Dynamic model of the car body

It is stated in section 1.1 of the design brief that "… the car can be modelled as a mass-spring-damper system …". In Figure 1 the model that will be used to derive the equations of motion for the car body is shown.

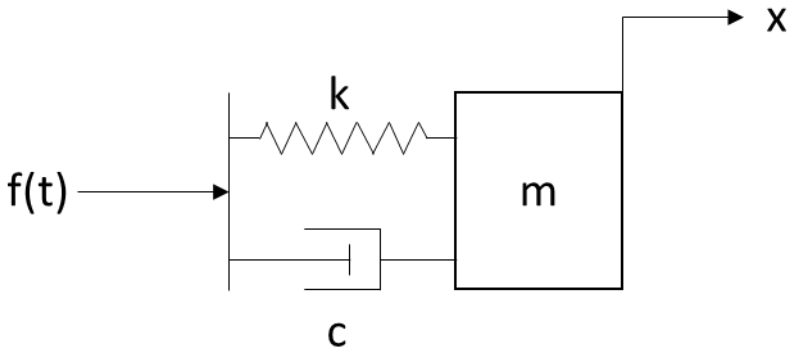

Figure 1: Mass-spring-damper equivalent model of the car body

In Figure 1 the spring constant is given by k (N/m), the damping factor is given by c (Ns/m), the car mass is given by m (kg) and the applied force is given by f(t) (N); the positive x-direction is as shown.

The free body diagram that relates to the model is given in Figure 2 below, where the parameters k, c, m and f(t) are as described above.

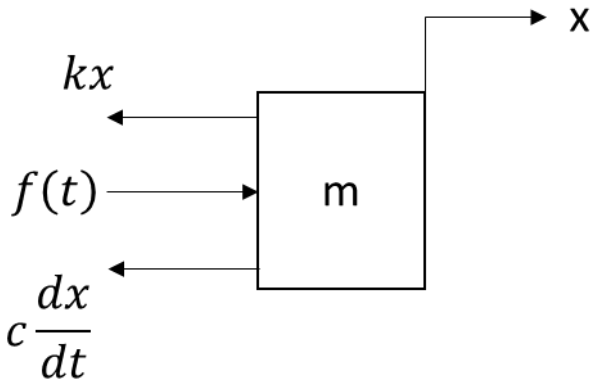

Figure 2: Car body Free Body Diagram

The equation of motion for the car body is now developed.

$$\sum F = m\ddot{x}$$

$$m\ddot{x} = -c\dot{x} - kx + f(t)$$

$$\ddot{x} + \frac{c}{m}\dot{x} + \frac{k}{m}x = \frac{1}{m}f(t) \qquad (1)$$

Laplace transform equation (1) ignoring any initial conditions.

$$\left(s^2 + \frac{c}{m}s + \frac{k}{m}\right)X(s) = \frac{1}{m}F(s)$$

and hence

$$\frac{X(s)}{F(s)} = \frac{\frac{1}{m}}{\left(s^2 + \frac{c}{m}s + \frac{k}{m}\right)}$$

(2)

The transfer function in equation (2) relates position in the x-direction to applied force. The speed of the vehicle is the parameter that is required to be controlled; hence, the transfer function in equation (2) requires an s term to be applied to the numerator. Multiplication by s in the Laplace domain is equivalent to differentiation in the time domain. Thus

$$G_p(s) = \frac{V(s)}{F(s)} = \frac{\frac{1}{m}s}{\left(s^2 + \frac{c}{m}s + \frac{k}{m}\right)}$$

(3)

In equation (3) the term V(s) is the Laplace transform of the velocity of the car body.

The PID controller that will control the car speed is given by

$$G_c(s) = k_p + \frac{k_i}{s} + k_d\frac{1}{1 + \frac{N}{s}}$$

$$= \frac{k_d N s^2 + k_p s(s + N) + k_i(s + N)}{s(s + N)}$$

(4)

In equation (4) k_d is the derivative gain, k_p is the proportional gain, k_i is the integral gain and N is the filter coefficient. The closed loop control system used to control the speed of the car is shown below in Figure 3. It should be noted that the disturbance input and any required limiting functions for engine power are not included in this configuration.

