Design Tasks

TME102- Vehicle Motion and Control

Adam Dyson

Michele Ghisleni

Abstract

Writing this report, we decided to divide it into four chapters, one for each design task. Within each of the tasks there will be references to the Matlab files used to carry out the task requests. Where possible, we will add part of the Matlab code or screenshots of the Simulink files for completeness. There is a references section at the end of the report with all the appendix. We decided not to attach the CarMaker files to our report as they do not differ substantially from the instructions given in the task text.

1. Normal force estimator

We created the test track using CarMaker.

1.1. Create the normal force estimator

We solved this task using the files TME102_Task1.slx and TME102_NormalLoadEstimator.m. The first one contains the Simulink model that allows us to connect CarMaker with Matlab and save the measurements made by the sensors during the simulation in CarMaker. The second allows us to calculate the normal force estimator and evaluate its goodness.

First, we had to copy some parameters of the car used by the CarMaker model (Parameters.Car) and insert an inertial sensor to measure transactional and rotational acceleration (Parameters.Car.Sensors.Inertial). Finally, we used the formulas in the task text to calculate the force estimators. (Appendix no. 1).

We would like to emphasise that the exact values of Cf and Cr have not been calculated as only the ratio between the two is important and the exact value does not matter. Furthermore, Mz is assumed to be very small as limited yaw torque is generated by the longitudinal tyre forces in our case. It is also difficult to determine with our sensor setup as tyre slip needs to be known.

Appendix no.2 shows the four plots of the four normal forces. Plotted in the same graph are both the real and estimated forces are plotted in the same graph. Note how the course of the forces respects the conformation of the road on which the simulation was carried out, and this is also why the plot of the forces varies as the simulation parameters vary. The only relevant point is what happens at about 45 sec. Exactly when the car is about to make the curve inclined with 15 % banking, the value of the estimated forces and the actual forces become very different. Theoretically, this is due to the fact that the sensors are aligned with the car and therefore do not take into account the change in the inclination of the road and thus the change in the weight force. We tried to modify the equations by inserting instead of the acceleration of gravity, the normalised acceleration along the z-axis (az_n = az+g). In this way, we hoped to compensate for the change in weight force, but we were only able to improve the final result slightly. The improvement was also noticeable thanks to the RMS value, which we will now discuss.

To verify the accuracy of the measurement, we calculated three RMS values, one for the actual forces, one for the estimated forces and one for the difference between the two. (Appendix no.3) The aim is to have two similar RMS values for the individual forces and to have a value tending towards zero for the difference between the two. This is not exactly verified, but the results obtained are in line with the assumptions, neglection, and measurement accuracies. The RMS_difference array indicates the values of the difference between the actual and estimated forces. The result of the RMS calculation gives the values of [500, 731, 612, 625] for the normal forces [FL, FR, RL, RR] respectively. If we do not take into account the values obtained when the machine does the inclined curve, the previous RMS values decrease a lot.

Finally, we were asked to implement a warning function to alert the driver if any vertical wheel load falls below 1/3 of the static case. To solve this question, we implemented a subsystem on Simulink that would calculate the RMS of the just taken normal force measurements instant by instant and compare it with the current normal force value. You can see a screenshot of the implementation in the Appendix no.4. We had thought about using the Assertion block, but we noticed that every time we introduced this block, the simulation would slow down a lot and freeze at second 120.02. We therefore opted for a less efficient but working solution. Within the Matlab file used for this task, it is possible to find a section that uses the "warning" function to print an error message on the screen whenever one of the four forces exceeds the allowed limit value. We show this implementation in Appendix no.5.

2. Yaw rate frequency response

This task was solved using the TME102_OneTrackModel.m, TME102_Task2.slx and TME102_yawRateResponse_b.m files. With regard to the frequency response obtained from the single track model, it was relatively easy to come to a conclusion. Unfortunately, we were not so lucky with the complete machine model and therefore decided not to continue with the task.

In the Simulink file, all implementations were done in TME102_Task2 - CarMaker - VehicleControl - SubSystem, where SubSystem is nothing more than a copy of the CreateBus VhclCtrl subsystem. In this section we modified the signals for steeering angle, steering angle velocity and steering angle acceleration. In particular, we have imposed an external signal so that the car traces a trajectory with a sinusoidal steering angle trend (alternately curving to the right and left with the same amplitude) and an increasing frequency. In particular, we made the frequency of the steering angle linearly dependent on time so that the more time passes, the faster the sinusoidal signal of the steering angle. We were then able to use Simulink's "Derivative" blocks to calculate speed and acceleration. In Appendix no.6 we can see a screenshot of the implementation on Simulink.

2.1. Single track model

For this task we had to estimate the value of the cornering stiffness based on the lateral force and slide slip angle, an output of the CarMaker simulation. This is only valid for low speed cornering.

In Appendix no.7 we can see the three Bode plots depicting the three frequency responses relative to the speeds of 80km/h, 110km/h and 130km/h. It can be seen that the three results have only minor differences between them. In particular, the frequency response has a greater initial value as speed increases. Furthermore, the higher the speed, the more the frequency response moves away from what is the typical pattern of a first-order system. It almost seems as if there is an overshoot around the resonance frequency of about 3 rad/s that grows as the speed increases. The TME102_OneTrackModel.m file is used to plot this graph.

2.2. CarMaker model

Here, we could not complete the task. We tried several ways:
- Using the Fourier transform.
- Using Matlab's built-in function to calculate the estimated transfer function.
- Using the 'findpeaks' function to find the peaks of the steering angle (which, let's remember, has a sinusoidal trend with increasing frequency as time passes) and then use them to plot a Bode plot after performing a kind of discrete sampling on the values of the peaks obtained.

Unfortunately, none of this seemed to work. In fact, the best result we got was the one shown in Appendix no.8 However, we knew from the theory studied in class that the shape of the frequency response of the CarMaker model should be similar to that of the Single Track Model.

In particular, it should be noted that at high speeds tyre slip becomes a predominant factor within the model, and therefore only at high speeds should the two models begin to differ. In particular, due to the effect of tyre slip, the magnitude of the yaw should decrease much more rapidly

3. Speed controller

This task was solved using the TME102_Task3.slx, TME102_Task3_2d.slx (for the second part of the task) files. The entire implementation took place on Simulink. CarMaker was only used to create a simple road to be used in simulation and to obtain measurements.

