Simulation 4
ME – 565
Spring, 2024

By: Andres F. Hoyos

Consider a vehicle with the following parameters:

$W = 3000$ lbs,
$W_s = 2700$ lbs,
$x_1 = 3.5$ ft,
$x_2 = -4.5$ ft,
$h = -1.0$ ft,
$t = 6.0$ ft,
$I_z = 40000$ lbs*ft^2,
$I_x = 15000$ lbs*ft^2,
$c = 0.5$ ft,
$\frac{\partial L}{\partial \phi_f} = 40000$ lbs*ft,
$\frac{\partial L}{\partial \phi_r} = 5000$ lbs*ft,
$\frac{\partial L}{\partial \dot{\phi}_f} = 1000$ lbs*ft/sec,
$\frac{\partial L}{\partial \dot{\phi}_f} = 500$ lbs*ft/sec,
$p = d = 12$ in,
$\eta = 15:1$,
$\epsilon = 1$ (steering gearbox efficiency, NOT roll steer coefficient – let's call it 1),
$K_s = 10$ in*lbs/deg, and
$t_m = 3$ in.

Cornering coefficients:
For the linear part of the simulation use: $C_i = 140$ lbs/deg.

The following formula for nonlinear tire stiffness can be used where W is the weight on a given tire in pounds (lbs):

$$C_i = (0.2 * W_i - 0.0000942 * W_i^2) \text{ lbs/deg.}$$

Initial setup:

Before proceeding with the simulation, an initial set up is required. The first step will be to convert units since some units are incompatible. **Note:** ft are going to be used instead of in.

Unit conversions:

1. $I_z = \frac{I_z}{g} = \frac{40000}{32.174} = 1243.2$ lb*ft → lbm*ft.

2. $I_x = \frac{I_x}{g} = \frac{15000}{32.174} = 466.215$ lb*ft → lbm*ft.

3. $p = \frac{p}{12} = \frac{12}{12} = 1$ in → ft.

4. $d = \frac{d}{12} = \frac{12}{12} = 1$ in → ft.

5. $K_s = (K_s)(\frac{1}{12})(\frac{180}{\pi})$ in*lbs/deg → in*lbs/rad

6. $t_m = \frac{t_m}{12} = \frac{3}{12}$ in → ft.

Some necessary calculations:

Masses can be computed as

$$m = \frac{W}{g} = 93.243 \quad \text{lbm},$$

$$m_s = \frac{W_s}{g} = 83.919 \quad \text{lbm}.$$

The cornering stiffnesses of each axle can be calculated from the given information. The cornering stiffnesses are for the tires, hence, to get the cornering stiffness of the axle, the left and right tire cornering stiffnesses for that axle should be added and the units can be converted to compatible ones with the notation used on this work:

$$C_1 = (C_{1r} + C_{1l})(\frac{180}{\pi}) = (140 + 140)(\frac{180}{\pi}) = 16043 \quad \text{lbs/rad},$$

$$C_2 = (C_{2r} + C_{2l})(\frac{180}{\pi}) = (140 + 140)(\frac{180}{\pi}) = 16043 \quad \text{lbs/rad}.$$

The following Matlab script loads the parameters into the workspace and performs all the calculations previously mentioned (**Note:** this code snippet should be executed before running the next scripts for part 1):

```
%% Load some initial values:
% Conversion factors:
deg2rad = pi / 180;
rad2deg = 180 / pi;
in2ft = 1 / 12;
ft2in = 12;
mph2ftps = 5280 / 3600;
ftps2mph = 3600 / 5280;
% Bicycle model parameters:
W = 3000; % lbs
Ws = 2700; % lbs
g = 32.174; % ft/sec^2
x1 = 3.5; % ft
x2 = -4.5; % ft = -1.0; % ft
h = -1.0; % ft
track_width = 6.0; % ft
Iz = 40000 / g; % lbs*ft^2
Ix = 15000 / g; % lbs*ft^2
c = 0.5; % ft
dl_phi_f = 8000; % lbs*ft
dl_phi_r = 5000; % lbs*ft
dl_dphi_f = 1000; % lbs*ft
dl_dphi_r = 500; % lbs*ft
p = 12*in2ft; % ft
d = 12*in2ft; % ft
gear_ratio = 15; % []
efficiency = 1.0; % Steering gearbox efficiency
Ks = 10*in2ft*rad2deg; % (in*lbs/deg)*(ft/in)*(deg/rad) -> ft*lbs / rad
tm = 3*in2ft; % in*(ft/in)-> ft - Pneumatic trail
% Masses:
m = W / g;
ms = Ws / g;
% u = 30; % ft/sec
C1 = 2*140*180/pi; %*rad2deg; % lbs/deg * (deg/rad) -> lbs / rad
C2 = 2*140*180/pi; % *rad2deg; % lbs/deg * (deg/rad) -> lbs / rad
l2 = x1 - x2; % Wheelbase
dt = 0.01;
t_initial = 0;
t_final = 5;
```

Part 1: 2-DoF Vehicle (Linear + Non-Linear)

1. Develop a base 2-DoF linear state space yaw rate model and calculate the system eigen values when forward speed is 30 and 60 mph.

