Homework 8.2: Vehicle Dynamics and Stability

Author: Anton Østergaard Thorsen, 202004932
Date: 21/12/2023

1998 Honda Civic

Most vehicle parameters are found from https://www.eng.auburn.edu/~dmbevly/mech4420/vehicle_params.pdf (see Vehicle No. 452)

Parameter	**Value**
Mass, m	$m = 1143\,\text{kg}$
Length of car, L	$L = 2.621\,\text{m}$
Distance to front wheels from CG, a	$a = 1.038\,\text{m}$
Yaw moment of inertia, I_{zz}	$I_{zz} = 1785\,\text{kg}\cdot\text{m}^2$
Cornering stiffnesses, $C_{\alpha f}$ & $C_{\alpha r}$	$C_{\alpha f} = C_{\alpha r} = 50990.278\,\frac{\text{N}}{\text{rad}}$
Proportioning of front to rear lateral load transfer, $\overline{p}$	$\overline{p} = 0.5$
Height to center of mass, H	$H = 0.513\,\text{m}$
Width of car, T	$T = 1.695\,\text{m}$
Roll gain, $\overline{\phi}$	$\overline{\phi} = -4\,\frac{\text{deg}}{\text{g}}$
Roll Frequency	$freq\_R = 1.5\,\text{Hz}$
Pitch Frequency	$freq\_P = 1\,\text{Hz}$
Brake limit (from HW6)	$BLIMIT = 1787.88\,\text{N}$
Braking ratio (from HW6), Q	$Q = 2.48$
Pitch gain, $\overline{\theta}$	$\overline{\theta} = -0.1\,\frac{\text{rad}}{\text{g}}$

Strategy

The strategy of this task was to first and foremost decide on a start location and initial orientation angle.
I choose $X(0) = -20\,\text{m}$ somewhat arbitrarily. The initial orientation angle is found using $X(0)$ and the location of the first target $(0\,\text{m}, 3\,\text{m})$:

$$\psi(0) = invTan\left(\frac{-3\,\text{m}}{-20\,\text{m}}\right) = 8.531\ \text{deg}$$

Next up I made some assumptions:

- The car drives in a straight path from the starting point to the first target with an initial velocity, u_i.
- At the first target it initiates a step steer, δ_f, that will guide the car all the way to the final target.
- At some point, X_{brake}, between the first target and the final target the car will start braking at some deceleration, *decel*.

Next up I created a brute-force algorithm that finds the best solution amongst:

- 10 values between 15 and 25 m/s of initial velocity
- 10 values between 0.2 and 0.6 g's of deceleration
- 10 values of step steer values between -1.5 and 2 degrees.
- 10 values between 10 and 40 meters of the location of applying the brakes X_{brake}

Which results in 10000 sets of parameters (which resulted in rather large computation time)

This gave me a solution that took 4.51 seconds to reach the final target with the parameters:

$$u_i = 17.2222\,\frac{\text{m}}{\text{s}},\quad decel = 0.4222g,\quad \delta_f = 1.6667\,\text{deg},\quad X_{brake} = 26.6667\,\text{m}$$

I did another run to narrow down the values around the optimized solution:

- 10 values between 16 and 20 m/s of initial velocity
- 10 values between 0.4 and 0.45 g's of deceleration
- 10 values of step steer values between -1.5 and 2 degrees.
- 10 values between 24 and 28 meters of the location of applying the brakes X_{brake}

Which gave me a solution time of 4.39 seconds with the parameters I decided to settle on:

$$u_i = 17.7778\,\frac{\text{m}}{\text{s}},\quad decel = 0.4444\,g,\quad \delta_f = 1.7222\,\text{deg},\quad X_{brake} = 26.2222\,\text{m}$$

Now the way the algorithm determines whether a set of parameters results in a viable solution is by checking the following list of conditions:

1. Is the final velocity below 10 m/s?
2. Are there no lockups at any point in time on any of the wheels?
3. Is the Y position of the front left wheel positive, the Y position of the rear right wheel negative, and the Y position of the center of gravity within the range $[-0.3\,\text{m}, 0.3\,\text{m}]$

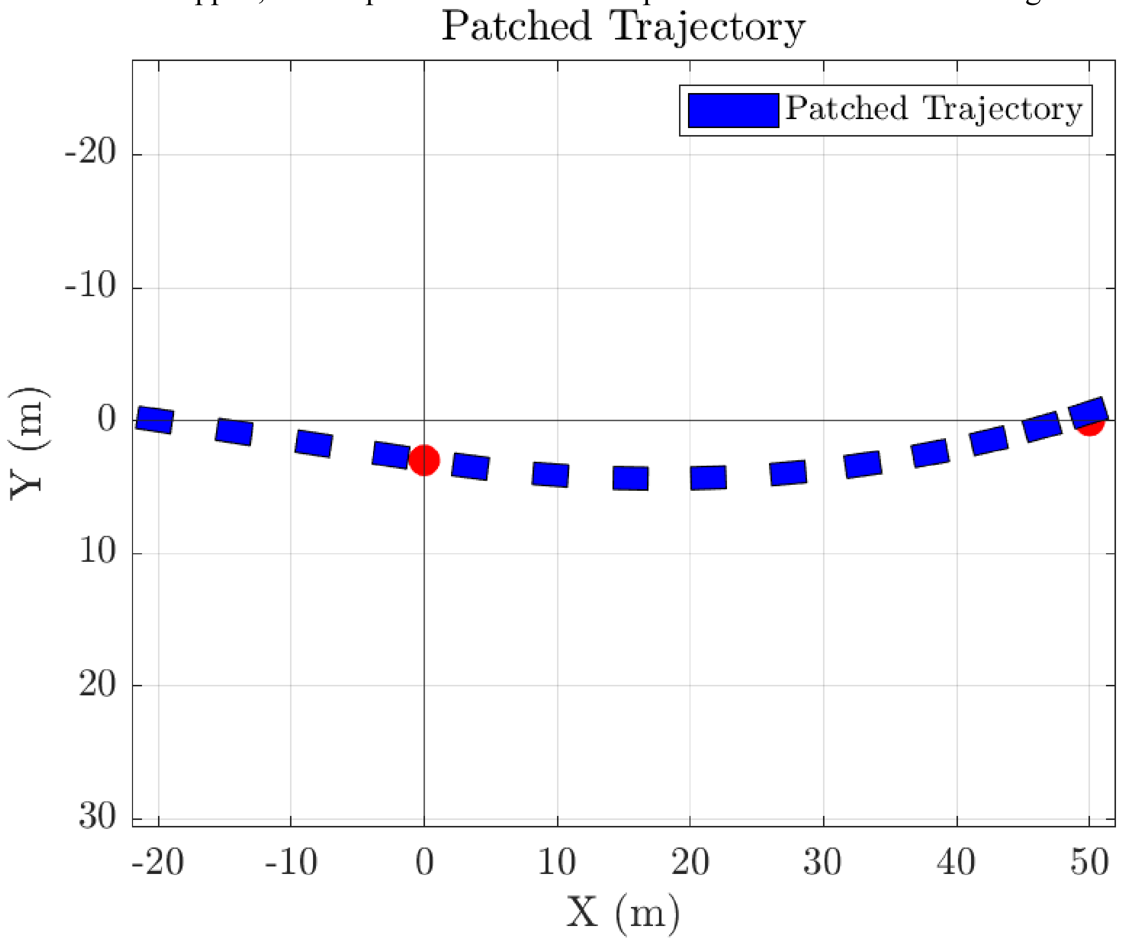
I found that condition nr. 3 would ensure that the final destination is reached.
If all these conditions where satisfied and the solution time was quicker than the solution time of any of the other sets of parameters, the new set of parameters would be stored as the current fastest solution.