All implementations made in the Simulink model were made within TME102_Task3 - CarMaker - VehicleControl - SubSystem, where SubSystem is nothing more than a copy of the CreateBus VhclCtrl subsystem.

For the speed profile, we used a very simple but effective way. We used an Add block on Simulink to combine various signals (a constant signal, a ramp and three steps) delayed by a reasonable amount of time to allow the vehicle to reach all the speeds required by the task. (Appendix no.9). First, we have a ramp with a settable value in Matlab (we used 5). After a time equal to ramp/40, i.e. the time it takes for the speed to reach a value of 40, we enter a constant signal equal to 40km/h via a switch. For the next three steps, we used the 'Step' block, selecting the correct step times.

The speed controller is a normal proportional controller that has been selected to try to find the right combination of execution speed and robustness. Its value is 0.5. We model the system that is 'seen' by the controller is as a first-order system equal to:

$$\frac{1}{s+1}$$

This is the transfer function of the machine used in the slide folder LongCtrl from the lectures.

Speed control works via a closed-loop control system, where the output signal is compared with the actual longitudinal speed of the machine (Car.vx) to see if the machine needs to accelerate or brake to follow the required speed profile. The compared signal is then divided into two signals: the first, positive, represents the acceleration signal; the second, negative, represents the deceleration signal. Both signals are normalised to a value between 0 and 1 in order to communicate efficiently with CarMaker. In fact, these two normalised signals are fed into CarMaker via the gas and brake pedal signals. It should be noted that the brake signal was limited from 0 to 0.5 because we noticed that a brake signal greater than 0.5 made the car slow down too much and prevented the true speed profile from being followed during the simulation.

Appendix no.10 shows all the implementation.

3.1. Curve speed limitation

For this point, we followed the same reasoning used in the previous point. In particular, before implementing the lateral acceleration limitation, we noticed that for different values of the proportional controller, we obtained different results in the simulations, but always the car could not keep itself on the road. To solve this problem, we implemented a series of blocks on Simulink that would compare the fixed speed of 70 km/h with the speed value of the car calculated using the steering angle. This would then be the input signal to our controller. Once again, the controller is a proportional type and 'sees' the same transfer function as in the previous point.

The idea behind the input signal to the controller, i.e. our reference signal, is to consider the minimum between the speed of 70km/h and the maximum allowed speed to have a lateral acceleration of 2 m/s^2. To obtain this value we used the formula

$$a_y = \frac{V_x^2}{R}$$

Where the acceleration is the limit given by the task normalised with the wheelbase value of 2.984, the longitudinal speed is what we wanted to calculate, the curve radius is calculated using linear geometry as

$$R = \sin\left(SWA * ratio\right)$$

Where SWA represents the steering angle and ratio represents the steering ratio, which in our case was calculated to be equal to π.

In this case, the value of the proportional controller was set to 1.3.

You can see a preview of the system implementation in Appendix no.11. While in Appendix no.12 and Appendix no.13 you can see graphs of the longitudinal speed and lateral acceleration of the vehicle.

In all these tasks, for tuning reasons, we decided to insert "Gain" blocks within Simulink in order to modify and attenuate the value of the signals, usually by changing them by a factor of 0.1 or 0.3. This was done to limit the peak values of the signals and in general to try to have smoother signals without sudden changes in value. We observed experimentally that this helped us obtain better results during simulation. In particular, the way the car reacted to accelerations and decelerations was much smoother.

4. Handling diagram

To complete this task, we simply had to follow step by step the steps described in the task text and the slides presented in the figure.

On a practical level, we first modified the Simulink file. In the block TME102_Task4.slx - CarMaker - VehicleControl - Subsystem (the same as in the previous task) we modified the signals relating to steering angle, steering angle velocity and steering angle acceleration by no longer using the value calculated by CarMaker, but an externally imposed signal. We decided to opt for a very trivial solution, namely to place a linear dependency between the simulation time and the steering angle. The more time passes, the greater the value of the steering angle will increase linearly. In order to calculate speed and acceleration, we could thus simply use two "Derivative" blocks as our input signal is a continuous time-dependent signal (it is actually the vector representing time itself). You can see a screenshot of the Simulink implementation in Appendix no.14

4.1. Electric SUV

For ease of calculation and less confusion, we decided to plot three handling together: that of the ICE machine, that of the electric machine and that of the modified machine to obtain the same characteristics as the ICE machine. The TME102_HandlingDiagram.m file is used to plot the graph.

You can observe the simulation result in Appendix no.15. As can be seen from the graph, especially for small values of -δ+l/R, the diagram for the ICE machine and that of the adjusted electrical machine match almost perfectly, while that of the initial electrical machine deviates greatly. This means that the adjustments made to the electrical machine are working correctly.

What we could understand from this task is that the electric car model has the same weight as the previous ICE car, but the position of its centre of gravity changes. This affects the performance of the car by decreasing the underseering and increasing the lateral acceleration limit as the ratio of the distance from the centre of gravity to the front and rear wheels changes.

In order to return to a case similar to that of the ICE car, it is necessary to modify the model of the anti roll bar. In particular, it is necessary to make it stiffer in the rear and softer in the front.

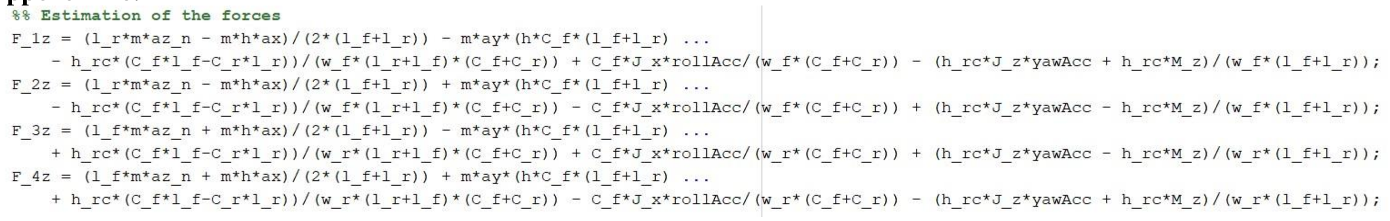
One could also modify the wheels, using larger wheels in the rear. However, we found this implementation more difficult and less accurate using CarMaker's wheel tuning.