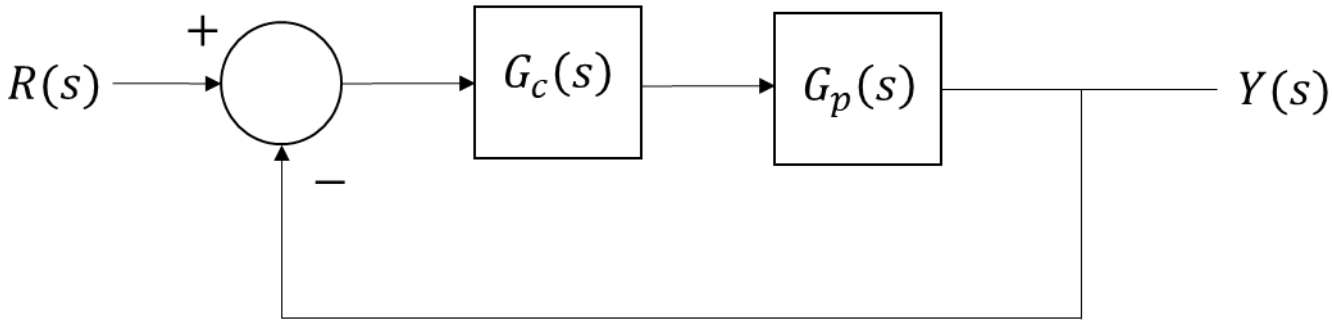

Figure 3: Closed loop car speed control system

In Figure 3 the PID controller is represented by $G_c(s)$ and the car engine and body dynamics by $G_p(s)$. The closed loop transfer function can be shown to be given by

$$\frac{Y(s)}{R(s)} = \frac{G_c(s)G_p(s)}{1 + G_c(s)G_p(s)} \tag{5}$$

Substituting for $G_c(s)$ and $G_p(s)$ from equations (4) and (3) respectively, into equation (5) gives

$$\frac{Y(s)}{R(s)} = \frac{\left(k_d N s^2 + k_p s(s+N) + k_i(s+N)\right)}{m(s+N)\left(s^2 + \frac{c}{m}s + \frac{k}{m}\right) + \left(k_d N s^2 + k_p s(s+N) + k_i(s+N)\right)} \tag{6}$$

For a step input $R(s) = \frac{1}{s}$

$$Y(s) = \frac{\left(k_d N s^2 + k_p s(s+N) + k_i(s+N)\right)}{s\left(m(s+N)\left(s^2 + \frac{c}{m}s + \frac{k}{m}\right) + \left(k_d N s^2 + k_p s(s+N) + k_i(s+N)\right)\right)}$$

The steady state output is found from

$$y_{ss} = \lim_{t\to\infty} y(t) = \lim_{s\to 0} sY(s)$$

$$\lim_{s\to 0} sY(s) = \lim_{s\to 0}\left(\frac{\left(k_d N s^2 + k_p s(s+N) + k_i(s+N)\right)}{\left(m(s+N)\left(s^2 + \frac{c}{m}s + \frac{k}{m}\right) + \left(k_d N s^2 + k_p s(s+N) + k_i(s+N)\right)\right)}\right)$$

hence

$$y_{ss} = \frac{k_i}{(k + k_i)}$$

Thus, it can be seen that for zero steady state error to a unit step input, then k must be zero.

Setting k equal to zero in equation (3) gives

$$\frac{V(s)}{F(s)} = \frac{\frac{1}{m}s}{\left(s^2 + \frac{c}{m}s\right)}$$

$$\frac{V(s)}{F(s)} = \frac{\frac{1}{m}}{\left(s + \frac{c}{m}\right)} \tag{7}$$

A further consequence of having a non-zero value for k is that, since there would be a steady state error, then the controller output would continue to increase, or decrease, due to the integral term attempting to reduce the steady state error to zero.