The 2-DoF Yaw Plane Model in state space form is given by

$$\begin{bmatrix}\dot{\beta}\\ \dot{r}\end{bmatrix}=\begin{bmatrix}\frac{-C_1-C_2}{mu} & \frac{-x_1C_1-x_2C_2}{mu}-1\\ \frac{-x_1C_1-x_2C_2}{I_z} & \frac{-x_1^2C_1-x_2^2C_2}{I_zu}\end{bmatrix}\cdot\begin{bmatrix}\beta\\ r\end{bmatrix}+\begin{bmatrix}\frac{C_1}{mu}\\ \frac{x_1C_1}{I_z}\end{bmatrix}\cdot[\delta]\ ,$$

where

$$A=\begin{bmatrix}\frac{-C_1-C_2}{mu} & \frac{-x_1C_1-x_2C_2}{mu}-1\\ \frac{-x_1C_1-x_2C_2}{I_z} & \frac{-x_1^2C_1-x_2^2C_2}{I_zu}\end{bmatrix}\quad \text{and}\quad B=\begin{bmatrix}\frac{C_1}{mu}\\ \frac{x_1C_1}{I_z}\end{bmatrix}\ .$$

The eigenvalues of A matrix are the pole locations of the system. Hence, the eigenvalues for u=30 mph and u=60 mph can be calculated from A matrix. To solve the problem, the following Matlab script was written:

```
%% 1. Develop a base 2-DoF linear state space yaw rate model and calculate the system eigen values
% when forward speed is 30 and 60 mph
u30 = 30*mph2ftps;
u60 = 60*mph2ftps;
A30 = [
(-C1 - C2)/(m*u30), (((-x1*C1 - x2*C2)/(m*u30^2)) - 1);
(-x1*C1 - x2*C2)/(Iz), (-x1*x1*C1 - x2*x2*C2)/(Iz*u30);
]
A60 = [
(-C1 - C2)/(m*u60), ((-x1*C1 - x2*C2)/(m*u60^2)) - 1;
(-x1*C1 - x2*C2)/(Iz), (-x1*x1*C1 - x2*x2*C2)/(Iz*u60);
]
eg_30mph = eig(A30)
eg_60mph = eig(A60)
```

Running this code generates the following results:

```
A30 =

   -7.8206   -0.9111
   12.9040   -9.5314


A60 =

   -3.9103   -0.9778
   12.9040   -4.7657


eg_30mph =

  -8.6760 + 3.3205i
  -8.6760 - 3.3205i


eg_60mph =

  -4.3380 + 3.5262i
  -4.3380 – 3.5262i
```

Analysis: Given these results, it can be seen that the **real part** of the eigenvalues have been halved after increasing the speed from 30 mph to 60 mph. It can be seen that the system became slower after performing this change. Moreover, it can be seen that as the speed increased from 30 mph to 60 mph, the **imaginary part** increased in magnitude which means that the transient response at 60 mph is going to have more oscillations than the transient response at 30 mph since imaginary poles produce oscillatory behaviors.

2. Using the steady-state yaw rate response, construct plots from 0 to 120 mph (i.e. 10 by 10) for various speeds, every 10 mph or so.

The following code is written in Matlab to solve the exercise:

```
%% 2. Using the steady-state yaw rate response, construct plots from 0 to 120 mph (i.e. r/    ) for various
% speeds, every 10 mph or so.
speeds = linspace(10, 120, 12)*mph2ftps;
delta2r_gain_arr = zeros(1, length(speeds));
t = linspace(t_initial, t_final, (t_final - t_initial) / dt);
figure;
hold on;
grid on;
title('Yaw rate steady state gains for different speeds')
xlabel('Time (seconds)');
ylabel('Gain (r/    )');
xlim([0, 5]);
```

```
ylim([0, 10.0]);
K_understeer = -m*(x1*C1 + x2*C2)/(C1*C2*l2);
legend_arr = cell(1, length(speeds));
for i = 1:length(speeds)
u = speeds(i);
delta2r_gain_arr(i) = u / (l2 + u*u*K_understeer);
% plot each gain in an xy plot:
yline(delta2r_gain_arr(i),'--', ['s.s gain = ' num2str(delta2r_gain_arr(i)) ' @ ' num2str(u*ftps2mph) 'mph'], 'Color', rand(1,3),
'LineWidth', 2);
% plot(t, delta2r_gain_arr(i)*ones(length(t)), '--', 'LineWidth', 2);
legend_arr{i} = [num2str(u*ftps2mph) ' mph'];
hold on;
end
legend(legend_arr{:}, 'Location', 'eastoutside');
hold off;
```

Note: the code converts the speed units from mph to ft/sec and then converts back to mph to plot the results.