Appendix no.16 shows a small table showing the values changed between one model of the car and another.

5. Appendix

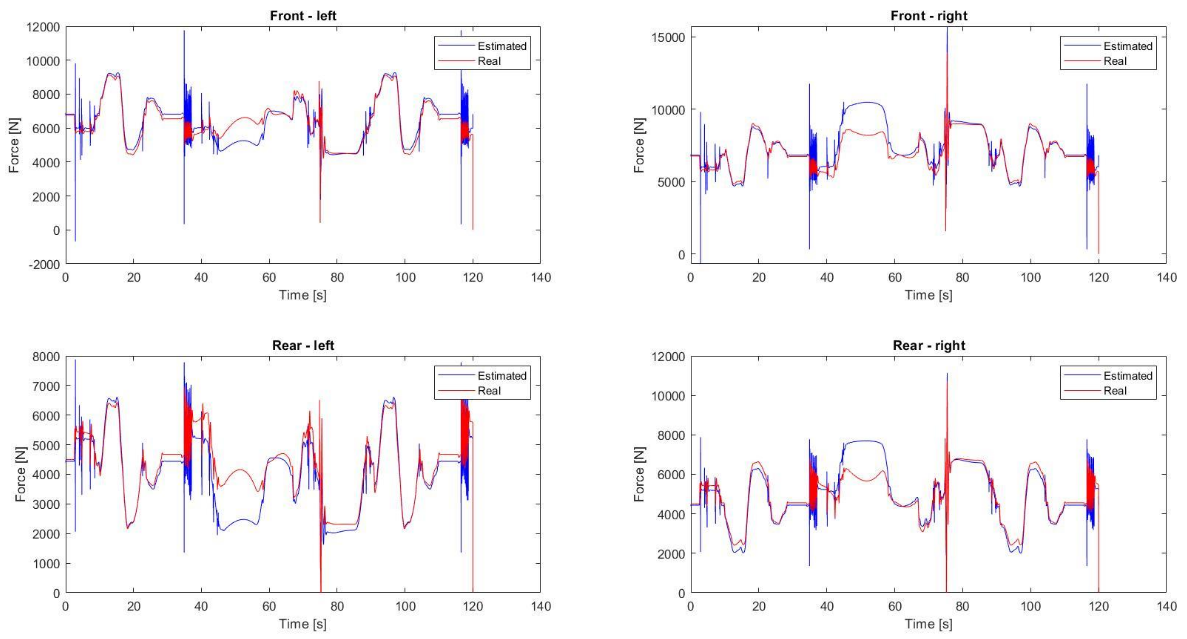
On the next page you can find all the annexes mentioned above.

Appendix no.1

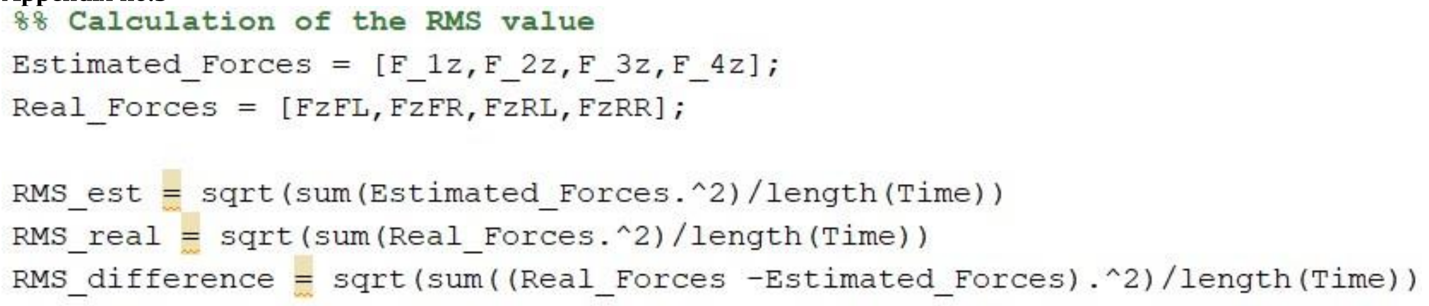


```
%% Estimation of the forces
F_1z = (l_r*m*az_n - m*h*ax)/(2*(l_f+l_r)) - m*ay*(h*C_f*(l_f+l_r) ...
    - h_rc*(C_f*l_f-C_r*l_r))/(w_f*(l_r+l_f)*(C_f+C_r)) + C_f*J_x*rollAcc/(w_f*(C_f+C_r)) - (h_rc*J_z*yawAcc + h_rc*M_z)/(w_f*(l_f+l_r));
F_2z = (l_r*m*az_n - m*h*ax)/(2*(l_f+l_r)) + m*ay*(h*C_f*(l_f+l_r) ...
    - h_rc*(C_f*l_f-C_r*l_r))/(w_f*(l_r+l_f)*(C_f+C_r)) - C_f*J_x*rollAcc/(w_f*(C_f+C_r)) + (h_rc*J_z*yawAcc - h_rc*M_z)/(w_f*(l_f+l_r));
F_3z = (l_f*m*az_n + m*h*ax)/(2*(l_f+l_r)) - m*ay*(h*C_f*(l_f+l_r) ...
    + h_rc*(C_f*l_f-C_r*l_r))/(w_r*(l_r+l_f)*(C_f+C_r)) + C_f*J_x*rollAcc/(w_r*(C_f+C_r)) + (h_rc*J_z*yawAcc - h_rc*M_z)/(w_r*(l_f+l_r));
F_4z = (l_f*m*az_n + m*h*ax)/(2*(l_f+l_r)) + m*ay*(h*C_f*(l_f+l_r) ...
    + h_rc*(C_f*l_f-C_r*l_r))/(w_r*(l_r+l_f)*(C_f+C_r)) - C_f*J_x*rollAcc/(w_r*(C_f+C_r)) - (h_rc*J_z*yawAcc + h_rc*M_z)/(w_r*(l_f+l_r));
```