The above result, k must be equal to zero for zero steady state error, applies equally to a system controlled by a PI controller; the PI controller would have the following transfer function.

$$G_c(s) = k_p + \frac{k_i}{s}$$

Simulink model of the car engine and car body dynamics

In section 1.1 of the design brief, it states that "The combined effect of the car and rolling resistance constitute the damping of the vehicle, given by an effective damping coefficient of c = 41." The units for c are not given and so the assumed units for c are Ns/m. Additionally, the design brief gives the maximum engine power as 90kW. The car will travel at a constant velocity when the power developed by the car engine is equal to the power required to overcome the car damping effects. Hence, for constant velocity

$$P_{engine} = P_{car}$$

$$f(t)v(t) = cv^2(t)$$

$$f(t) = cv(t) \tag{8}$$

A constant velocity is achieved when there is power balance between the engine and the car which is the same as having force balance between the engine and car as seen in equation (8).

Knowing the maximum engine power and the value of the damping coefficient allows the maximum speed of the car to be calculated.

$$v_{max} = \sqrt{\frac{90{,}000}{41}} = 46.85\ (ms^{-1}) \tag{9}$$

Figure 4 shows the Simulink model of the car engine, car body dynamics and the disturbance input to the system. The engine model has to incorporate a mechanism to limit the maximum power and hence the maximum force that can be applied to the car body.

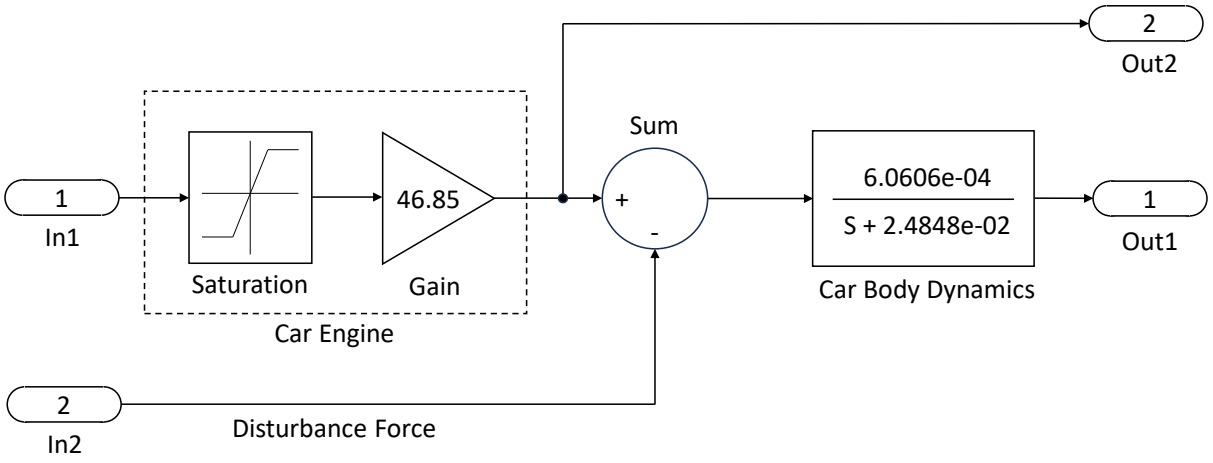

Figure 4: Simulink model of the engine and car body dynamics subsystem

The output of the speed controller is applied to the model at terminal In1. The maximum force that the engine can apply to the car body is limited by the saturation block, to a value of 1921(N). This value is calculated from

$$F_{max} = \frac{P_{max}}{v_{max}} = \frac{90{,}000}{46.85} = 1921(N)$$

By limiting the controller output to 1921(N) and multiplying by the maximum velocity, the car engine cannot exceed the stated 90(kW) maximum power limit.

The road inclination disturbance is input at terminal In2 of the model. The disturbance magnitude is calculated from

$$D = mgsin(\theta)$$

(10)

Where m is the car mass (kg), g is the acceleration due to gravity 9.81(ms^{-2}) and θ is the inclination angle which can be either positive or negative in value. The disturbance force is only dependent on the inclination angle and hence does not require to be scaled to velocity.

The force developed by the car engine and the disturbance force are then summed; the result of which is input to the car body transfer function.

The force developed by the engine is output from the subsystem at terminal Out2, with the car body velocity being output at terminal Out1.

Simulink Control System Model

The setup in Simulink used to test the performance of the control system is shown in Figure 5 below. The car engine and body dynamics subsystem has already been described in the section immediately above.