The code generates the following results:

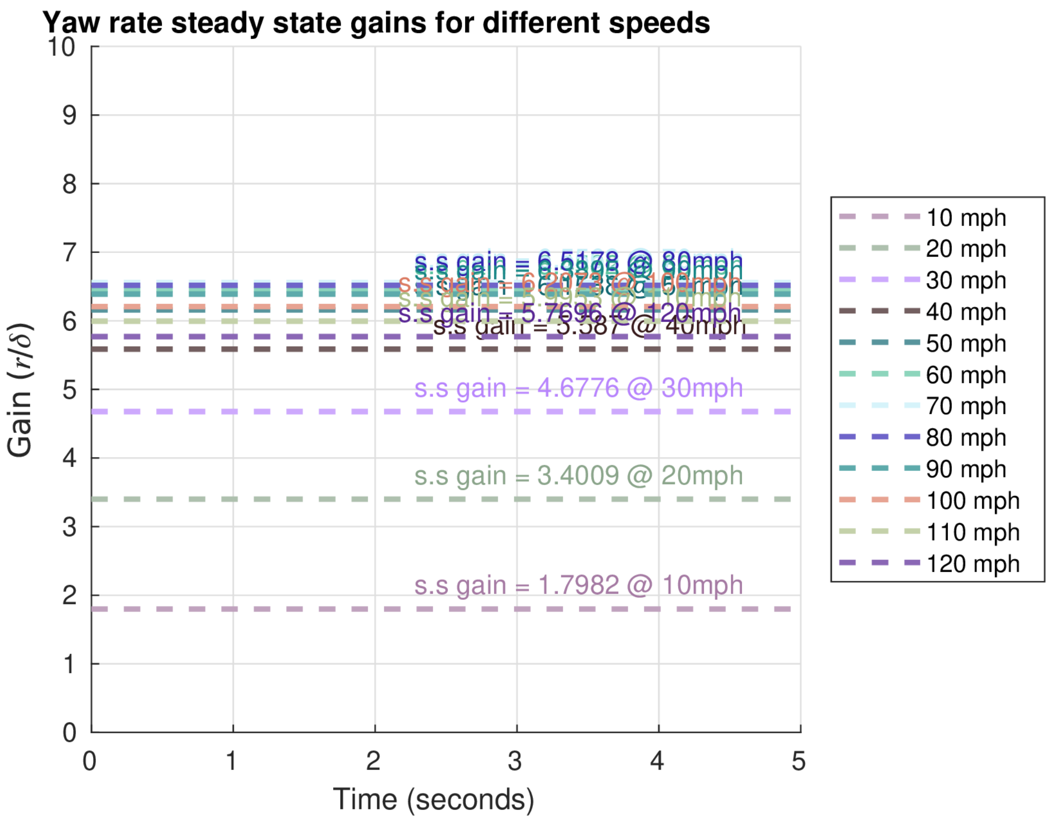


Figure 1: Shows the steady state gains of this vehicle for different speeds.

Analysis: It can be seen from Figure 1 that before 70 mph, the steady-state gain increases, but after this value, the steady state gain decreases. A more in depth analysis will be to actually calculate the **characteristic speed** that will give information on exactly at which speed does the yaw rate steady state gain starts decreasing. The characteristic speed can be calculated as follows

$$u_{char}=\sqrt{\frac{x_1-x_2}{K_{understeer}}}=\sqrt{\frac{8}{7.2652*10^{-4}}}=104.94 \quad \text{ft/sec = 71.55 mph, where}$$

$$K_{understeer}=\frac{-m(x_1C_1+x_2C_2)}{C_1C_2l_2}=7.2652*10^{-4} \quad .$$

This value is consistent with the experiment shown in Figure 1.

3. Using the 2-DoF equations of motion, simulate the vehicle response to a steering input. Determine the steering input required to result in a 400 ft radius turn. (hint: knowing the forward speed and radius of a circle, the required yaw rate can be calculated). Repeat the result at increments of 10 mph from 10 to 120 mph. Compare the results with the steady-state results from 2) above. We are using this step to validate your model.