Appendix no.2



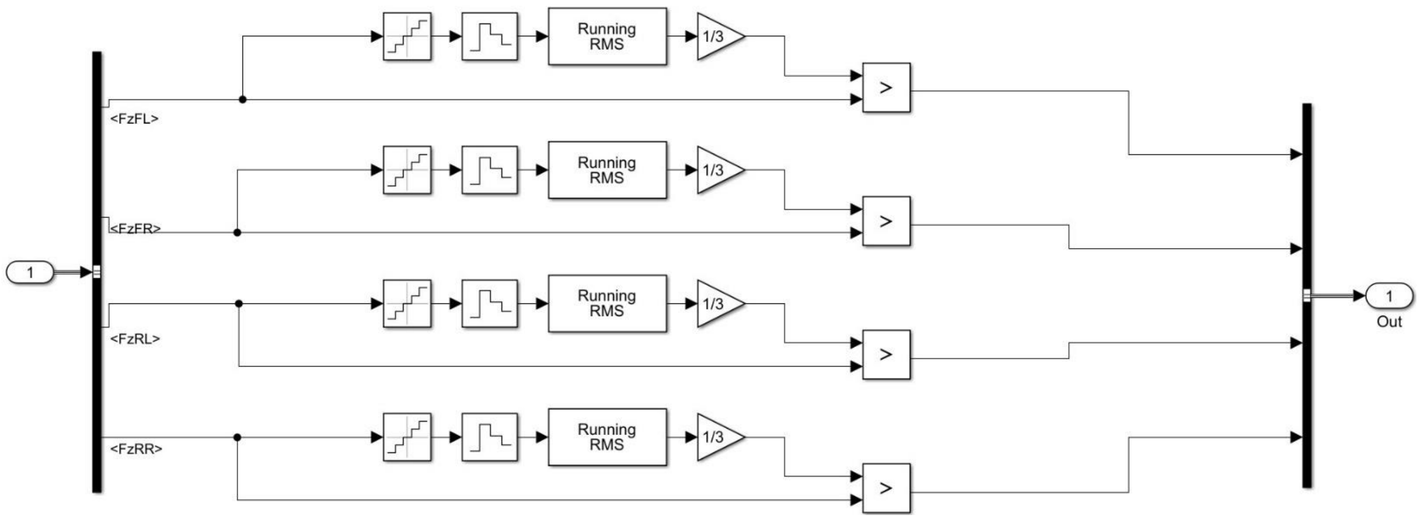
Appendix no.3



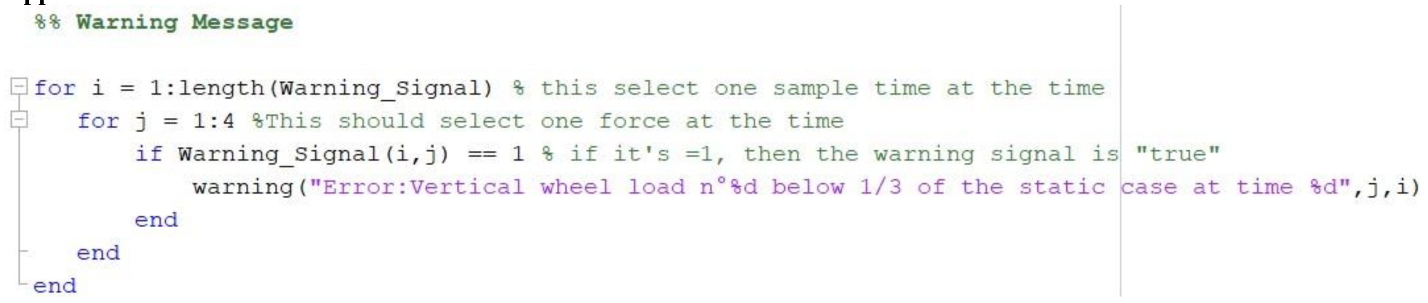
```
%% Calculation of the RMS value
Estimated_Forces = [F_1z,F_2z,F_3z,F_4z];
Real_Forces = [FzFL,FzFR,FzRL,FzRR];

RMS_est = sqrt(sum(Estimated_Forces.^2)/length(Time))
RMS_real = sqrt(sum(Real_Forces.^2)/length(Time))
RMS_difference = sqrt(sum((Real_Forces -Estimated_Forces).^2)/length(Time))
```