Additionally it is worth noting that the MATLAB function fric_coeff (see all codes in the bottom pages) was also necessary to know the *X*-position of each wheel in order of determining whether a wheel had crossed into X > 0 which would result in a change of friction coefficient.

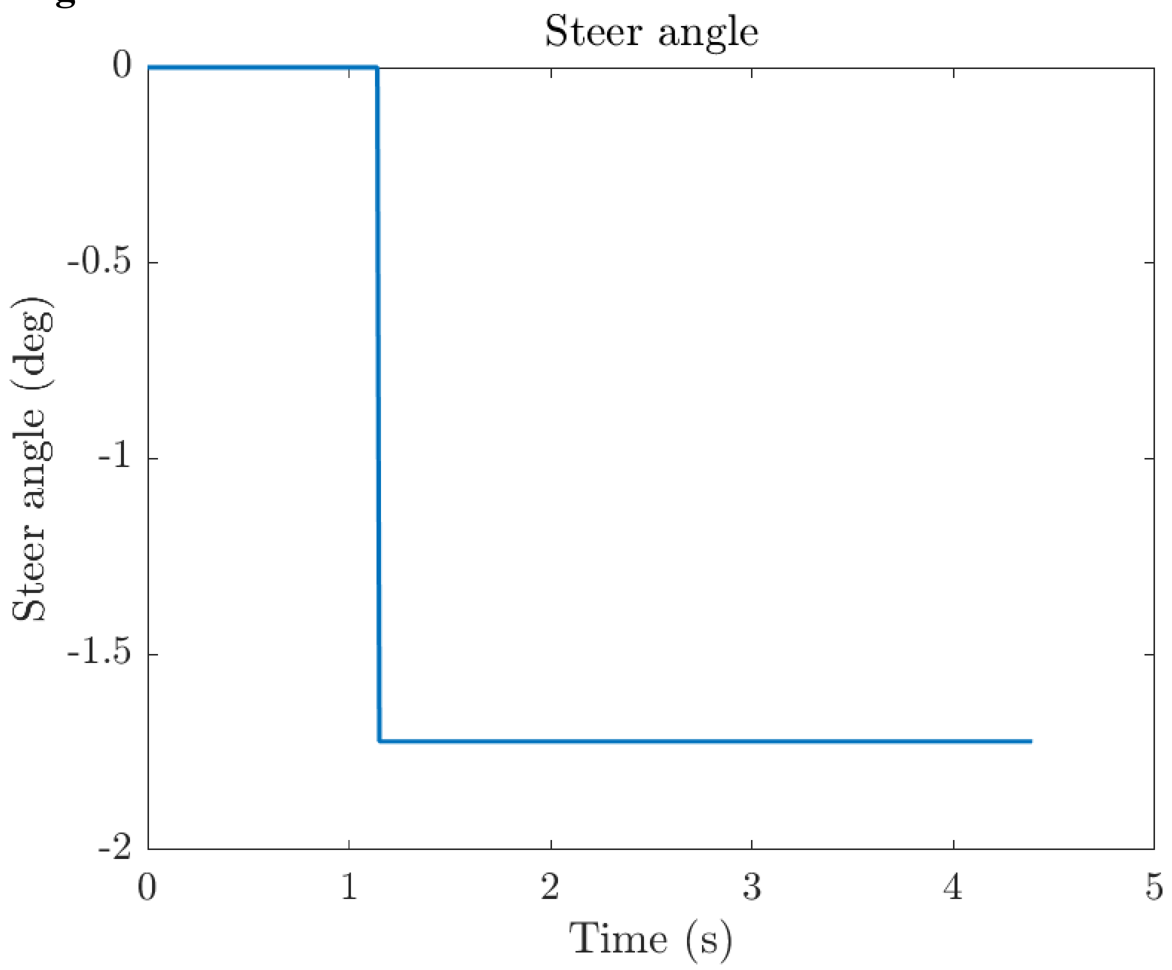
Trajectory

From start $(X_i, Y_i) = (-20\text{m}, 0\text{m})$ to finish $(X_f, Y_f) = (50\text{m}, 0\text{m})$ it takes 4.39 seconds with the optimized solution.

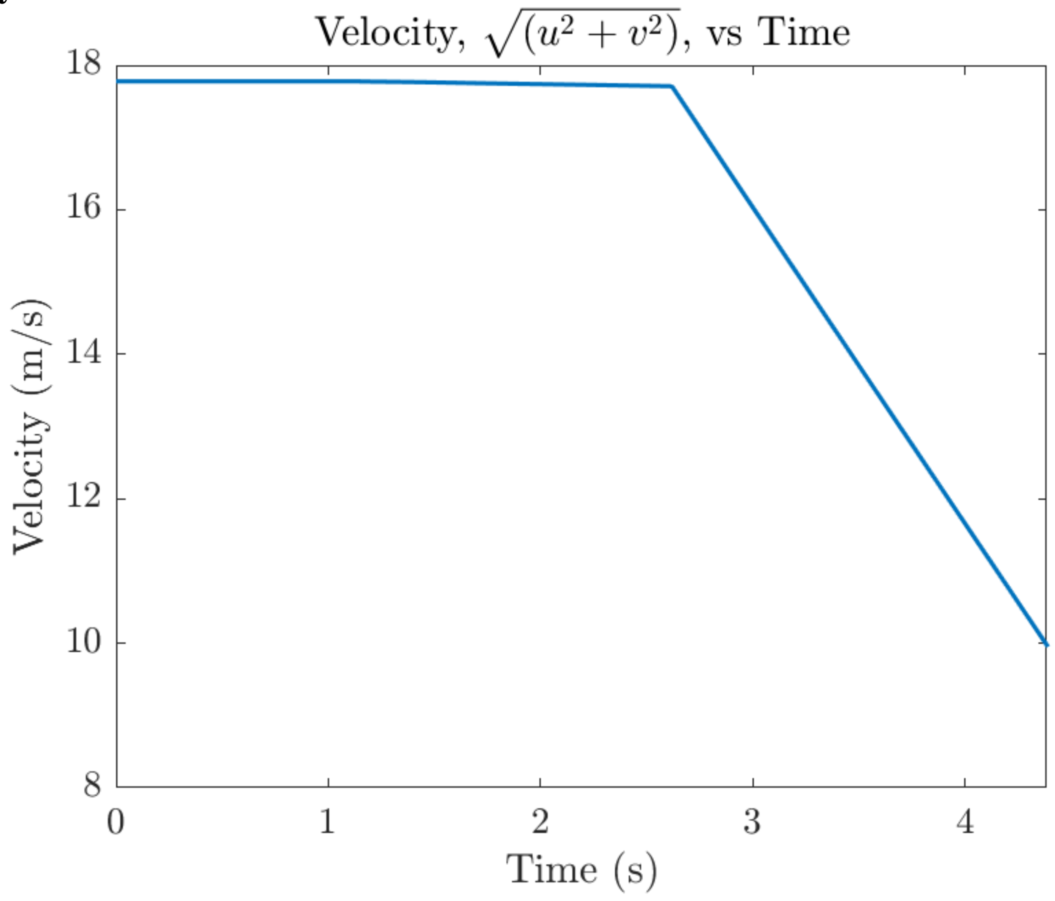
Note that the Y-axis flipped, so that positive values correspond with what looks like a right hand turn.



