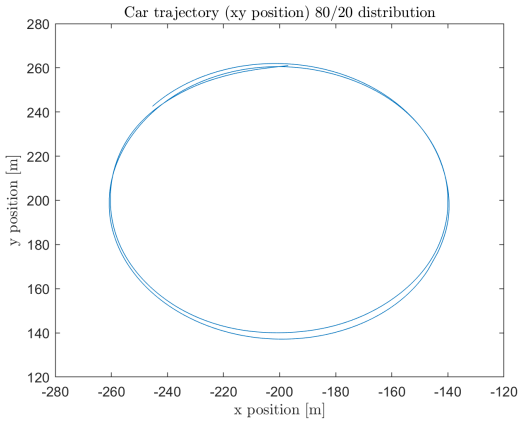
Table 2 shows the results from simulation of the three cases. These values was approved during the validation session.

Table 2: Maximum and minimum values for longitudinal acceleration, lateral acceleration, and yaw velocity for each case.

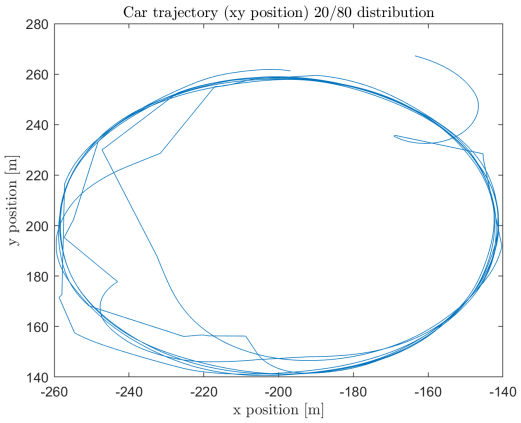
Case	$a_{x,max}[\mathrm{m/s^2}]$	$a_{x,min}[\mathrm{m/s^2}]$	$a_{y,max}[\mathrm{m/s^2}]$	$a_{y,min}[\mathrm{m/s^2}]$	$\dot{\omega}_{y,max}[\mathrm{m/s^2}]$	$\dot{\omega}_{y,min}[\mathrm{m/s^2}]$
1	0	-0.013	1.9	0	4.95	0
2	0	-1.2	0	0	0	0
3	0	-0.6	0.575	0	1.5	0

Task 5.2: Participate in simulator drive session

Figures 15a and 15b illustrates the trajectories for the two cases. Figure 15a has a 80/20 load distribution which means that the car is understeered. Achieving a consistent turning radius for this load case was relatively straightforward, leading to a uniform curve in the trajectory plot. In the second instance, a 20/80 load distribution was applied, resulting in oversteering of the car. This made it challenging to keep the vehicle on the circular path, as it was prone to slipping and spinning, as can be seen in Figure 15b.



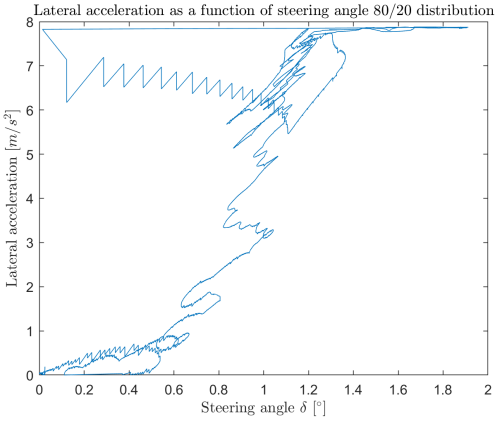
(a) Car trajectory for 80/20 distribution.



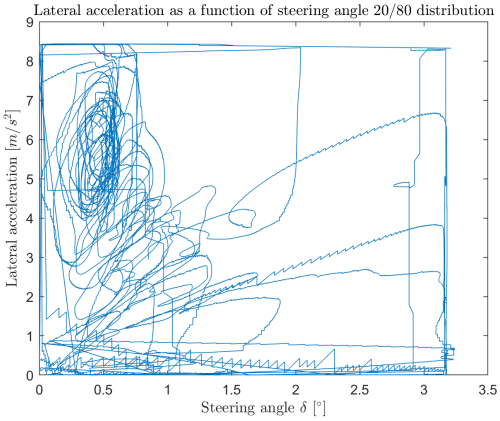
(b) Car trajectory for 20/80 distribution.