Appendix no.4



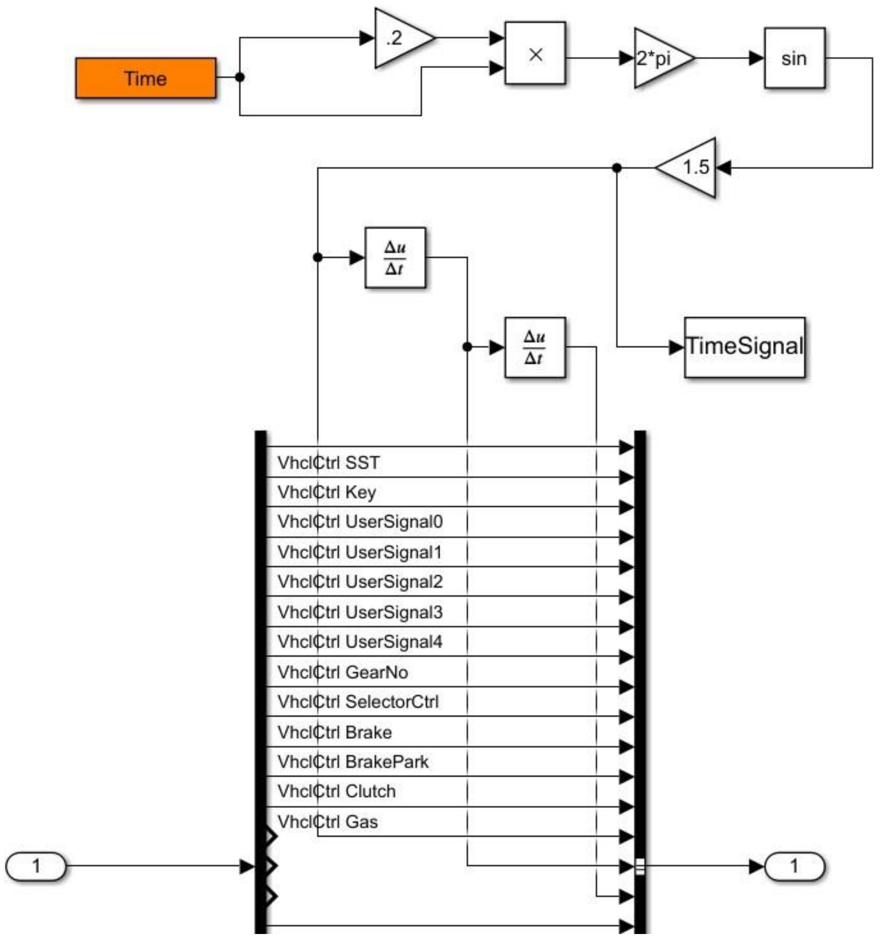
Appendix no.5



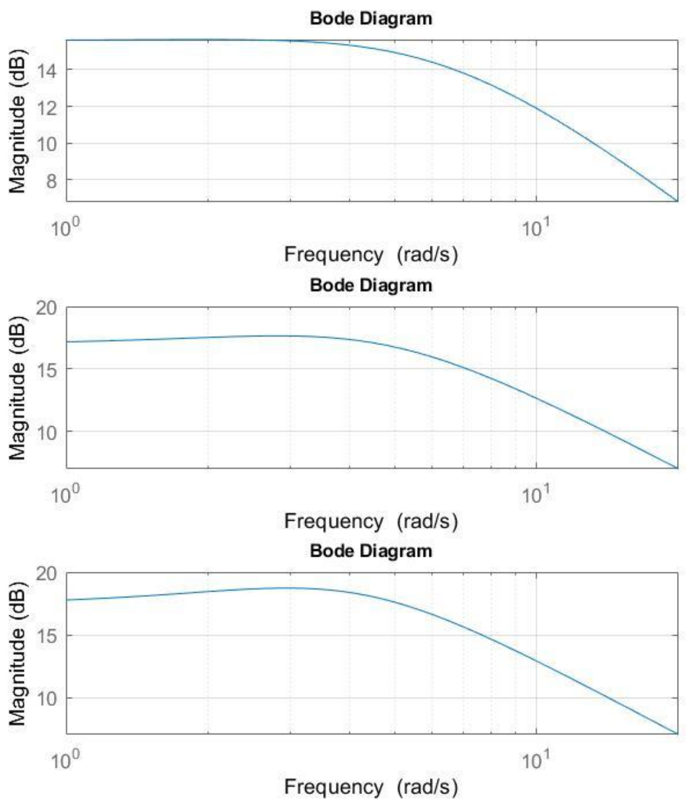
```
%% Warning Message

for i = 1:length(Warning_Signal) % this select one sample time at the time
    for j = 1:4 %This should select one force at the time
        if Warning_Signal(i,j) == 1 % if it's =1, then the warning signal is "true"
            warning("Error:Vertical wheel load n°%d below 1/3 of the static case at time %d",j,i)
        end
    end
end
```