Steering:

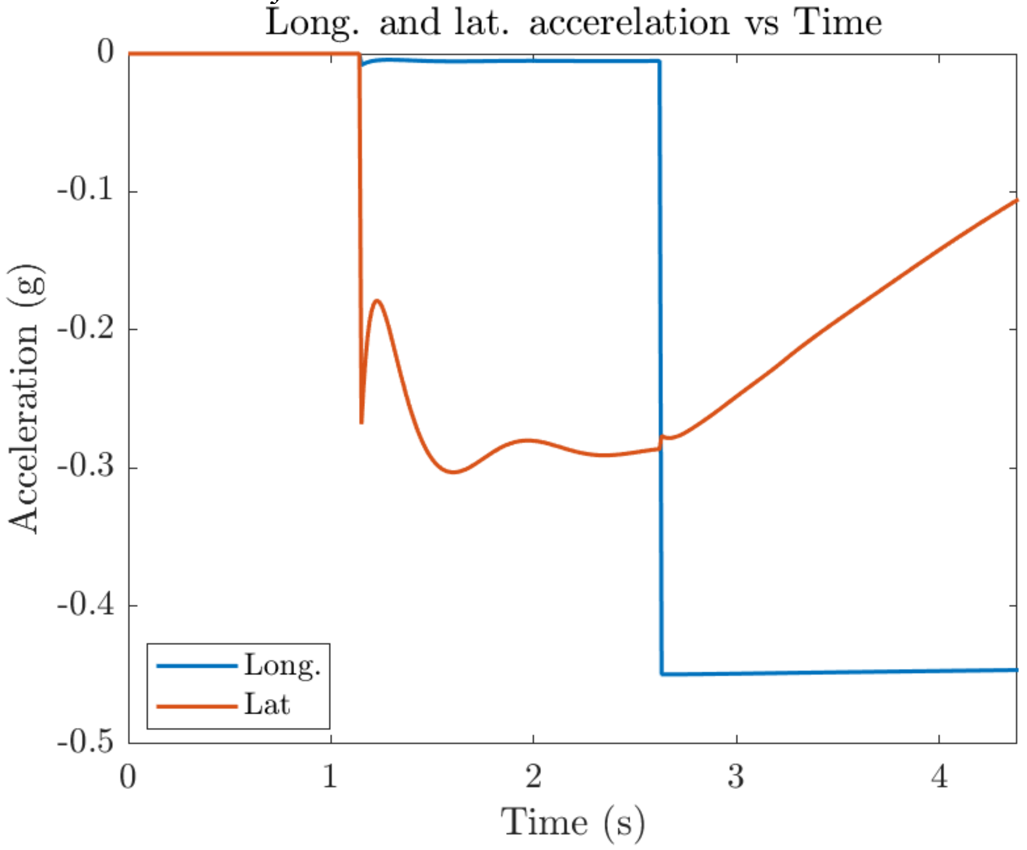


Velocity:

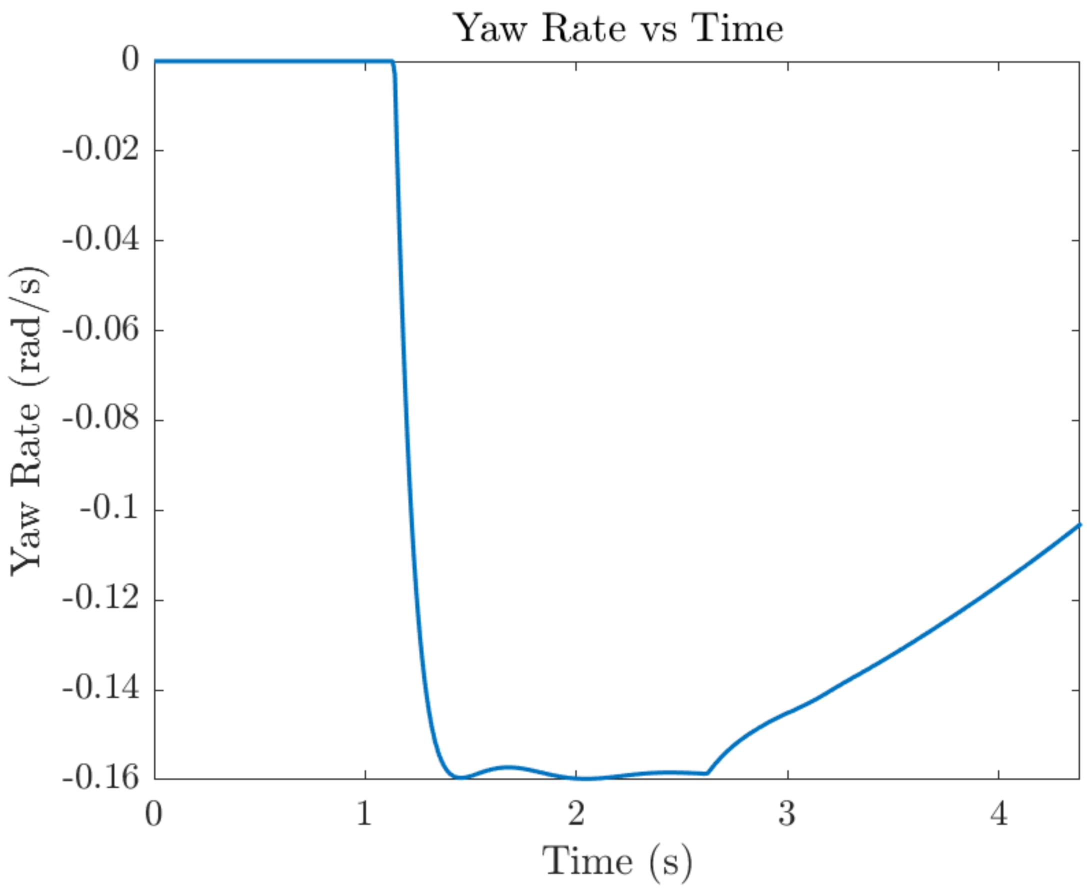


Velocity, $\sqrt{(u^2 + v^2)}$, vs Time
Velocity (m/s)
Time (s)
18
16
14
12
10
8
0
1
2
3
4