To result in a turn radius of 400 ft, the steady state gain from steering wheel angle to yaw rate and the centripetal acceleration equation can be used to determine the required steering wheel angle δ as follows

$$\frac{r_{ss}}{\delta}=\frac{(u/R)}{\delta}=\frac{u}{l_2+u^2K_{understeer}} \quad ,$$

thus

$$\delta=\frac{l_2+u^2K_{understeer}}{R} \quad .$$

The following code is written in Matlab to solve this exercise applying the formula just presented to calculate the steering input required to produce a 400 ft radius turn:

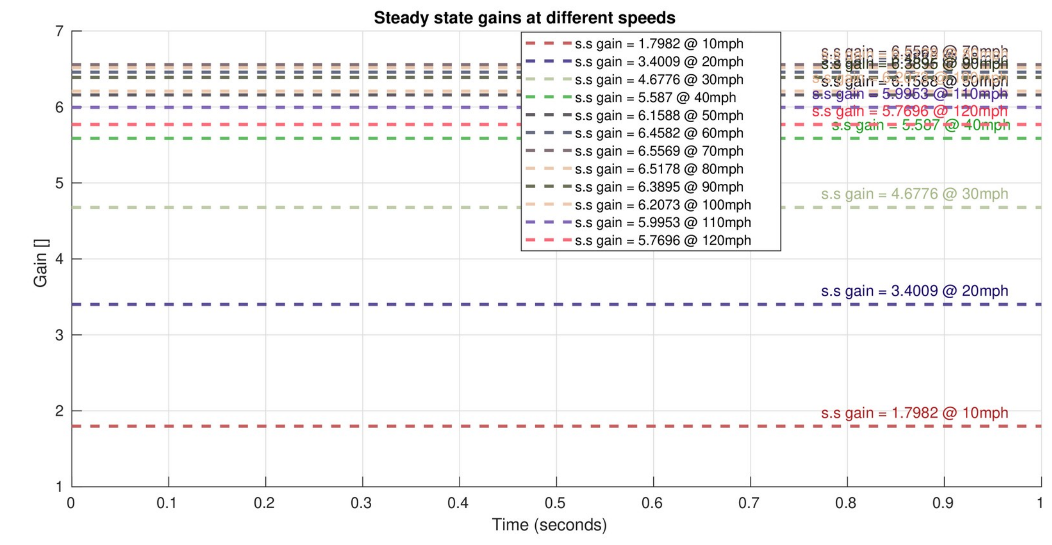
```
%% 3. Using the 2-DoF equations of motion, simulate the vehicle response to a steering input.
% Determine the steering input required to result in a 400 ft radius turn. (hint: knowing the forward
% speed and radius of a circle, the required yaw rate can be calculated). Repeat the result at
% increments of 10 mph from 10 to 120 mph. Compare the results with the steady-state results from
% 2) above. We are using this step to validate your model.
radius = 400; % ft
speeds = linspace(10, 120, 12);
speeds = speeds*mph2ftps;
% Time domain simulation:
t_initial = 0; % sec
t_final = 5; % sec
dt = 0.01; % sec
t = linspace(t_initial, t_final, (t_final - t_initial) / dt);
figure;
subplot(3,1,1);
title('Steady state gains at different speeds');
xlabel('Time (seconds)');
ylabel('Gain []');
```