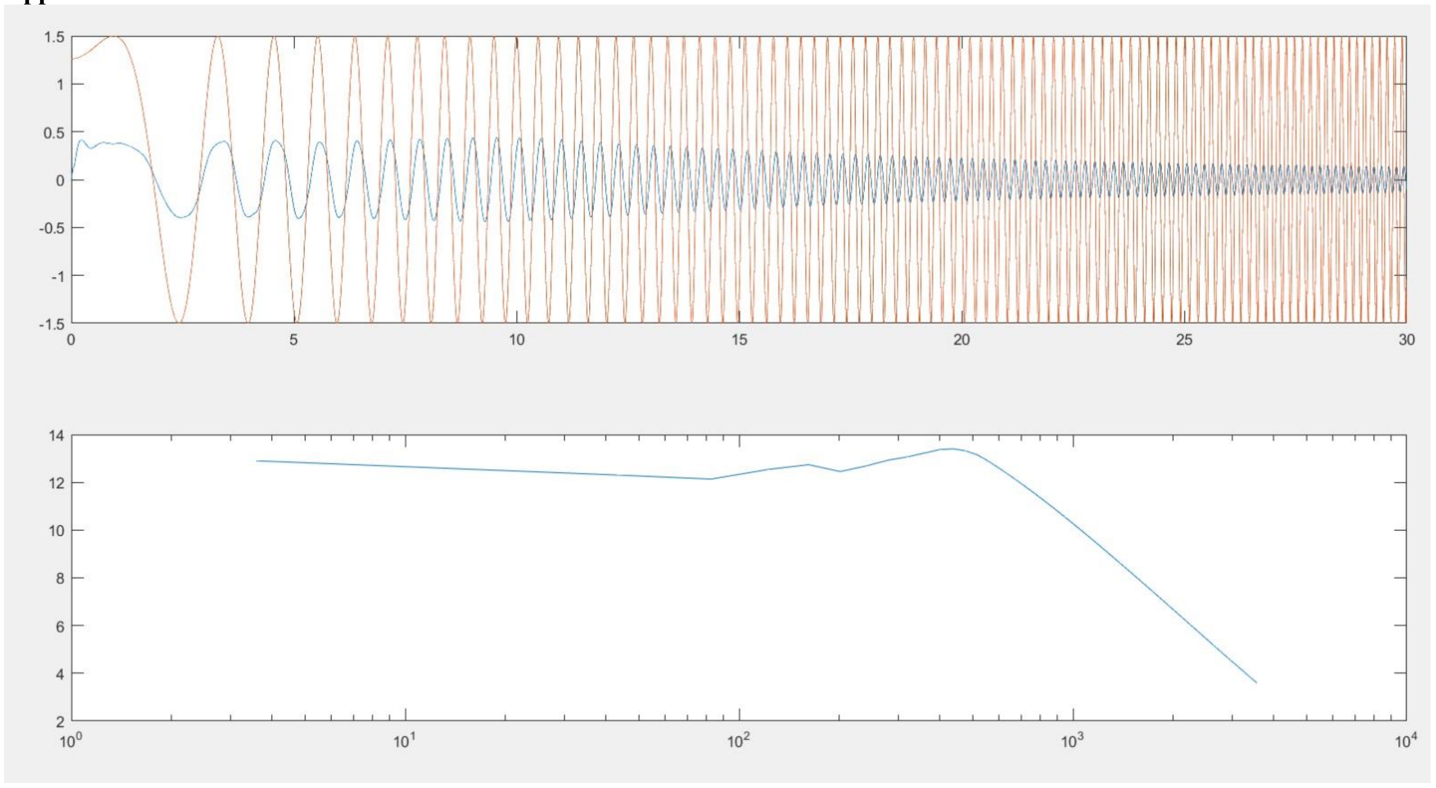
Appendix no.6



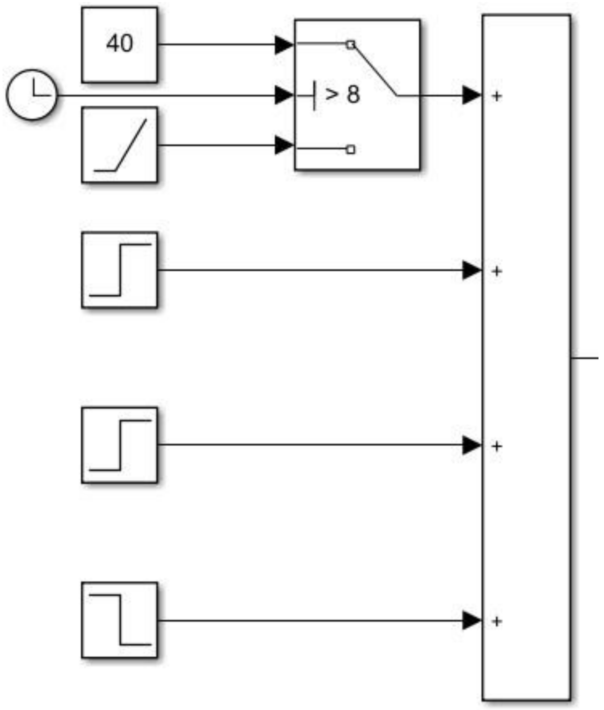
Appendix no.7



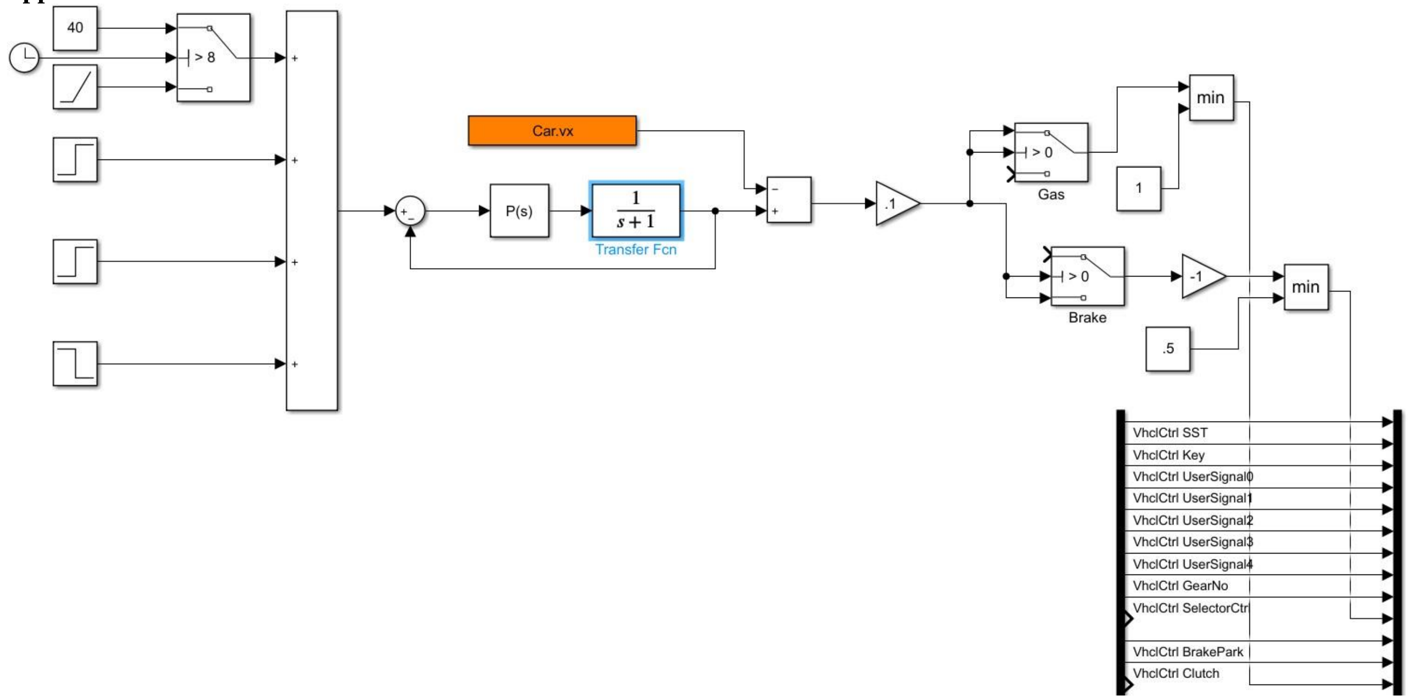
Appendix no.8



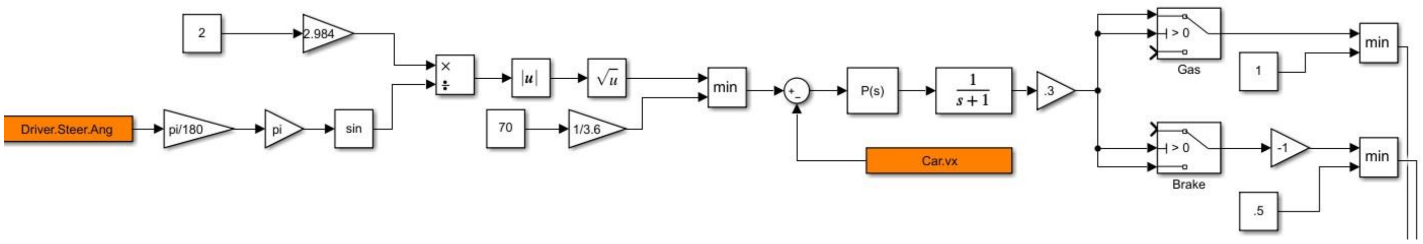
Appendix no.9



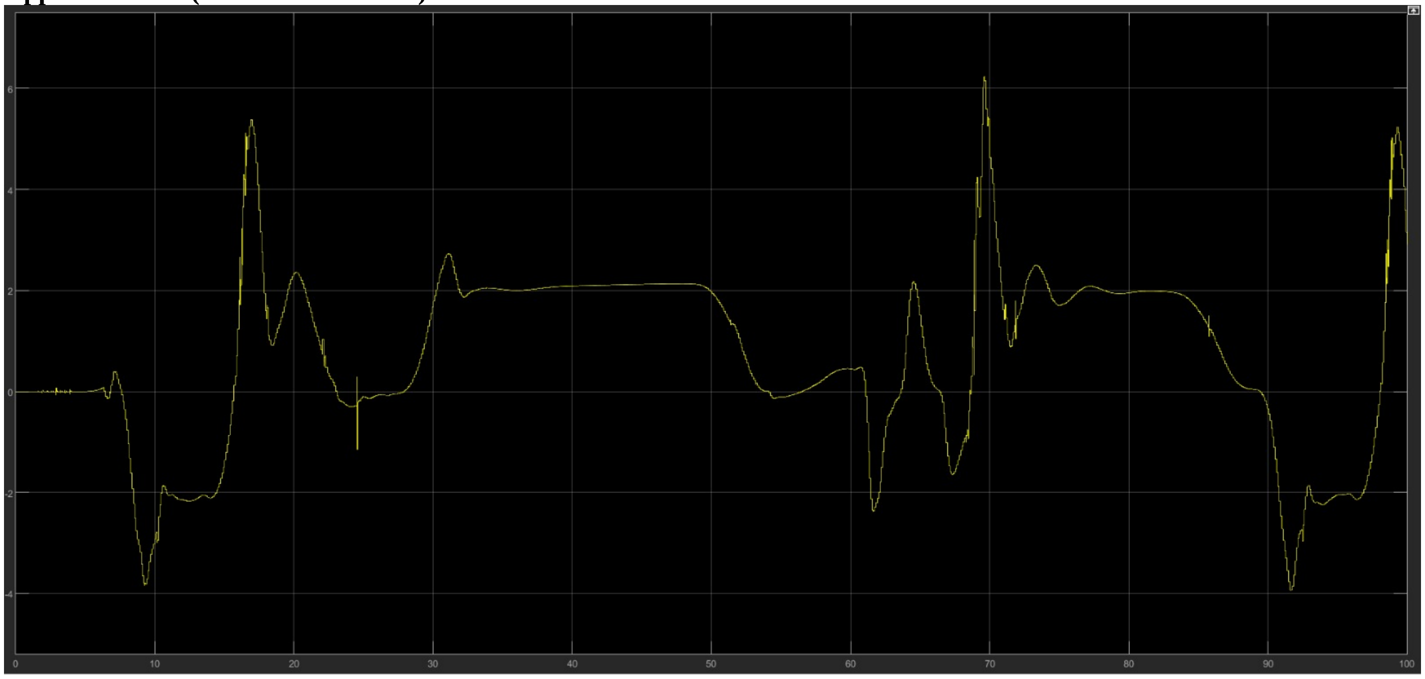