Superimposed a_x and a_y in g's

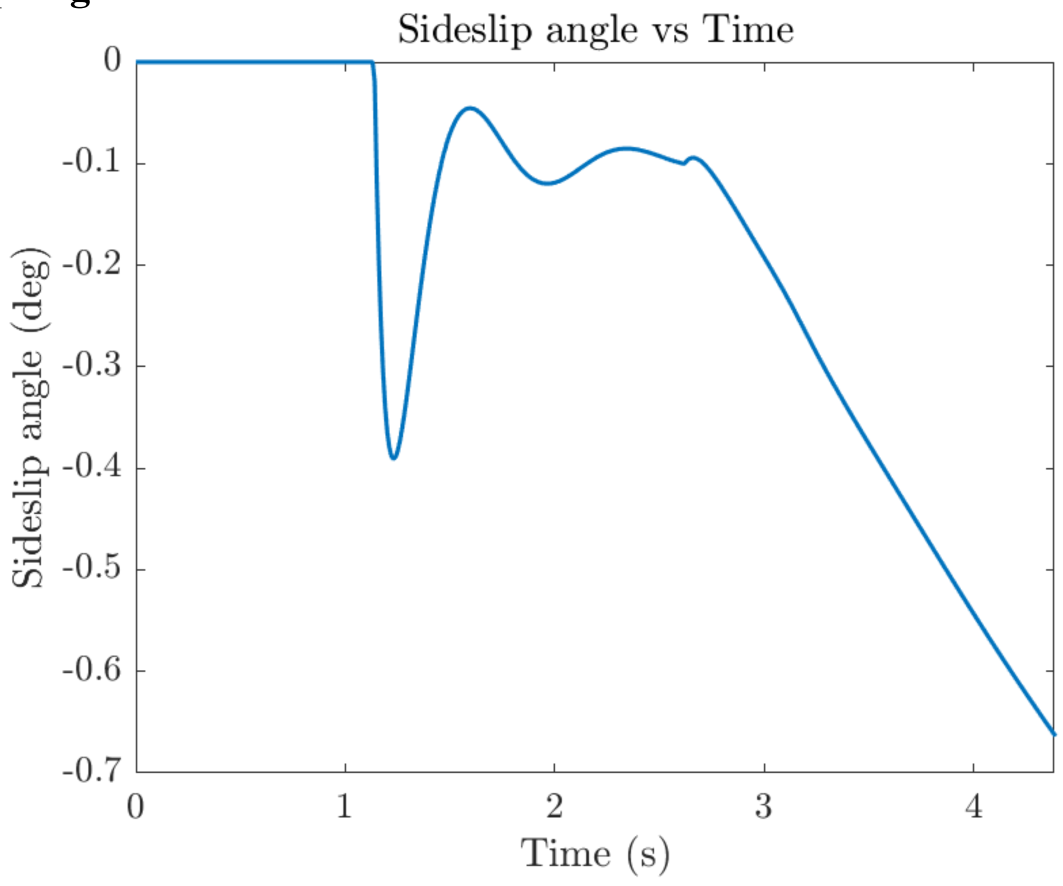


Yaw rate:

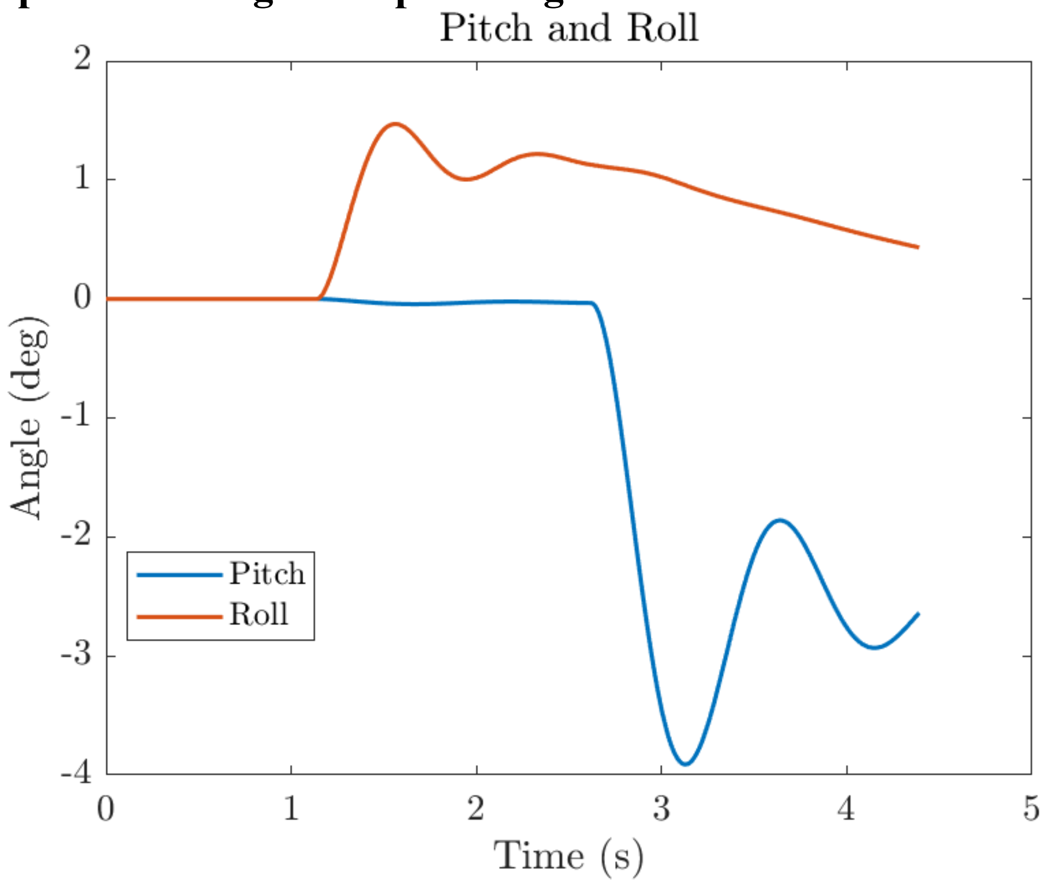


Yaw Rate vs Time
Yaw Rate (rad/s)
Time (s)
0
-0.02
-0.04
-0.06
-0.08
-0.1
-0.12
-0.14
-0.16
0
1
2
3
4