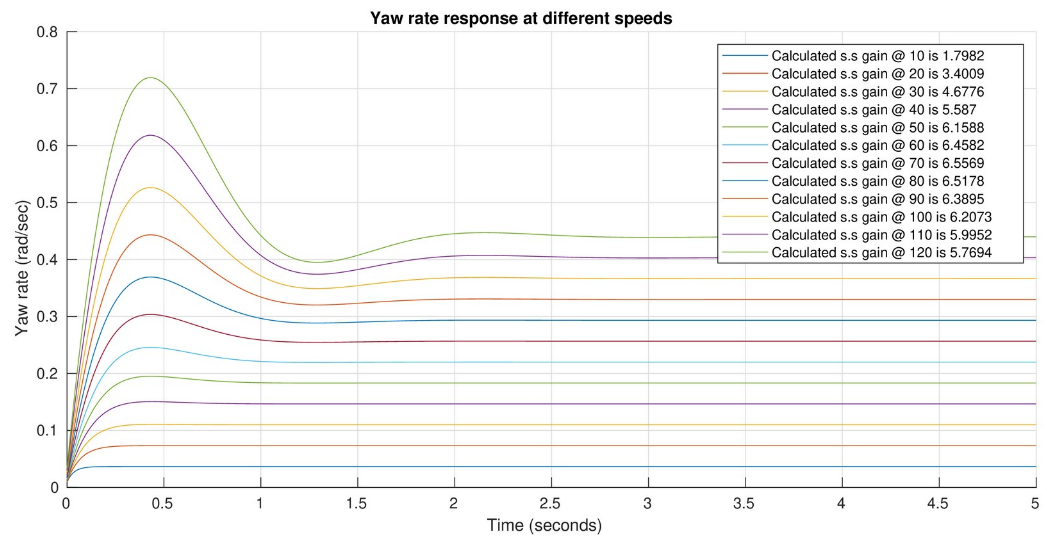
```
hold on;
grid on;
subplot(3,1,2);
title('Yaw rate response at different speeds');
xlabel('Time (seconds)');
ylabel('Yaw rate (rad/sec)');
hold on;
subplot(3,1,3);
title('Steering angle input at different speeds');
xlabel('Time (seconds)');
ylabel('Steering angle (rad)');
hold on;
legend_gains_arr = cell(1, length(speeds));
legend_yaw_rate_arr = cell(1, length(speeds));
legend_steering_angle_arr = cell(1, length(speeds));
for i = 1:length(speeds)
u = speeds(i);
subplot(3,1,1);
yline(delta2r_gain_arr(i),'--', ['s.s gain = ' num2str(delta2r_gain_arr(i)) ' @ ' num2str(u*ftps2mph) 'mph'], 'Color', rand(1,3),
'LineWidth', 2);
% plot(t, delta2r_gain_arr(i)*ones(length(t)), '--', 'LineWidth', 2);
legend_gains_arr{i} = ['s.s gain = ' num2str(delta2r_gain_arr(i)) ' @ ' num2str(u*ftps2mph) 'mph'];
subplot(3,1,2);
delta2 = (1 / radius)*(l2 + (u*u*K_understeer))*ones(1, length(t));
states_arr2 = simulate_bike_2dof(dt, t, m, x1, x2, C1, C2, Iz, u, delta2);
plot(t, states_arr2(2,:));
legend_yaw_rate_arr{i} = ['Calculated s.s gain @ ' num2str(u*ftps2mph) ' is ' num2str(states_arr2(2,
length(t))/delta2(length(t)))];
subplot(3,1,3);
plot(t, delta2);
legend_steering_angle_arr{i} = ['    = ' num2str(delta2(1)) ' @ ' num2str(u*ftps2mph) 'mph'];
end
subplot(3,1,1);
legend(legend_gains_arr);
grid on;
subplot(3,1,2);
legend(legend_yaw_rate_arr);
grid on;
subplot(3,1,3);
legend(legend_steering_angle_arr);
grid on;
hold off;

function states_arr = simulate_bike_2dof(dt, t, m, x1, x2, C1, C2, Iz, u, delta)
% Bicycle model using attack angle conventions:
% Attack angle is the negative of the slip angle
% The sign is embedded in the x1 and x2 distances
A = [
(-C1 - C2)/(m*u), ((-x1*C1 - x2*C2)/(m*u^2)) - 1;
(-x1*C1 - x2*C2)/(Iz), (-(x1^2)*C1 - (x2^2)*C2)/(Iz*u);
];
B = [
(C1)/(m*u);
(x1*C1)/Iz;
];
% Discretize using Euler:
A_dis = eye(2) + dt * A;
B_dis = dt * B;
```

```
% Initial conditions:
states = [0; 0];
states_arr = zeros(2,length(t));
% simulation loop:
for i = 1:length(t)
states = A_dis * states + B_dis * delta(i);
states_arr(:, i) = states;
end
end
```

This code generates the following result:


Steady state gains at different speeds
Gain []
Time (seconds)
s.s gain = 1.7982 @ 10mph
s.s gain = 3.4009 @ 20mph
s.s gain = 4.6776 @ 30mph
s.s gain = 5.587 @ 40mph
s.s gain = 6.1588 @ 50mph
s.s gain = 6.4582 @ 60mph
s.s gain = 6.5569 @ 70mph
s.s gain = 6.5178 @ 80mph
s.s gain = 6.3895 @ 90mph
s.s gain = 6.2073 @ 100mph
s.s gain = 5.9953 @ 110mph
s.s gain = 5.7696 @ 120mph
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
1
2
3
4
5
6
7


Yaw rate response at different speeds
Yaw rate (rad/sec)
Time (seconds)
Calculated s.s gain @ 10 is 1.7982
Calculated s.s gain @ 20 is 3.4009
Calculated s.s gain @ 30 is 4.6776
Calculated s.s gain @ 40 is 5.587
Calculated s.s gain @ 50 is 6.1588
Calculated s.s gain @ 60 is 6.4582
Calculated s.s gain @ 70 is 6.5569
Calculated s.s gain @ 80 is 6.5178
Calculated s.s gain @ 90 is 6.3895
Calculated s.s gain @ 100 is 6.2073
Calculated s.s gain @ 110 is 5.9952
Calculated s.s gain @ 120 is 5.7694
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8