Appendix no.10



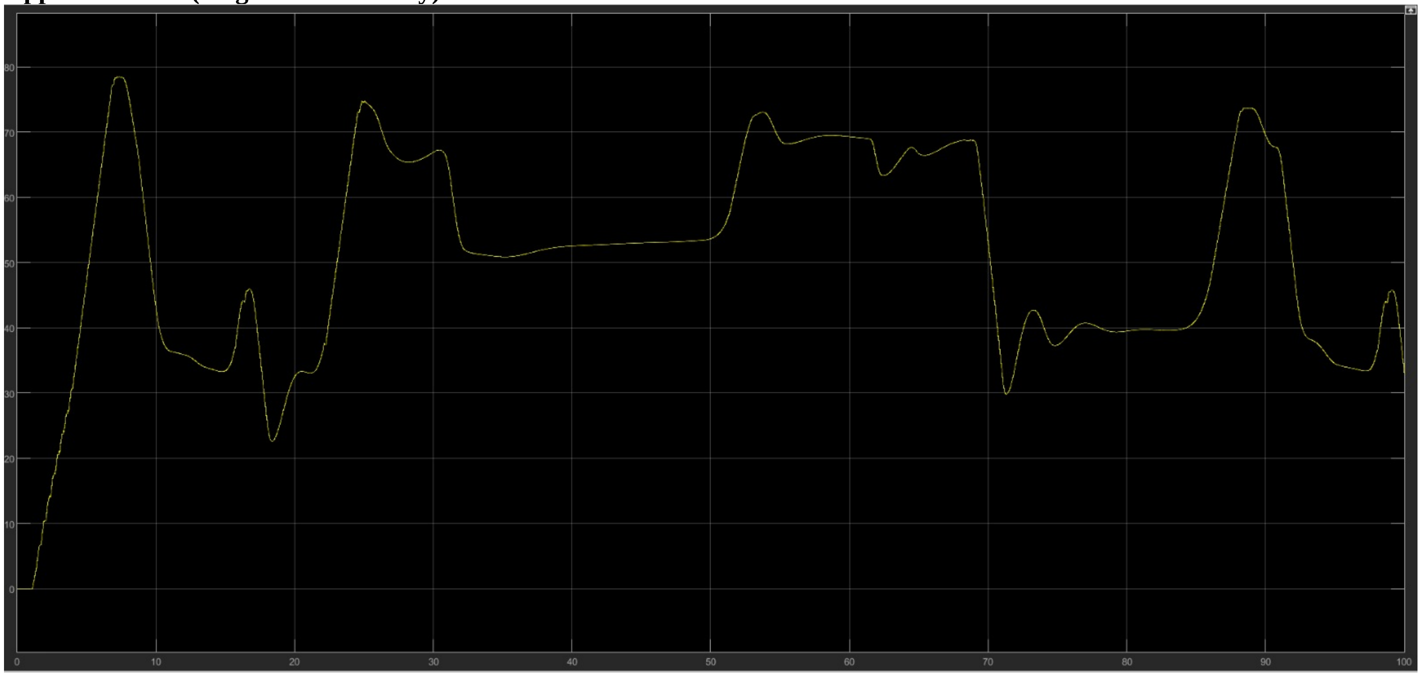
Appendix no.11



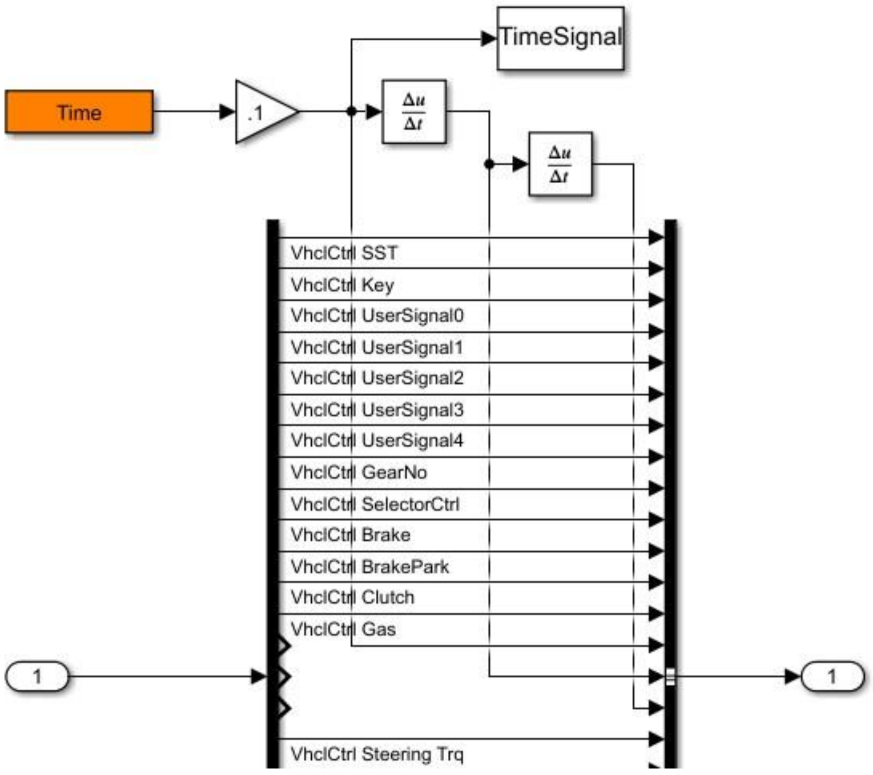
Appendix no.12 (lateral acceleration)



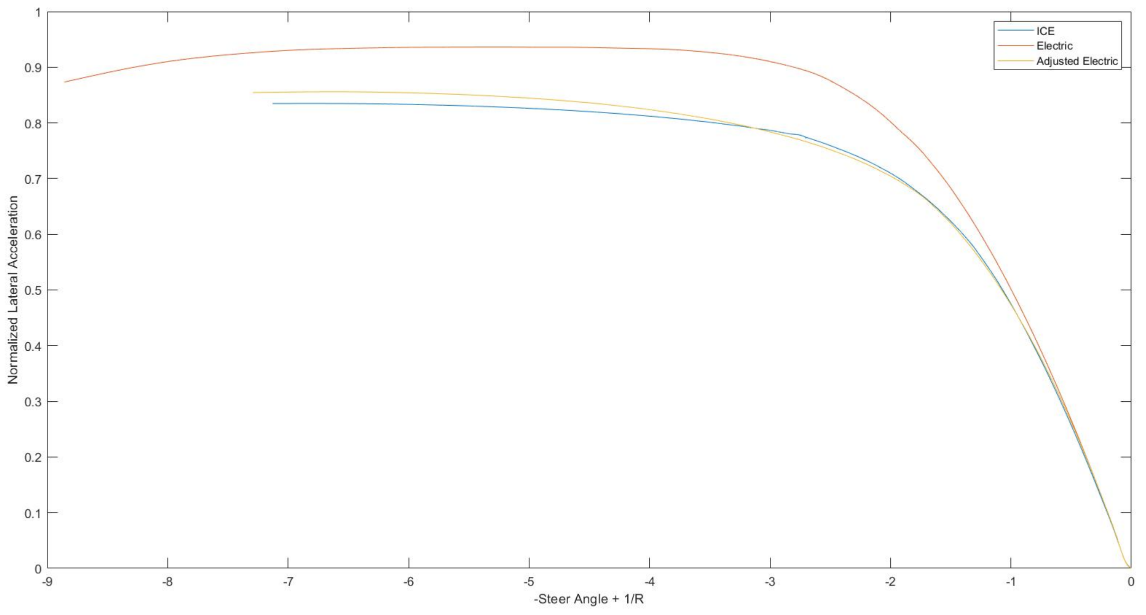
Appendix no.13 (longitudinal velocity)