Figure 15: xy positions for the two load distributions.

Figures 16a and 16b illustrates lateral acceleration as a function of steering angle for the two load distributions. As can be seen in Figure 16a, when increasing the velocity of the car the driver has to gradually increase the steering angle to increase lateral acceleration and maintain the car on the circular path. The car is understeerinng. Figure 16b represents lateral acceleration as a function of steering angle for load distribution 20/80 (oversteered car). While the trend may not be apparent in the plot, it indicates a reduction in the steering angle with increases in both vehicle speed and lateral acceleration. This observed pattern is typical for an oversteered car.



(a) Lateral acceleration 80/20 distribution.



(b) Lateral acceleration 20/80 distribution.

Figure 16: Lateral acceleration as a function of steering angle for the two load distributions.

From Figure 16a and 16b, we can guess an approximation of how the handling diagram can look by observing the trend, see Figure 17. For the understeer vehicle we need more and more steering angle to achieve the same lateral acceleration. The oversteer vehicle was the opposite, less and less steering angle was needed to achieve the same lateral acceleration, until we reach critical speed and losses control.

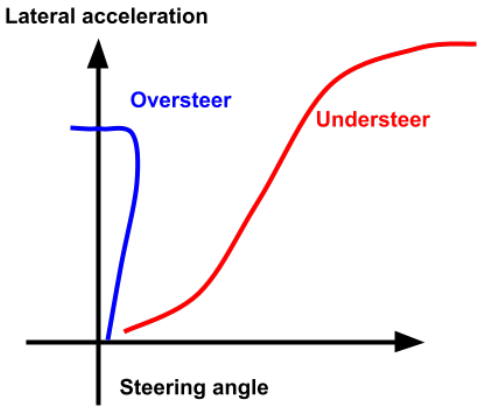


Figure 17: Handling diagram approximation.

Task 5.3: Documentation in assignment report

The model create was tested in the Caster simulator to verify the behaviour of the vehicle during certain manoeuvres. This includes steady-state cornering and turning while braking, using different setups. The model behaved as expected in the different scenarios.

Pros about using a simulator to verify the model is that changing the setup can be done within minutes, followed by a simulated test to evaluate the change. This is more cost effective and less time consuming than real world testing. However, it is important that you can verify that the model is a good representation of what will happen in the real world. Therefore good knowledge about vehicle dynamics and assumptions that can be considered is a must. Otherwise the model will give inaccurate results and will not give a good representative of the real world.

The model used in this case is a One-track model which mainly focuses on the lateral dynamics of the vehicle. We still implement longitudinal load transfer to get the affects of braking and acceleration. However, implementing a Two-track model would give greater accuracy when considering longitudinal dynamics. In this case the One-track model provided a good enough representation and the vehicle behaved as expected when the setup changed. It is hard to judge if the Two-track model would have been worth implementing for just these tests.

References

[1] Wallpaperuse (2020) - SAAB 9-3. Published online at wallpaperuse.com [Creative Commons] Retrieved from: https://www.wallpaperuse.com/vien/hwxhxm/ [Online Resource]