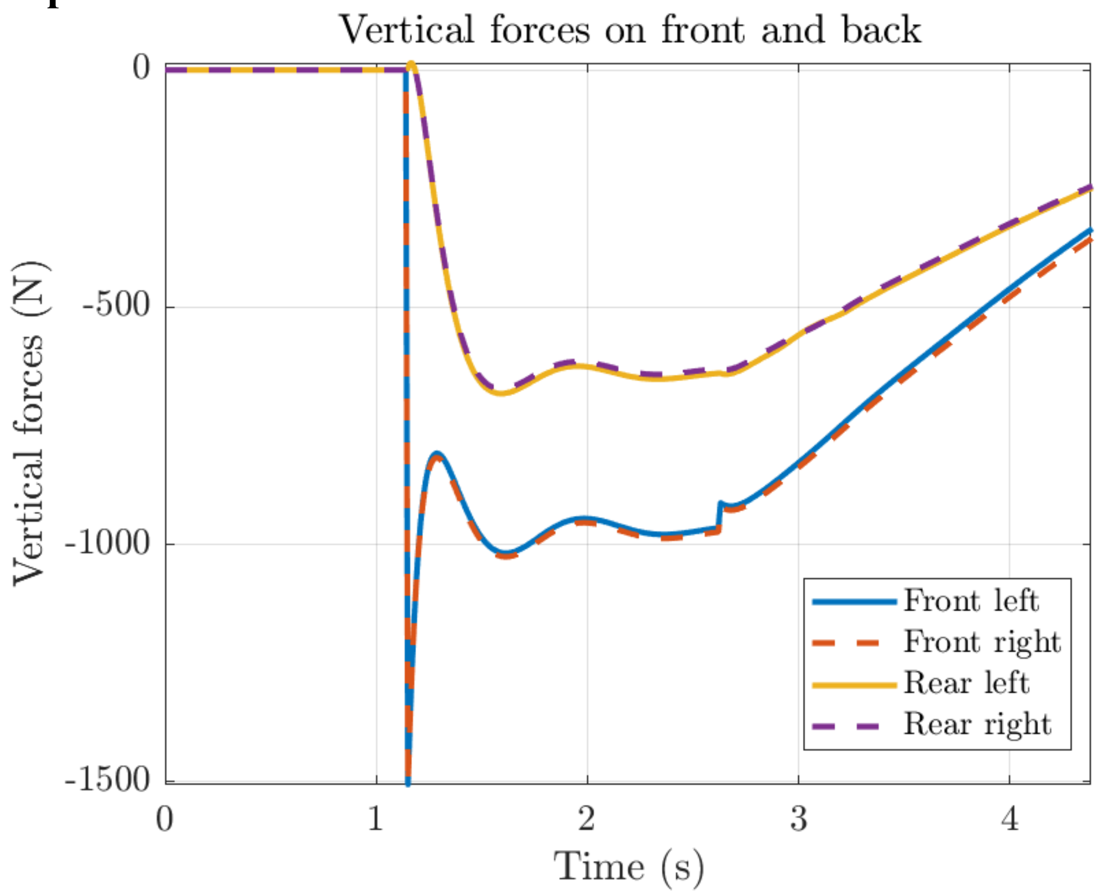
Sideslip angle:



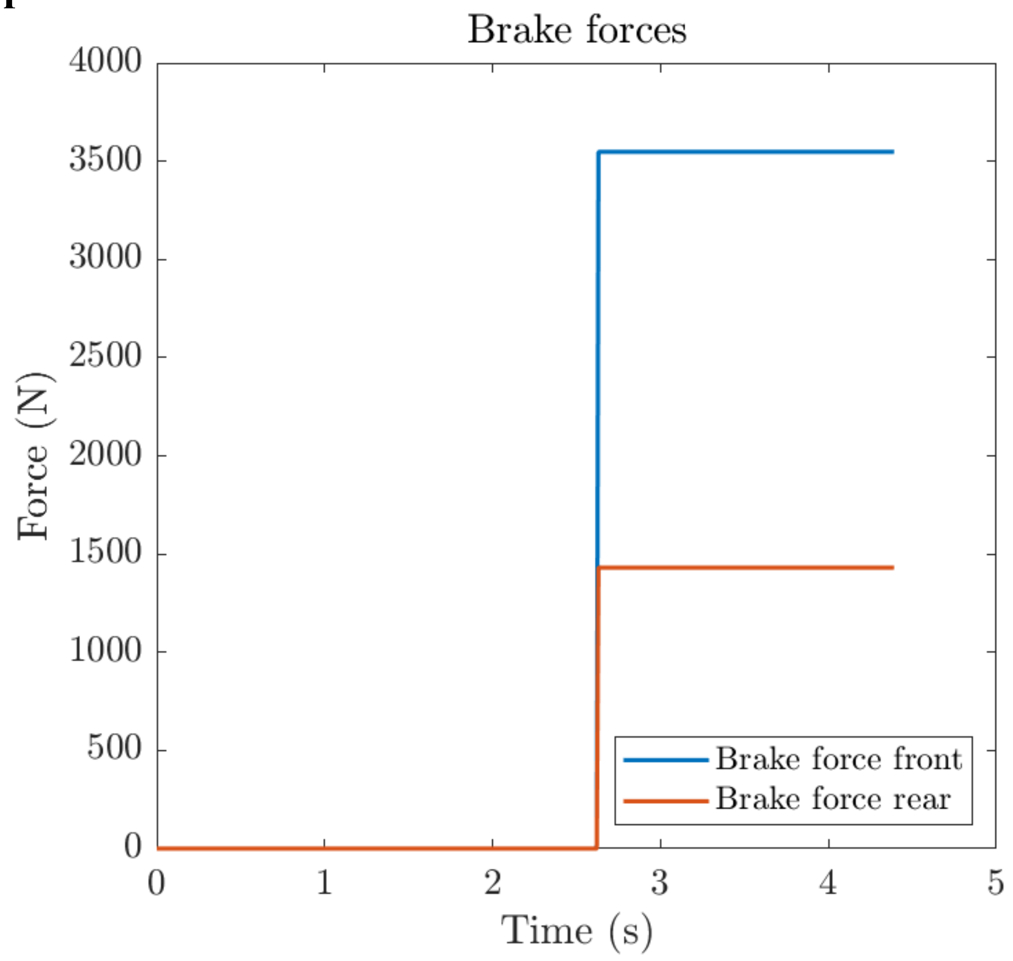
Superimposed roll angle and pitch angle:



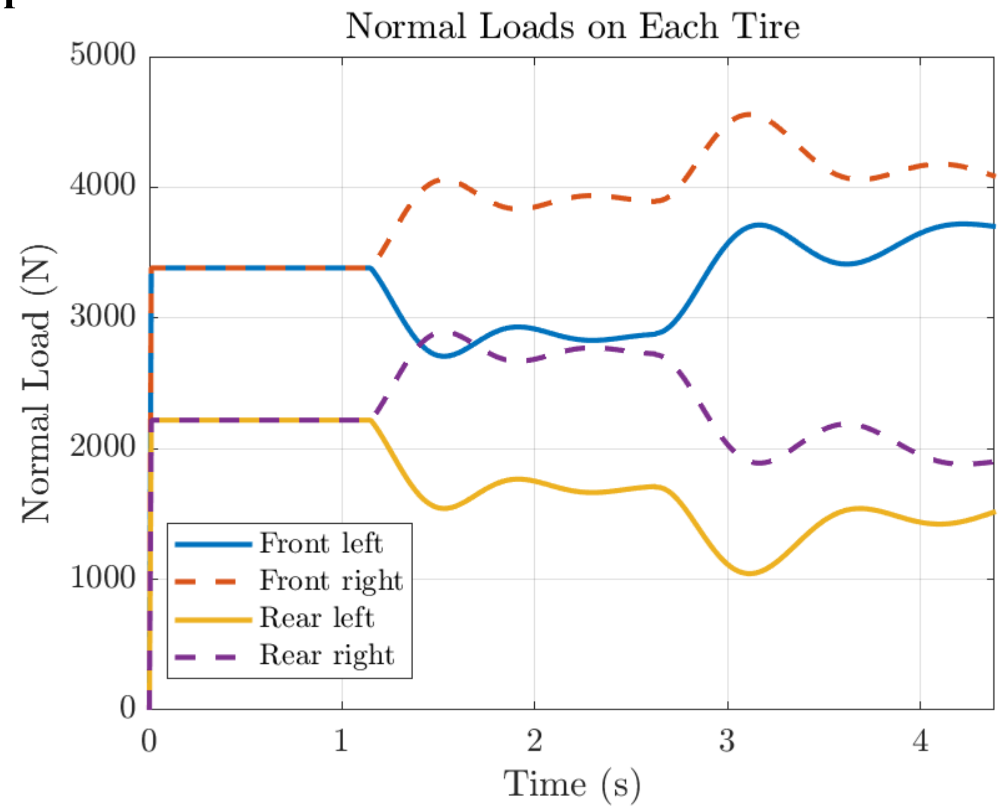
Superimposed vertical forces:



Superimposed brake forces:

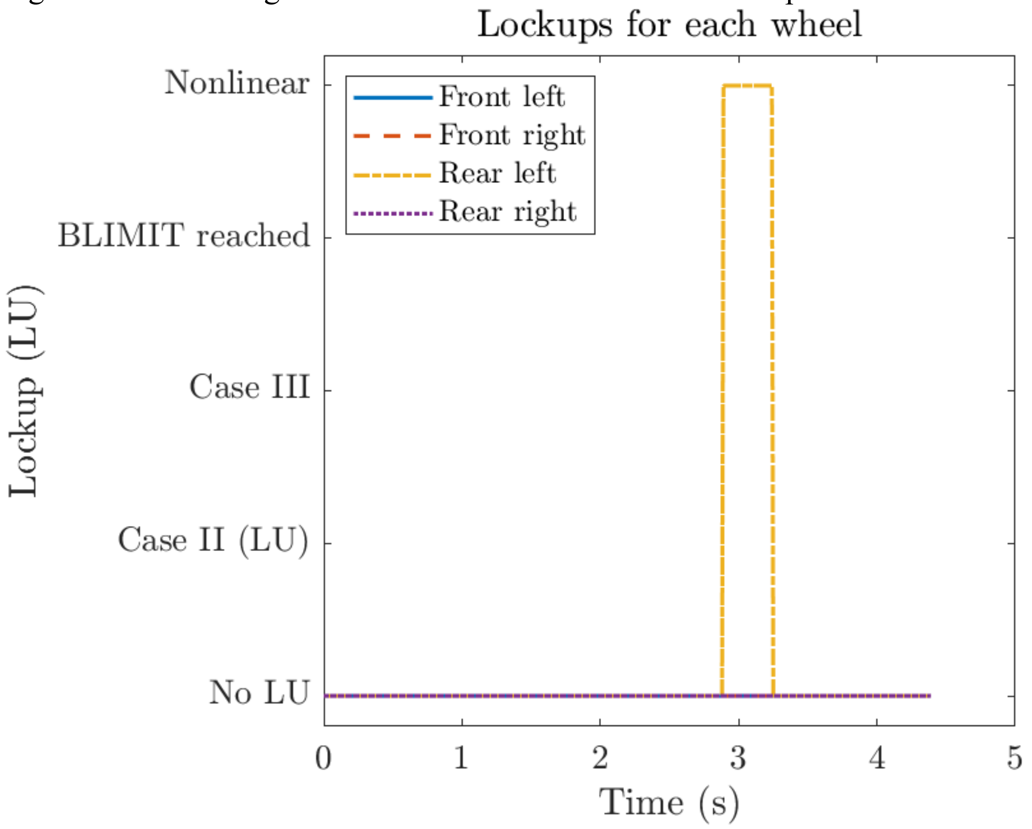


Superimposed N's:



Extra plot:

Plot describing the inner workings of the code. It illustrates that no lockups occur:



Derivative function, car_modelNL, and the functions it calls in order of occurence

```
function [Xdot, a_y, a_x, N_all, Fx_all, BF, delta_f, LU, Fy_all] = car_modelNL(X↙
t, car) %n is the iteration number
    tspan = car.tspan; m = car.m; a = car.a; L = car.L; b = car.b; Izz = car.Izz↙
C_f = car.C_f; C_r = car.C_r; g = car.g;
    delta_f = car.delta_f; p_bar = car.p_bar; H2 = car.H2; W = car.W; T = car.T↙
phi_bar = car.phi_bar; K = car.K; Ixx = car.Ixx; c_roll = car.c_roll;
    delta_r = car.delta_r;BLIMIT = car.BLIMIT; Q_brak = car.Q_brak; theta_ss = car↙
theta_ss; Iyy = car.Iyy; K_bar = car.K_bar; H2 = car.H2; c_pitch = car.c_pitch;
    K_dbar = car.K_dbar; X_brake = car.X_brake; decel = car.decel;

    r = X(1);
    v = X(2);
    glob_X = X(3);
    glob_Y = X(4);
    psi = X(5);
    phi = X(6);
    phi_dot = X(7);
    u = X(8);            % New variables for longitudinal speed, pitch and pitch↙
rate
    theta = X(9);
    theta_dot = X(10);

    Q = delta_r/phi_bar;    % delta_r is 0, so this is 0

    if glob_X < 0
        delta_f = 0;
        Q = 0;
    else
        delta_f = delta_f;
        Q = Q;
    end

    %Static loads
    N_f = (W * b) / L;
    N_r = (W * a) / L;

    %If we are not braking
    BF_f = 0; BF_r = 0;  DeltaN_pitch = 0;%When not braking there is no brake↙
force, and no pitch.

    % Normal loads - the only difference in normal loads w/o breaking are left-to↙
right because of roll
    DeltaN_roll = 2 / T * (-K * phi - c_roll * phi_dot);
    DeltaN_f_roll = p_bar * DeltaN_roll;
    DeltaN_r_roll = (1 - p_bar) * DeltaN_roll;

    %If we are braking:
    if glob_X > X_brake %Break when ~steady state, at t > t_brake
        %decel = 0.39;
        [~, BF_f, BF_r, ~, ~, ~, ~, ~] = brake_no_trailer(car, decel); %Calculate↙
forces when braking
        DeltaN_pitch = K_dbar*theta;
        %BF_f = 10^6; %To lock up front wheels
        %BF_r = 10^6; %To lock up rear wheels
    end

    BF = [BF_f, BF_r];
    % Brake forces
    SUM_BF = BF_f + BF_r; %[UNUSED]

    %Normal loads
    N_fl = N_f / 2 + DeltaN_f_roll / 2 + DeltaN_pitch / 2;
    N_fr = N_f / 2 - DeltaN_f_roll / 2 + DeltaN_pitch / 2;
    N_rl = N_r / 2 + DeltaN_r_roll / 2 - DeltaN_pitch / 2;
    N_rr = N_r / 2 - DeltaN_r_roll / 2 - DeltaN_pitch / 2;
    N_all = [N_fl, N_fr, N_rl, N_rr];

    % Slip angles and lateral forces
    alpha_fL = atan2( v + a * r, u + T*r/2) - delta_f;
    alpha_fR = atan2( v + a * r, u - T*r/2) - delta_f;
    alpha_rL = atan2( v - b * r, u + T*r/2) - Q*phi;
    alpha_rR = atan2( v - b * r, u - T*r/2) - Q*phi;

    [mu_fL, mu_fR, mu_rL, mu_rR, ~, ~] = fric_coeff(glob_X, psi, car, glob_Y);

    [Fx_FL_tire, Fy_FL_tire, LU(1)] = NLTire(C_f, alpha_fL, mu_fL, N_fl, BF_f/2);↙
%Tiremodel is for 1 wheel, so divide BF with 2
    [Fx_FR_tire, Fy_FR_tire, LU(2)] = NLTire(C_f, alpha_fR, mu_fR, N_fr, BF_f/2);
    [Fx_RL_tire, Fy_RL_tire, LU(3)] = NLTire(C_r, alpha_rL, mu_rL, N_rl, BF_r/2↙
BLIMIT);
    [Fx_RR_tire, Fy_RR_tire, LU(4)] = NLTire(C_r, alpha_rR, mu_rR, N_rr, BF_r/2↙
BLIMIT);

    Fx_FL = Fx_FL_tire*cos(delta_f) - Fy_FL_tire*sin(delta_f);
    Fy_FL = Fx_FL_tire*sin(delta_f) + Fy_FL_tire*cos(delta_f);
    Fx_FR = Fx_FR_tire*cos(delta_f) - Fy_FR_tire*sin(delta_f);
    Fy_FR = Fx_FR_tire*sin(delta_f) + Fy_FR_tire*cos(delta_f);
    Fx_RL = Fx_RL_tire;
    Fy_RL = Fy_RL_tire;
    Fx_RR = Fx_RR_tire;
    Fy_RR = Fy_RR_tire;
    Fx_all = [Fx_FL, Fx_FR, Fx_RL, Fx_RR];
    Fy_all = [Fy_FL, Fy_FR, Fy_RL, Fy_RR];

    % Calculate SUMs
    SUM_Ff_lat = Fy_FL + Fy_FR;
    SUM_Fr_lat = Fy_RL + Fy_RR;
    SUM_lat = SUM_Ff_lat + SUM_Fr_lat;

    SUM_Ff_long = Fx_FL + Fx_FR;
    SUM_Fr_long = Fx_RL + Fx_RR;
    SUM_long = SUM_Ff_long + SUM_Fr_long;

    % Calculate dummy variables
    phi_ddot = ((-(SUM_lat)*H2) - ((K - W * H2) * phi) - c_roll * phi_dot) / Ixx;
    theta_ddot = (((SUM_long)*H2) - (K_bar * theta) - c_pitch * theta_dot) / Iyy;
    % Yaw acceleration and lateral acceleration
    r_dot = (SUM_Ff_lat * a - SUM_Fr_lat * b + (Fx_FL - Fx_FR + Fx_RL - Fx_RR)*T/2)
/ Izz;
    v_dot = (SUM_lat / m) - (u * r) - H2 * phi_ddot;
    u_dot = (SUM_long / m) + (v * r); % Should I add a phi_ddot term?
    glob_X_dot = u * cos(psi) - v * sin(psi);
    glob_Y_dot = u * sin(psi) + v * cos(psi);
    psi_dot = r;

    % Extracting acceleration
    a_y = SUM_lat / m;
    a_x = SUM_long / m;

    % State derivatives
    Xdot(1, 1) = r_dot;
    Xdot(2, 1) = v_dot;
    Xdot(3, 1) = glob_X_dot;
    Xdot(4, 1) = glob_Y_dot;
    Xdot(5, 1) = psi_dot;
    Xdot(6, 1) = phi_dot;
    Xdot(7, 1) = phi_ddot;

    % Derivatives for new variables
    Xdot(8, 1) = u_dot;
    Xdot(9, 1) = theta_dot;
    Xdot(10, 1) = theta_ddot;
end
```

```
function [decel, BF_f, BF_r, N_f, N_r, mu_f, mu_r, eta] = brake_no_trailer(car,
decel)
m = car.m; a = car.a; L = car.L; b = car.b; g=car.g; H2 = car.H2; W = car.W; BLIMIT
= car.BLIMIT; Q_brak = car.Q_brak; K_dbar = car.K_dbar;

Nf_static = W*b/L;
Nr_static = W*a/L;

% Calculate rear brake force
BF_r = W * decel / (1 + Q_brak);
BF_f = Q_brak*BF_r;

if BF_r > BLIMIT
  BF_r = BLIMIT;
  if Q_brak ~= 0 %Check if we have rear-brakes only-case
    BF_f = W * decel - BF_r;
  end
end

% Calculate normal loads
N_f = Nf_static + W * (decel) * H2 / L;
N_r = Nr_static - W * (decel) * H2 / L;

% Calculate coefficients of friction
mu_f = BF_f / N_f;
mu_r = BF_r / N_r;

% Determine effective coefficient of friction
mu = mu_f;
if mu_r > mu
    mu = mu_r;
end

% Calculate brake efficiency
eta = decel / mu;
end
```

```matlab
function [mu_fL, mu_fR, mu_rL, mu_rR, F_wx, F_wy] = fric_coeff(X, psi, car, Y)
T = car.T; a = car.a; b=car.b;
% Distance between CP and a front wheel
r_a=(a^2+(T/2)^2)^0.5;
% Distance between CP and a rear wheel
r_b=(b^2+(T/2)^2)^0.5;
% Angle between the x-axis of the car and the distance vector to the front
% wheel
v_a=atan(T/(2*a));
% Angle between the x-axis of the car and the distance vector to the rear
v_b=atan(T/(2*b));

% Orientation equation
FR_wx=X+r_a*(cos(v_a)*cos(psi)-sin(v_a)*sin(psi));
RR_wx=X+r_b*(-cos(v_b)*cos(psi)-sin(v_b)*sin(psi));
RL_wx=X+r_b*(-cos(v_b)*cos(psi)+sin(v_b)*sin(psi));
FL_wx=X+r_a*(cos(v_a)*cos(psi)+sin(v_a)*sin(psi));

FR_wy=Y+r_a*(cos(v_a)*sin(psi)+sin(v_a)*cos(psi));
RR_wy=Y+r_b*(-cos(v_b)*sin(psi)+sin(v_b)*cos(psi));
RL_wy=Y+r_b*(-cos(v_b)*sin(psi)-sin(v_b)*cos(psi));
FL_wy=Y+r_a*(cos(v_a)*sin(psi)-sin(v_a)*cos(psi));
F_wx = [FR_wx, RR_wx, RL_wx, FL_wx];
F_wy = [FR_wy, RR_wy, RL_wy, FL_wy];


% Set friction coefficients based on wheel locations
if  (FL_wx < 0) %If the front left positive then i
    mu_fL = 0.3;
else
    mu_fL = 0.9;
end
if  (FR_wx < 0)
    mu_fR = 0.3;
else
    mu_fR = 0.9;
end

if  (RL_wx < 0)
    mu_rL = 0.3;
else
    mu_rL = 0.9;
end

if  (RR_wx < 0)
    mu_rR = 0.3;
else
    mu_rR = 0.9;
end
end
```

```
function [Fx_tire, Fy_tire, LU] = NLTire(C_alpha, alpha, mu, N, BF, varargin)
if ~isempty(varargin)
    BLIMIT = varargin{1};
end
LU = 0;
Fx = -BF;
Fy = -C_alpha*alpha;
    if abs(Fy) > (mu*N)/2
       Fy = -mu*N*sign(alpha) * (1 - (mu*N)/(4*C_alpha*abs(tan(alpha))));
       LU = 2;
    end
% Check for lockup
F_check = -mu*N*cos(alpha);
    if abs(F_check) < BF
        Fy = -mu*N*sin(alpha);
        Fx = F_check;
        LU = 0.5;
    elseif (mu*N)^2 < (BF^2 + Fy^2)
        Fy = -sign(alpha)*sqrt((mu*N)^2 - (BF)^2);
        Fx = -BF;
        LU = 1;
    end
if ~isempty(varargin)
    if BLIMIT == 2*BF
        LU = 1.5;
    end
end
Fy_tire = Fy;
Fx_tire = Fx;
end
```