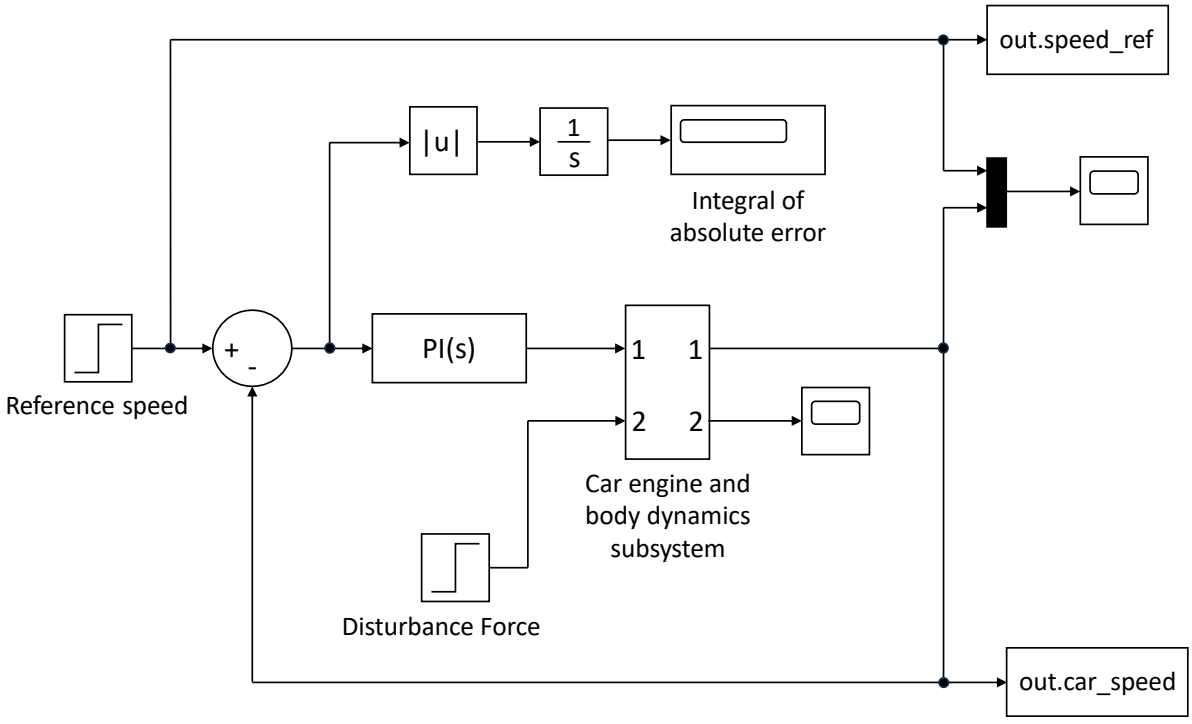

Figure 5: Control system model (PI control)

Three methods were used to tune the PID controller. These methods were:

1) tuning using the inbuilt tune function in Simulink and adjustment of the *speed of response* and *robustness* sliders to obtain an acceptable closed loop system response.
2) Using a script in Matlab.
3) Using a script in Matlab

The following script tunes a PID controller to meet the tuning goal of Setpoint tracking and Step Disturbance Rejection.

```
clear
close all
clc

% plant
m =1650; %car mass in kg
c = 41;  %car damping factor in Ns/m
Pmax = 90000; % car engine maximum power in W
Vmax = 46.85; %Car velocity scaling factor
s = tf('s');
Gp = Vmax*(1/m)/(s+c/m);

% PID controller
Kp = 37;
Ki = 0.972;
Kd = 11;
Tf = 0.4;

C0 = tunablePID('C', 'pid'); %Set up tuneable PID controller block C0
F0 = s/(s + c/m); %Model for step disturbance rejection

AP = AnalysisPoint('u');
T0 = feedback(Gp*AP*C0, 1);
T0.InputName = 'r';
T0.OutputName = 'y';

Req1 = TuningGoal.Tracking('r', 'y', 2);
Req2 = TuningGoal.StepRejection('u','y', F0);

rng('default');
Options = systuneOptions('RandomStart', 3);
[T, fSoft] = systune(T0, [Req1, Req2], Options);
getBlockValue(T, 'C')

Tdist = getIOTransfer(T, 'r', 'y');
fig_1 = figure;
stepplot(T);
grid;

% get position of fig_1 and shift position of Figure 1
pos1 = get(fig_1,'InnerPosition');
set(fig_1,'Position', pos1 + [-pos1(3)/2,0,0,0]);

fig_2 = figure;
viewGoal([Req1, Req2], T);

% get position of fig_2 and then shift position of Figure 2
pos2 = get(fig_2,'InnerPosition');
set(fig_2,'Position', pos2 + [pos2(3)/2,0,0,0])
```