1 Appendix 1

Listing 1: MATLAB code Task 1.1 - 1.5

```
1
2  %% Design Task 2
3  % Gabriel Wendel, Erik Lydig
4  close all;
5  clear all;
6  clc
7  %% Task 1.2, 1.3, 1.4
8  % Load Saab 9-3 parameters
9  run('InitModel.m')
10
11 % Given length ratios
12 length_ratio = [0.37 0.63 0.47];
13
14 l_f = [length_ratio(1)*vehicleData.L length_ratio(2)*vehicleData.L...
15     length_ratio(3)*vehicleData.L];
16 l_r = [(1-length_ratio(1))*vehicleData.L (1-length_ratio(2))*
       vehicleData.L...
17     (1-length_ratio(3))*vehicleData.L];
18
19 % Given parameters
20 c0 = 30.7;                % [1/rad]
21 c1 = -0.00235;            % [1/(N*rad)]
22 v_x = 100/3.6;            % velocity in x-direction [m/s]
23 acc_lat = 4;              % lateral acceleration [m/s^2]
24
25 % Initialize arrays
26 K_u = zeros(1,length(length_ratio));
27 v_xcrit = zeros(1,length(length_ratio));
28 v_xchar = zeros(1,length(length_ratio));
29 steer_ang = zeros(1,length(length_ratio));
30
31 % Loop over the load cases
32 for i=1:length(length_ratio)
33
34     % Calculate longitudinal force on front- and rear wheel
35     F_fz = vehicleData.m * vehicleData.g * (1-length_ratio(i))/2; %
           One wheel
36     F_rz = vehicleData.m * vehicleData.g * length_ratio(i)/2;     %
           One wheel
37
38     % Using longitudinal force, determine cornering stiffness
39     C_f = (c0*F_fz + c1 * F_fz^2)*2; % Multiply by two to get for rear
           axle
40     C_r = (c0*F_rz + c1 * F_rz^2)*2; % Multiply by two to get for rear
           axle
41
42     % Understeer gradients
43     K_u(i) = ((C_r*l_r(i))-(C_f*l_f(i))) / (C_f*C_r*vehicleData.L); %
           [1/N] or [rad/N] or [rad/(m/s^2)]
44
45     % Critical and characteristic speeds using understeer gradients
46     v_xcrit(i) = sqrt(vehicleData.L / (-K_u(i)*vehicleData.m));
47     v_xchar(i) = sqrt(vehicleData.L / (K_u(i)*vehicleData.m));
```

```
48
49      % Steering angle [rad]
50      steer_ang(i) = acc_lat*(vehicleData.L + K_u(i)*vehicleData.m*v_x
            ^2) / v_x^2; % From Equ 4.21 in Compendium
51
52  end
53
54  %% Task 1.5: Steering wheel angle plotted against varying operating
        points
55
56  radius = 200;                   % Curve radius [m]
57  v_x_points = 1:0.01:50;         % Array of operating points (velocity in
        x-direction) [m/s]
58
59  % Calculate steering angle using function "steering_angles"
60  steer_ang_1 = Steering_func(v_x_points, radius, K_u(1), vehicleData);
61  steer_ang_2 = Steering_func(v_x_points, radius, K_u(2), vehicleData);
62  steer_ang_3 = Steering_func(v_x_points, radius, K_u(3), vehicleData);
63
64  % Plot steering angle [rad] as a function of operating point [v_x] for
        all
65  % three length ratios
66  figure(1)
67  hold on
68  plot(v_x_points, steer_ang_1)
69  plot(v_x_points, steer_ang_2)
70  plot(v_x_points, steer_ang_3)
71  ylim([0, 0.45]);
72  xlabel('Operating points, $v_x$ [m/s]','Interpreter','latex')
73  ylabel('Steering Angle [rad]','Interpreter','latex')
74  legend('Load case 1','Load case 2', 'Load case 3','Interpreter','latex
        ')
75  title('Steering Angle vs Operating points, $v_x$','Interpreter','latex
        ')
76
77
78  % Function that computes Steering Angle for given operating point (v_x
        ),
79  % curve radius, understeer gradient and vehicle data (Saab 9-3)
80  function steering_angles = Steering_func(v_x_points, radius, K_u,
        vehicleData)
81      steering_angles = (vehicleData.L/radius + K_u*(vehicleData.m.*
            v_x_points.^2) / ...
82          radius)*vehicleData.steeringRatio;
83  end
```