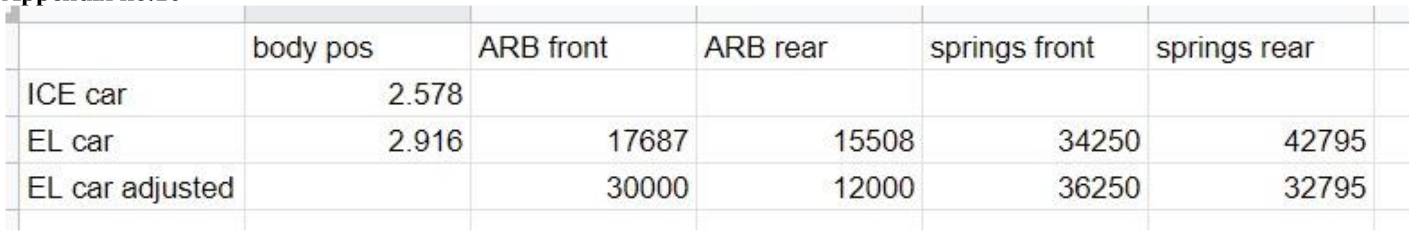
Appendix no.14



Appendix no.15



Appendix no.16



	body pos	ARB front	ARB rear	springs front	springs rear
ICE car	2.578				
EL car	2.916	17687	15508	34250	42795
EL car adjusted		30000	12000	36250	32795