Extra: The brute force optimization algorithm

```
function [t_opt, u_i_opt, decel_opt, X_brake_opt, delta_f_opt] = BF_algorithm
(u_values, decel_values, X_brake_values, delta_f_values, car)
% Generate combinations of values
[comb_u, comb_decel, comb_X_brake, comb_delta_f] = ndgrid(u_values, decel_values,
X_brake_values, delta_f_values);

% Reshape the combinations into columns
comb_u = comb_u(:);
comb_decel = comb_decel(:);
comb_X_brake = comb_X_brake(:);
comb_delta_f = comb_delta_f(:);

% Combine all combinations into a single matrix
combinations = [comb_u, comb_decel, comb_X_brake, comb_delta_f];

t_opt = 10^9;
for i = 1:length(comb_u)
    ui = comb_u(i);

    decel = comb_decel(i);
    X_brake = comb_X_brake(i);
    delta_f = comb_delta_f(i);

    % Initial conditions
    r0 = 0; v0 = 0; glob_X0 = X_initial; glob_Y0 = 0; psi0 = psi_initial; phi0 = 0;
phi_dot0 = 0; u0 = ui; theta0 = 0; theta_dot0 = 0; %Velocity at beginning is given

    % Initial conditions with additional state variables
    X0 = [r0; v0; glob_X0; glob_Y0; psi0; phi0; phi_dot0; u0; theta0; theta_dot0];
    [X, a_y, a_x, N_all, Fx_all, BF, ~, LU] = odeRK4(@car_modelNL, X0, tspan, car);
    psi = X(end,5); glob_X = X(end,3); glob_Y = X(end, 4);
    [~,~,~,~, F_wx, F_wy] = fric_coeff(glob_X, psi, car, glob_Y);
    u_end = X(end, 8);
        if u_end < 10 && all(LU(:) == 0) && -F_wy(end,2) < 0 && -F_wy(end,4) > 0 &&
-0.3 < glob_Y < 0.3
            t = tspan(length(X));
            store_vals = [t, ui, decel, X_brake, delta_f];
            if t < t_opt
               t_opt = t;
               u_i_opt = ui;
               decel_opt = decel;
               X_brake_opt = X_brake;
               delta_f_opt = delta_f;
            end
        end
        if mod(i, 1000) == 0
            fprintf('Iteration number i: %d\n', i);
        end
end
end
```

Extra function - my own Runge Kutta 4th order ode solver

```
function [Y, a_y, a_x, N_all, Fx_all, BF, delta_f, LU, Fy_all] = odeRK4(dYdt, Y0↙
t, car)
% get size of problem
ne = length(Y0); % no. equations
nt = length(t);  % timesteps
h  = t(2)-t(1);  % step size

% initialize output
Y = zeros(nt,ne);  %(tom matrix)
Y(1,:) = Y0(:)'; % store results row-wise
% marching forward through time
    for n = 1:nt-1
        Yn = Y(n,:)';    %Take the first row of Y and turn it into a column vector
        [~, a_y(n+1,:),a_x(n+1,:), N_all(n+1,:), Fx_all(n+1,:), BF(n+1,:), delta_f↙
(n+1,:), LU(n+1,:), Fy_all(n+1,:)] = dYdt(Yn, t(n), car); %Extraction lateral↙
accelerations
        k1 = dYdt(Yn,            t(n), car);
        k2 = dYdt(Yn+(h/2)*k1,  t(n)+(h/2), car);
        k3 = dYdt(Yn+(h/2)*k2,  t(n)+(h/2), car);
        k4 = dYdt(Yn+h*k3,      t(n)+h, car);
        Y(n+1,:) = Yn + h/6*k1 + h/3*(k2+k3) + h/6*k4;
        if Y(n,3) >= 50  %If x-position is above 50 meters stop
            Y(n+2:end,:) = [];
            return
        end

    end
end
```