Appendix 2

Listing 2: MATLAB code Task 2.1 - 2.2

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % Vehicle Dynamics, MMF062, 2020
3  % Vertical assignment, Task 2
4  %
5  %
6  clear all;
```

```
close all;
clc;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Load parameters from file "InitParameters.m"

InitParametersSkeleton

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Task 2.1
%
% Front wheel
%
% Calculate road spectrum
roadSpectrum = zeros(length(angularFrequencyVector),1);

for i = 1 : length(angularFrequencyVector)
    % Equation 5.29 from the compendium
    roadSpectrum(i,:) = vehicleVelocity^(roadWaviness-1)*roadSeverity*
        angularFrequencyVector(i)^(-roadWaviness);
end

% Calculate transfer functions for front wheel Zr to Ride and Tyre
    force
% You can use the code from Task 1

% Consider one single front wheel (same expressions as Task 1)
sprungMassFront = 0.5*(distanceCogToRearAxle/wheelBase)*
    totalSprungMass;
unsprungMassFront =  0.25*totalUnsprungMass;

% Identify A and B matrix (same matrices as task 1)

Af =  [0 0 1 0;
       0 0 0 1;
       -(tireStiff + frontWheelSuspStiff)/unsprungMassFront
           frontWheelSuspStiff/unsprungMassFront -frontWheelSuspDamp/
           unsprungMassFront frontWheelSuspDamp/unsprungMassFront;
       frontWheelSuspStiff/sprungMassFront -frontWheelSuspStiff/
           sprungMassFront frontWheelSuspDamp/sprungMassFront -
           frontWheelSuspDamp/sprungMassFront];
Bf =  [0; 0; tireStiff/unsprungMassFront; 0];

% Calculate transfer functions for
% 1)front wheel Zr to Ride,
% 3) front wheel Zr to Tyre force
% similar to Task 1 define C and D matrices for both cases
C1f = [frontWheelSuspStiff/sprungMassFront -frontWheelSuspStiff/
    sprungMassFront frontWheelSuspDamp/sprungMassFront -
    frontWheelSuspDamp/sprungMassFront];
D1f = 0;

C3f = [-tireStiff 0 0 0];
D3f = tireStiff;

transferFunctionFrontZrToRide = zeros(length(angularFrequencyVector)
    ,1);
```

```
transferFunctionFrontZrToForce = zeros(length(angularFrequencyVector)
    ,1);

for j = 1 : length(angularFrequencyVector)
    % Calculate H(w) not the absolut value |H(w)| (same as task 1)
    transferFunctionFrontZrToRide(j,:) = C1f*inv(1i*
        angularFrequencyVector(j)*eye(4) - Af)*Bf + D1f;
    transferFunctionFrontZrToForce(j,:) = C3f*inv(1i*
        angularFrequencyVector(j)*eye(4) - Af)*Bf + D3f;
end

% Calculate acceleration and tyre force response spectrum
psdAcceleration = zeros(length(angularFrequencyVector),1);
psdForce = zeros(length(angularFrequencyVector),1);

for m = 1 : length(angularFrequencyVector)
    % Equation 5.30 from compendium
    psdAcceleration(m,:) = abs(transferFunctionFrontZrToRide(m))^2*
        roadSpectrum(m);
    psdForce(m,:) = abs(transferFunctionFrontZrToForce(m))^2*
        roadSpectrum(m);
end

% Calculate rms values of acceleration and tyre force
msAcceleration = 0;
msForce = 0;

for n = 1 : length(angularFrequencyVector)
    % Equation 3.51 from compendium
    msAcceleration = msAcceleration + psdAcceleration(n)*
        deltaAngularFrequency;
    msForce = msForce + psdForce(n)*deltaAngularFrequency;
end

rmsAcceleration = sqrt(msAcceleration);
rmsForce = sqrt(msForce);

figure(100);
semilogy(frequencyVector,psdAcceleration);grid
xlabel('Frequency [Hz]');
ylabel('Acceleration [(m/s^2)^2/Hz]');
title('Sprung mass acceleration PSD');
legend(['rms value = ',num2str(rmsAcceleration)])

figure;
semilogy(frequencyVector,psdForce);grid
xlabel('Frequency [Hz]');
ylabel('Acceleration [(N)^2/Hz]');
title('Tyre force PSD');
legend(['rms value = ',num2str(rmsForce)])