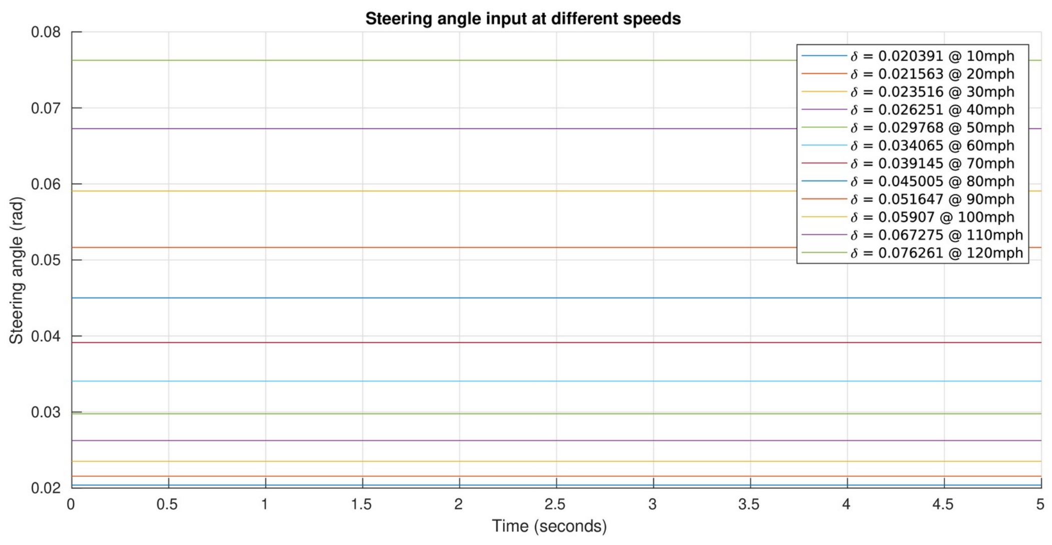

Steering angle input at different speeds
Steering angle (rad)
Time (seconds)
δ = 0.020391 @ 10mph
δ = 0.021563 @ 20mph
δ = 0.023516 @ 30mph
δ = 0.026251 @ 40mph
δ = 0.029768 @ 50mph
δ = 0.034065 @ 60mph
δ = 0.039145 @ 70mph
δ = 0.045005 @ 80mph
δ = 0.051647 @ 90mph
δ = 0.05907 @ 100mph
δ = 0.067275 @ 110mph
δ = 0.076261 @ 120mph
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
0.02
0.03
0.04
0.05
0.06
0.07
0.08

Figure 2: shows in the **top subplot** the steady state gains from steering wheel angle to yaw rate at different speeds, the **middle subplot** shows the time response of the yaw rate given the steering angle input signal shown in the **bottom plot**.

Analysis: This code generates the plot shown in Figure 2, this figure has 3 subplots. The **top subplot** shows the steady state gains from steering wheel angle to yaw rate for different speeds. The **subplot in the middle** shows the time response of the yaw rate given the steering input in the **bottom subplot.**

Notice that the steady state gain values of the **middle subplot** on Figure 2 match with the steady state gains (plotted as horizontal lines) in the **top subplot**, this is a method of verification for the mathematical model proposed to solve this exercise. The steady state gain from question 1 gives information on the steady state gain, and this value is confirmed by the gain at which the yaw rate stabilized given the specified inputs.

Also, notice that as the speed increases, the steering wheel input signal increases, this makes sense as taking a sharper turn at high speeds will require higher controller effort.

4. Determine a particular time series of handwheel inputs. We define this input as a 0° handwheel input for 1 second, then a +45° handwheel input for 3 seconds, then a −45° handwheel input for 3 seconds, then back to 0° for 3 seconds. Practically we know that the maximum handwheel input a driver can perform is 2 rev/sec, so modify the steering input so that the handwheel velocity is consistent with this value with a 180° input amplitude. Plot the time history of the steering input .

The steering rate saturation in radians per second is:

$sat = 2*2*\pi = 4\pi$ rad/sec.