The following script provides the Bode plots for the Sensitivity and Complementary Sensitivity Functions for the car cruise control system tuned with the PID controller parameters given in the script.

```
close all
clear
clc

%Set up the process and controller constants
%Process
c = 41;          %Damping constant
m = 1650;        %Mass of car
a = c/m;         %Car model pole
V = 46.85;       %Velocity scaling factor
%Controller
kp = 37;         %Proportional gain
ki = 0.964;      %Integral gain
kd = 10.9;       %Derivative gain
Tf = 2;          %Derivative filter time constant
N = 1/Tf;

%Set up the forward and closed loop transfer functions
s = tf('s');
Gp = V*(1/m)/(s + a);
Gc = (kp +ki/s + kd*(N/(1+N/s)));
Gp_ol = Gp * Gc;
G_cl = feedback(Gp_ol, 1);

%Bode plot of the complementary sensitivity function
w = logspace(-1, 2, 300);
[mag1, ph] = bode(G_cl, w);
magdb = 20.*log10(mag1(:));
fig_1 = figure;
semilogx(w, magdb(:))
grid
xlabel('Frequency (rad/s)');
ylabel('Complementary Sensitivity Magnitude (dB)');
title('Complementary Sensitivity Function');

% get position of fig_1 and shift position of Figure 1
pos1 = get(fig_1,'InnerPosition');
set(fig_1,'Position', pos1 + [-pos1(3)/2,0,0,0])

%Set up the sensitivity transfer function
sens = 1/(1 + Gp_ol);

%Bode plot of sensitivity function
[mag2, ph2] = bode(sens, w);
magdb2 = 20.*log10(mag2(:));
fig_2 = figure;
semilogx(w, magdb2(:))
grid
xlabel('Frequency (rad/s)');
ylabel('Sensitivity Magnitude (dB)');
title('Sensitivity Function');

% get position of fig_2 and shift position of Figure 2
pos2 = get(fig_2,'InnerPosition');
set(fig_2,'Position', pos2 + [pos2(3)/2,0,0,0])
```