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Task 2.2
%
% Front wheel
```

```
rmsAcceleration = zeros(length(frontWheelSuspStiffVector),length(
    frontWheelSuspDampVector));
rmsForce = zeros(length(frontWheelSuspStiffVector),length(
    frontWheelSuspDampVector));

for ind1 = 1 : length(frontWheelSuspStiffVector)

    for ind2 = 1 : length(frontWheelSuspDampVector)

        % Update suspension stiffness and damping
        frontWheelSuspStiff = frontWheelSuspStiffVector(ind1);
        frontWheelSuspDamp = frontWheelSuspDampVector(ind2);
        % Update A and C matrices
        Af =  [0 0 1 0;
               0 0 0 1;
               -(tireStiff + frontWheelSuspStiff)/unsprungMassFront
                   frontWheelSuspStiff/unsprungMassFront -
                   frontWheelSuspDamp/unsprungMassFront
                   frontWheelSuspDamp/unsprungMassFront;
               frontWheelSuspStiff/sprungMassFront -
                   frontWheelSuspStiff/sprungMassFront
                   frontWheelSuspDamp/sprungMassFront -
                   frontWheelSuspDamp/sprungMassFront];

        C1f = [frontWheelSuspStiff/sprungMassFront -
           frontWheelSuspStiff/sprungMassFront frontWheelSuspDamp/
           sprungMassFront -frontWheelSuspDamp/sprungMassFront];
        C3f = [-tireStiff 0 0 0];

        % Calculate transfer functions for front wheel Zr to Ride and
           Tyre force

            transferFunctionFrontZrToRide = zeros(length(
               angularFrequencyVector),1);
            transferFunctionFrontZrToForce = zeros(length(
               angularFrequencyVector),1);

            for j = 1 : length(angularFrequencyVector)
                % Calculate H(w) not the absolut value |H(w)|
                transferFunctionFrontZrToRide(j,:) = C1f*inv(1i*
                   angularFrequencyVector(j)*eye(4) - Af)*Bf + D1f;
                transferFunctionFrontZrToForce(j,:) = C3f*inv(1i*
                   angularFrequencyVector(j)*eye(4) - Af)*Bf + D3f;
            end


        % Calculate acceleration and tyre force response spectrum
        psdAcceleration = zeros(length(angularFrequencyVector),1);
        psdForce = zeros(length(angularFrequencyVector),1);

        for m = 1 : length(angularFrequencyVector)
            psdAcceleration(m,:) = abs(transferFunctionFrontZrToRide(m
               ))^2*roadSpectrum(m);
            psdForce(m,:) = abs(transferFunctionFrontZrToForce(m))^2*
               roadSpectrum(m);
        end
```