The following code is written in Matlab to solve this exercise:

```
%% 4. Determine a particular time series of handwheel inputs. We define this input as a 0° handwheel
% input for 1 second, then a +45° handwheel input for 3 seconds, then a −45° handwheel input for 3
% seconds, then back to 0° for 3 seconds. Practically we know that the maximum handwheel input a
% driver can perform is 2 rev/sec, so modify the steering input so that the handwheel velocity is
% consistent with this value with a 180° input amplitude. Plot the time history of the steering input     .
% Time array:
t_initial = 0;
t_final = 10;
t = linspace(t_initial, t_final, (t_final - t_initial)/dt);
% Generate the specified handwheel input:
delta = zeros(1, length(t));
% Input of 0° for 1 second:
delta(1, 1:1/dt) = 0;
% Input of +45° for 3 seconds:
delta(1, 1/dt:4/dt) = 0.707;
% Input of -45° for 3 seconds:
delta(1, 4/dt:7/dt) = -0.707;
% Input of 0° for 3 seconds:
delta(1, 7/dt:length(t)) = 0;
```

```
% Plot input:
figure;
plot(t, delta);
grid on;
title('Steering input with no rate saturation');
xlabel('time (sec)');
ylabel('delta (rad)');
% Modify the steering input:
slope = 2*(2*pi); % rad/sec
% Smooth step applied at 1 second:
left_bound = 0;
right_bound = 0.707;
at = 1; % /dt;
delta_mod = slope_it_down(t, dt, at, delta, left_bound, right_bound, sign(right_bound - left_bound)*slope);
% Smooth step applied at 4 seconds:
left_bound = 0.707;
right_bound = -0.707;
at = 4; % /dt;
delta_mod = slope_it_down(t, dt, at, delta_mod, left_bound, right_bound, sign(right_bound - left_bound)*slope);
% Smooth step applied at 7 seconds:
left_bound = -0.707;
right_bound = 0;
at = 7; % /dt;
delta_mod = slope_it_down(t, dt, at, delta_mod, left_bound, right_bound, sign(right_bound - left_bound)*slope);
% Plot input:
figure;
plot(t, delta);
hold on;
plot(t, delta_mod, 'r');
grid on;
title('Steering input with smoothed angle transitions');
xlabel('time (sec)');
ylabel('delta (rad)');
legend('Original input signal', 'Modified input signal');


function delta = slope_it_down(t, dt, at, delta, left_bound, right_bound, slope)
time_to_reach = (right_bound - left_bound) / slope; % time units
delta(1, at/dt:(at + time_to_reach)/dt) = left_bound + slope * (t(at/dt:(at + time_to_reach)/dt) - t(at/dt));
end
```

This codes generates the steering signal shown in Figure 3 .

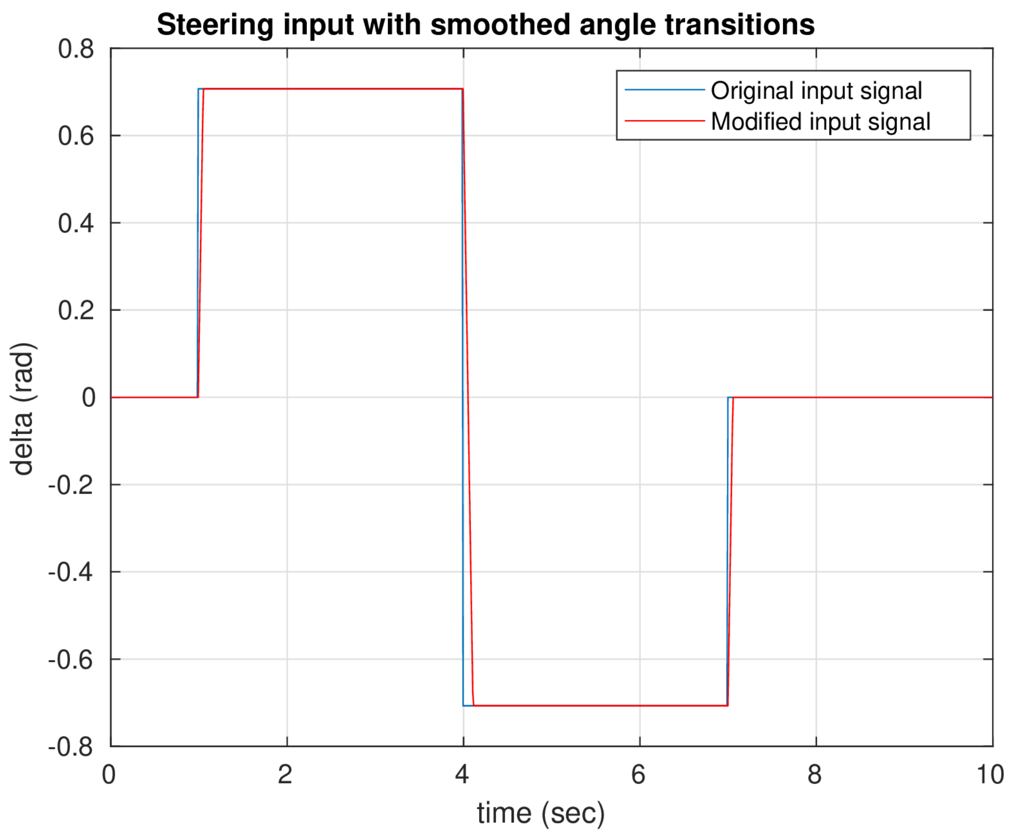


Figure 3: Steering input signal with steering angle rate saturation of 2 rev/sec. The plot compares the original input with no rate saturation (blue) and the modified input signal with rate saturation (red).

Analysis: From Figure 3 it can be see that the steering angle now transitions from angle to angle with a ramp steady state. Physically this makes more sense as a human driver is not able to command a steering wheel with step signals.