The following script plots the sensitivity of the Closed Loop Transfer Function to the variation of the parameters of the PID controller; namely kp, ki and kd.

```
clear;
close all;
clc;

kp = 37;
ki = 0.966;
kd = 10.9;
N = 2.4;
V = 46.85;
m = 1650;
k = V/m;
a = 2.4848e-02;

s= tf('s');
w = logspace(-2, 1, 500);

fig_1 = figure;
for i = 1:9
    kp_spread = [15 20 25 30 35 40 45 50 55];
    Num_Gcl_kp = kp_spread(i)*k*(s + N)*s + ki*k*(s + N) + kd*k*N*s^2;
    Den_Gcl_kp = s*(s + N)*(s + a) + kp_spread(i)*k*(s + N)*s + ...
        ki*k*(s + N) + kd*k*N*s^2;
    Sen_Gcl_kp = kp_spread(i)*k*(s + N)*s*(1/Num_Gcl_kp - 1/Den_Gcl_kp);
    [mag, ph] = bode(Sen_Gcl_kp, w);
    magdb = 20.*log10(mag(:));
    semilogx(w, magdb(:));
    grid on;
    hold on;
end
xlabel('Frequency (rad/s)');
ylabel('Sensitivity Magnitude (dB)');
title('Sensitivity Function - kp');
legend('kp = 15', 'kp = 20', 'kp = 25', 'kp = 30', 'kp = 35', ...
    'kp = 40', 'kp = 45', 'kp = 50', 'kp = 55');
% get position of fig_1 and then shift position of Figure 1
pos1 = get(fig_1,'Position');
set(fig_1,'Position', pos1 - [pos1(3),0,0,0]);
hold off;

fig_2 = figure;
for i = 1:9
    ki_spread = [0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4];
    Num_Gcl_ki = kp*k*(s + N)*s + ki_spread(i)*k*(s + N) + kd*k*N*s^2;
    Den_Gcl_ki = s*(s + N)*(s + a) + kp*k*(s + N)*s + ...
        ki_spread(i)*k*(s + N) + kd*k*N*s^2;
    Sen_Gcl_ki = ki_spread(i)*k*(s + N)*(1/Num_Gcl_ki - 1/Den_Gcl_ki);
    [mag, ph] = bode(Sen_Gcl_ki, w);
    magdb = 20.*log10(mag(:));
    semilogx(w, magdb(:))
    grid on;
    hold on;
end
```

```
xlabel('Frequency (rad/s)');
ylabel('Sensitivity Magnitude (dB)');
title('Sensitivity Function - ki');
legend('ki = 0.6', 'ki = 0.7', 'ki = 0.8', 'ki = 0.9', 'ki = 1', ...
    'ki = 1.1', 'ki = 1.2', 'ki = 1.3', 'ki = 1.4');
% get position of fig_2 and then shift position of Figure 2
pos2 = get(fig_2,'InnerPosition');
set(fig_2,'Position', pos2);
hold off;

fig_3 = figure;
for i = 1:9
    kd_spread = [6 7 8 9 10 11 12 13 14];
    Num_Gcl_kd = kp*k*(s + N)*s + ki*k*(s + N) + kd_spread(i)*k*N*s^2;
    Den_Gcl_kd = s*(s + N)*(s + a) + kp*k*(s + N)*s + ...
        ki*k*(s + N) + kd_spread(i)*k*N*s^2;
    Sen_Gcl_kd = kd_spread(i)*k*N*s^2*(1/Num_Gcl_kd - 1/Den_Gcl_kd);
    [mag, ph] = bode(Sen_Gcl_kd, w);
    magdb = 20.*log10(mag(:));
    semilogx(w, magdb(:))
    grid on;
    hold on;
end
xlabel('Frequency (rad/s)');
ylabel('Sensitivity Magnitude (dB)');
title('Sensitivity Function - kd');
legend('kd = 6', 'kd = 7', 'kd = 8', 'kd = 9', 'kd = 10', ...
    'kd = 11', 'kd = 12', 'kd = 13', 'kd = 14');
% get position of fig_3 and then shift position of Figure 3
pos3 = get(fig_3,'InnerPosition');
set(fig_3,'Position', pos3 + [pos3(3),0,0,0]);
hold off;
```