```
141
142         % Calculate rms values of acceleration and tyre force
143         msAcceleration = 0;
144         msForce = 0;
145
146         for n = 1 : length(angularFrequencyVector)
147             msAcceleration = msAcceleration + psdAcceleration(n)*
                    deltaAngularFrequency;
148             msForce = msForce + psdForce(n)*deltaAngularFrequency;
149         end
150
151         rmsAcceleration(ind1,ind2) = sqrt(msAcceleration);
152         rmsForce(ind1,ind2) = sqrt(msForce);
153
154     end
155 end
156
157 % Plot rms values vs damping
158 figure;
159 plot(frontWheelSuspDampVector,rmsAcceleration);grid
160 legend(num2str(frontWheelSuspStiffVector'));
161 xlabel('Damping coefficient [Ns/m]');
162 ylabel('Acceleration RMS [m/s^2]');
163 title('Sprung mass acceleration vs chassis damping for various spring
        stiffness');
164 axis([0 10000 0 3]);
165
166 figure;
167 plot(frontWheelSuspDampVector,rmsForce);grid
168 legend(num2str(frontWheelSuspStiffVector'));
169 xlabel('Damping coefficient [Ns/m]');
170 ylabel('Tyre Force RMS [N]');
171 title('Dynamic tyre force vs chassis damping for various spring
        stiffness');
172 axis([0 10000 0 1400]);
173
174 % Identify optimal damping values for each stiffness value
175 [optimalRmsAcceleration,iOptimalRmsAcceleration] = min(rmsAcceleration
        ,[],2);
176 [optimalRmsForce,iOptimalRmsForce] = min(rmsForce,[],2);
177
178 % Plot optimal damping for Ride and Tyre force vs Stiffness
179 figure;
180 % Use optimal rms acc/force index
181 plot(frontWheelSuspStiffVector, frontWheelSuspDampVector(
        iOptimalRmsAcceleration));
182 hold on
183 plot(frontWheelSuspStiffVector, frontWheelSuspDampVector(
        iOptimalRmsForce));
184 grid
185
186 frontWheelSuspDamp_optimal_points = frontWheelSuspDampVector(
        iOptimalRmsAcceleration);
187 frontWheelSuspStiff_optimal_points = frontWheelSuspDampVector(
        iOptimalRmsForce);
188
```