However, even though the transitions have been smoothed out, this input signal is quite aggressive as the wheel are being suddenly turned to 45 deg and then from 45 deg to -45 deg and then to 0 deg; this is a signal that can potentially induce high lateral forces given the amplitude and the rapid rate of change, specially when the input signals transitions from a ramp to a constant as it produces a jump in steering rate and steering angular acceleration.

5. Use the steering input calculated in 4) as an input to your simulation. Plot the time history of the state variables drift angle and yaw rate r at 30 and 60 mph.

The following code is developed in Matlab to perform this simulation and generate the plots:

```
%% 5. Use the steering input calculated in 4) as an input to your simulation. Plot the time history of the
% state variables drift angle     and yaw rate r at 30 and 60 mph.
% Speeds of interest:
u30 = 30*mph2ftps;
u60 = 60*mph2ftps;
% Time domain simulation:
states_arr30 = simulate_bike_2dof(dt, t, m, x1, x2, C1, C2, Iz, u30, delta_mod);
states_arr60 = simulate_bike_2dof(dt, t, m, x1, x2, C1, C2, Iz, u60, delta_mod);
% Plot the states:
figure;
subplot(2,1,1);
plot(t, states_arr30(1,:));
xlabel('time (sec)');
ylabel('drift angle (rad)');
grid on;
% title('yaw rate for 30 mph');
hold on;
plot(t, states_arr60(1,:));
xlabel('time (sec)');
ylabel('drift angle (rad)');
title('Drift angle for 30 and 60 mph');;
legend('30 mph', '60 mph');
subplot(2,1,2);
plot(t, states_arr30(2,:));
xlabel('time (sec)');
ylabel('yaw rate (rad/sec)');
grid on;
hold on;
plot(t, states_arr60(2,:));
title('Yaw rate for 30 and 60 mph');
legend('30 mph', '60 mph');
```

This code generates the plot shown in Figure 4.

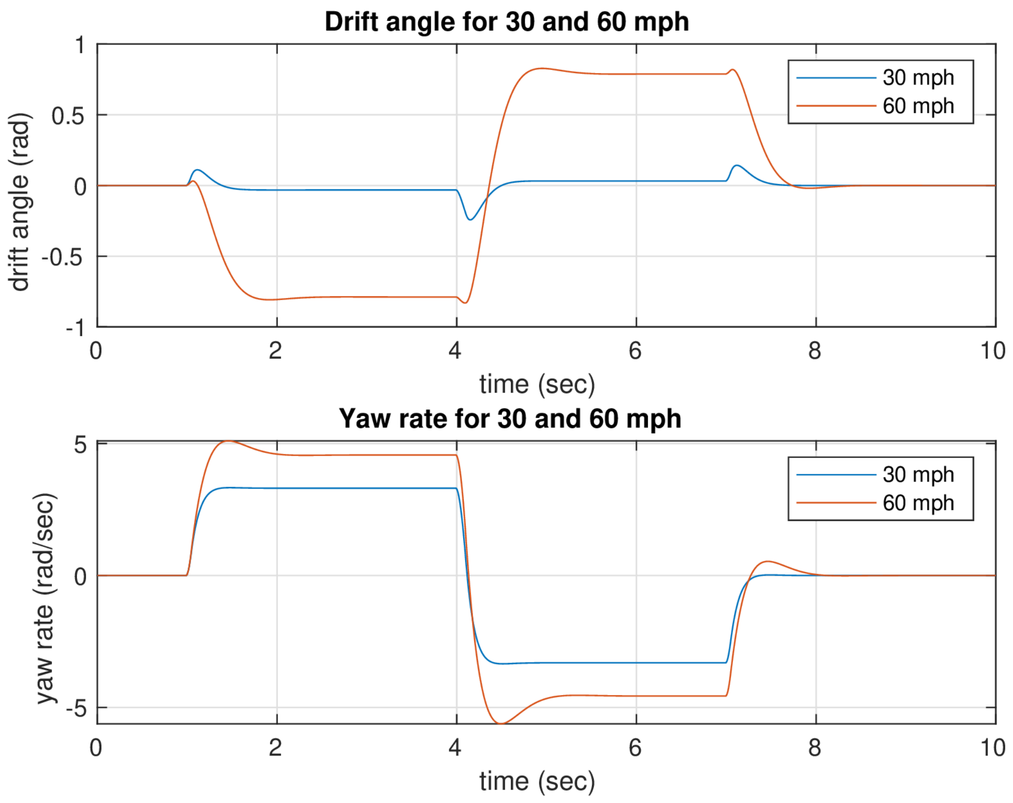


Figure 4: Yaw rate and drift angle time plots for 30 and 60 mph longitudinal speeds.