Some code for stuff

```
clear;
close all;
clc;

kp = 37;
ki = 0.972;
kd = 11;
Tf = 2.5;
N = 1/Tf;
m = 1650;
c = 41;
Vmax = 46.85;
k = Vmax/m;
a = c/m;

s = tf('s');
Num = k*(kd*N+kp)*(s^2 + k*(ki+kp*N)/(k*(kd*N+kp))*s + k*ki*N/(k*(kd*N+kp)));
Den = s^3 + (N+a+k*(kd*N+kp))*s^2 + (N*a+k*(ki+kp*N))*s + k*ki*N;
G = Num/Den;
T = minreal(G);

fig_1 = figure;
pole_zero = pzplot(T, 'b');
```

```
fig_2 = figure;
stepplot(T);

zero(T)
pole(T)

T
T_1 = k*(kd*N+kp)*(s+0.3545)*(s+0.0265)/((s+1.2377)*(s+0.3361)*(s+0.0265))



G = tf(1,[1 1 1]);
C = pidtune(G,'PI');
Tref = getPIDLoopResponse(C,G,"closed-loop");
Tdist = getPIDLoopResponse(C,G,"input-disturbance");
step(Tref,Tdist)
legend("Reference Tracking","Disturbance Rejection")

[C_pidf_fast,info] = pidtune(sys,'PIDF',1.0)
T_pidf_fast =  feedback(C_pidf_fast*sys,1);
step(T_pi_fast, T_pidf_fast);
axis([0 30 0 1.4]);
legend('PI,fast','PIDF,fast');
S_pi_fast = feedback(sys,C_pi_fast);
S_pidf_fast = feedback(sys,C_pidf_fast);
step(S_pi_fast,S_pidf_fast);
axis([0 50 0 0.4]);
legend('PI,fast','PIDF,fast');

sys = tf(1,[1 3 3 1]);
opts = pidtuneOptions('PhaseMargin',45,'DesignFocus','disturbance-rejection');
[C,info] = pidtune(sys,'pid',opts);



The following script tunes a PID controller to meet the tuning goal of Phase
Margin.


clear
close all
clc

%car model constants
m =1650; %car mass in kg
c = 41;  %car damping factor in Ns/m
Pmax = 90000; %car engine maximum power in W
Vmax = 46.85; %maximum car velocity

%car model
s = tf('s');
G = (1/m)/(s+c/m);

%generate PI controller parameters
[C_PI, info] = pidtune(G,'PI')
```

```
%determine the closed loop and disturbance step responses
T_PI_cl = getPIDLoopResponse(C_PI,G,"closed-loop");
T_PI_dist = getPIDLoopResponse(C_PI,G,"input-disturbance");
fig_1 = figure;
step(T_PI_cl,T_PI_dist);
legend("Reference Tracking","Disturbance Rejection")
grid
title('PI controller tuned using default phase margin');
% get position of fig_1 and shift position of Figure 1
pos1 = get(fig_1,'Position');
set(fig_1,'Position', pos1 + [-pos1(3)/2,pos1(4)/4,0,0]);

%generate PID controller parameters
[C_PIDF,info] = pidtune(G,'PIDF')

%display the closed loop and disturbance step responses
T_PIDF_cl = getPIDLoopResponse(C_PIDF,G,"closed-loop");
T_PIDF_dist = getPIDLoopResponse(C_PIDF,G,"input-disturbance");
fig_2 = figure;
step(T_PIDF_cl,T_PIDF_dist);
%axis([0 30 0 1.4]);
legend("Reference Tracking","Disturbance Rejection");
grid
title('PID controller tuned using default phase margin');
% get position of fig_2 and shift position of Figure 2
pos2 = get(fig_2,'Position');
set(fig_2,'Position', pos2 + [pos2(3)/2,pos2(4)/4,0,0])


%generate PI controller parameters for a target phase margin
opts = pidtuneOptions('PhaseMargin',50,'DesignFocus','disturbance-rejection');
[C_PI_opt,info_opt] = pidtune(G,'PI',opts)

%determine the closed loop and disturbance step responses
T_PI_opt_cl = getPIDLoopResponse(C_PI_opt,G,"closed-loop");
T_PI_opt_dist = getPIDLoopResponse(C_PI_opt,G,"input-disturbance");
fig_3 = figure;
step(T_PI_opt_cl,T_PI_opt_dist);
legend("Reference Tracking","Disturbance Rejection")
grid
title('PI controller tuned using option on phase margin');
% get position of fig_3 and shift position of Figure 3
pos3 = get(fig_3,'Position');
set(fig_3,'Position', pos3 + [-pos3(3)/2,-pos3(4)*0.95,0,0]);

%generate PID controller parameters for a target phase margin
opts = pidtuneOptions('PhaseMargin',45,'DesignFocus','disturbance-rejection');
[C_PIDF_opt,info_opt] = pidtune(G,'PIDF',opts)

%display the closed loop and disturbance step responses
T_PIDF_opt_cl = getPIDLoopResponse(C_PIDF_opt,G,"closed-loop");
T_PIDF_opt_dist = getPIDLoopResponse(C_PIDF_opt,G,"input-disturbance");
fig_4 = figure;
step(T_PIDF_cl,T_PIDF_dist);
%axis([0 30 0 1.4]);
legend("Reference Tracking","Disturbance Rejection");
grid
title('PID controller tuned using option on phase margin');
% get position of fig_4 and shift position of Figure 4
```

```
pos4 = get(fig_4,'Position');
set(fig_4,'Position', pos4 + [pos4(3)/2,-pos4(4)*0.95,0,0])
```