```
% Plot optimal damping coefficient as a mean
for i=1:length(frontWheelSuspDampVector(iOptimalRmsForce))
    mean_optimal_damping(i) = (frontWheelSuspDamp_optimal_points(i) +
        frontWheelSuspStiff_optimal_points(i))/2;
end
scatter(frontWheelSuspStiffVector,mean_optimal_damping,'o','filled', '
   MarkerEdgeColor', 'k', 'MarkerFaceColor', 'g', 'SizeData', 50)
xlabel('Spring stiffness [N/m]');
ylabel('Damping coefficient [Ns/m]');
title('Optimal damping vs spring stiffness');
legend('rmsAcce','rmsForce','mean value of optimal rms');
axis([10000 65000 0 4000]);
```

Appendix 3

Listing 3: MATLAB code Task 3.1 - 3.2

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Vehicle Dynamics, MMF062, 2020
% Vertical assignment, Task 3

clear all;
close all;
clc;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Load parameters from file "InitParameters.m"

InitParametersSkeleton

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Task 3.1

numberOfTripsPerDay = totalDrivingTime/...
    (distanceSmoothRoad/vehicleVelocitySmooth+distanceRoughRoad/
        vehicleVelocityRough+distanceVeryRoughRoad/
        vehicleVelocityVeryRough);

% i)
% Calculate road spectrum
roadSpectrumSmooth = zeros(length(angularFrequencyVector),1);
roadSpectrumRough = zeros(length(angularFrequencyVector),1);
roadSpectrumVeryRough = zeros(length(angularFrequencyVector),1);

for j = 1 : length(angularFrequencyVector)
    roadSpectrumSmooth(j,:) = vehicleVelocitySmooth^(
        roadWavinessSmooth-1)*roadSeveritySmooth*angularFrequencyVector
        (j)^(-roadWavinessSmooth);
    roadSpectrumRough(j,:) = vehicleVelocityRough^(roadWavinessRough
        -1)*roadSeverityRough*angularFrequencyVector(j)^(-
        roadWavinessRough);
    roadSpectrumVeryRough(j,:) = vehicleVelocityVeryRough^(
        roadWavinessVeryRough-1)*roadSeverityVeryRough*
        angularFrequencyVector(j)^(-roadWavinessVeryRough);
end

% Calculate transfer functions for front wheel Zr to Ride
```

```
% You can use result from Task 1

sprungMassFront = 0.5*(distanceCogToRearAxle/wheelBase)*
    totalSprungMass;
unsprungMassFront =  0.25*totalUnsprungMass;

Af =  [0 0 1 0;
       0 0 0 1;
       -(tireStiff + frontWheelSuspStiff)/unsprungMassFront
           frontWheelSuspStiff/unsprungMassFront -frontWheelSuspDamp/
           unsprungMassFront frontWheelSuspDamp/unsprungMassFront;
       frontWheelSuspStiff/sprungMassFront -frontWheelSuspStiff/
           sprungMassFront frontWheelSuspDamp/sprungMassFront -
           frontWheelSuspDamp/sprungMassFront];
Bf =  [0; 0; tireStiff/unsprungMassFront; 0];

C1f = [frontWheelSuspStiff/sprungMassFront -frontWheelSuspStiff/
    sprungMassFront frontWheelSuspDamp/sprungMassFront -
    frontWheelSuspDamp/sprungMassFront];
D1f = 0;

transferFunctionFrontZrToRide = zeros(length(angularFrequencyVector)
    ,1);

for j = 1 : length(angularFrequencyVector)
    transferFunctionFrontZrToRide(j,:) = C1f*inv(1i*
        angularFrequencyVector(j)*eye(4) - Af)*Bf + D1f;
end

% Calculate acceleration response spectrum for all roads

psdAccelerationSmooth = zeros(length(angularFrequencyVector),1);
psdAccelerationRough = zeros(length(angularFrequencyVector),1);
psdAccelerationVeryRough = zeros(length(angularFrequencyVector),1);

for j = 1 : length(angularFrequencyVector)
    psdAccelerationSmooth(j,:) = abs(transferFunctionFrontZrToRide(j))
        ^2*roadSpectrumSmooth(j);
    psdAccelerationRough(j,:) = abs(transferFunctionFrontZrToRide(j))
        ^2*roadSpectrumRough(j);
    psdAccelerationVeryRough(j,:) = abs(transferFunctionFrontZrToRide(
        j))^2*roadSpectrumVeryRough(j);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ii)
% Calculate rms values weighted according to ISO2631
[weightedRmsAccelerationSmooth] = CalculateIsoWeightedRms(
    frequencyVector,psdAccelerationSmooth)
[weightedRmsAccelerationRough] = CalculateIsoWeightedRms(
    frequencyVector,psdAccelerationRough)
[weightedRmsAccelerationVeryRough] = CalculateIsoWeightedRms(
    frequencyVector,psdAccelerationVeryRough)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% iii)
```

```
% Calculate time averaged vibration exposure value for 8h period

timeOnSmoothRoad = (distanceSmoothRoad / vehicleVelocitySmooth)*
    numberOfTripsPerDay;
timeOnRoughRoad = (distanceRoughRoad / vehicleVelocityRough)*
    numberOfTripsPerDay;
timeOnVeryRoughRoad = (distanceVeryRoughRoad /
    vehicleVelocityVeryRough)*numberOfTripsPerDay;

timeWeightedMsAcceleration = (weightedRmsAccelerationSmooth^2*
    timeOnSmoothRoad + ...
     weightedRmsAccelerationRough^2*timeOnRoughRoad +
         weightedRmsAccelerationVeryRough^2*timeOnVeryRoughRoad) ...
     / (timeOnSmoothRoad + timeOnRoughRoad + timeOnVeryRoughRoad);

timeWeightedRmsAcceleration = sqrt(timeWeightedMsAcceleration);

disp(['timeWeightedRmsAcceleration =' num2str(
    timeWeightedRmsAcceleration),' m/s2 (1.15)',...
     ', numberOfTripsPerDay = ' num2str(numberOfTripsPerDay)]);
disp(['vehicleVelocitySmooth = ' num2str(vehicleVelocitySmooth*3.6),'
    km/h',...
     ', vehicleVelocityRough = ' num2str(vehicleVelocityRough*3.6),' km
         /h',...
     ', vehicleVelocityVeryRough = ' num2str(vehicleVelocityVeryRough
         *3.6),' km/h'])
```