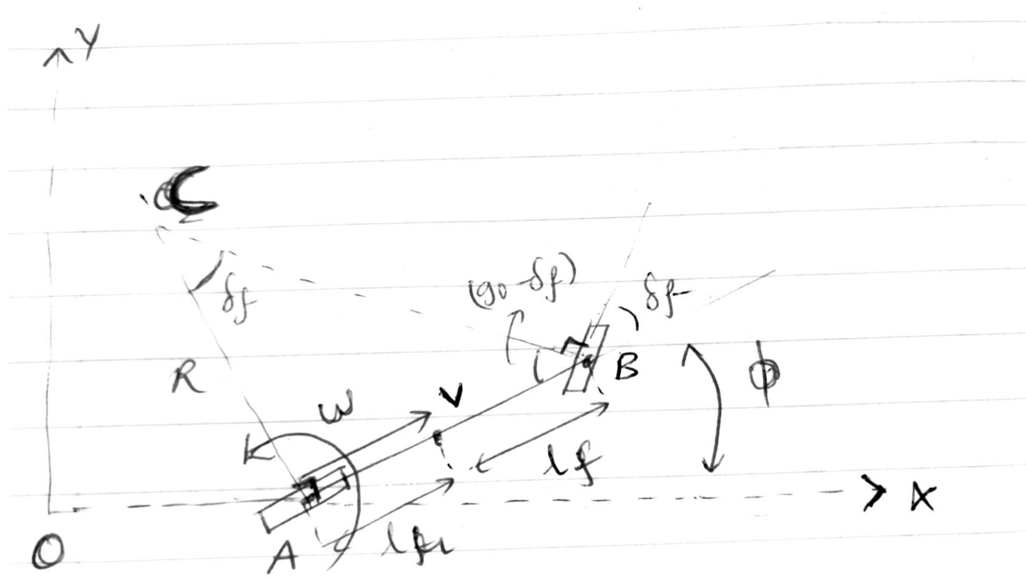
<u>16-665 Robot Mobility : Homework 01 (AD)</u>

Name: Kavin Kailash Ravie
Andrew ID: kravie **Date: 09/21/2023**

<u>Question 1.1 - Pepy KBM Model Derivation</u>



Pepy Kinematic Bicycle Model Geometry

From the above geometry, we formalize the following trivial observations

$$\frac{dx}{dt} = V cos(\phi)$$

$$\frac{dy}{dt} = V sin(\phi)$$

$$\frac{d\phi}{dt} = \omega$$

But we need to express the angular velocity ω in terms of the known variables. This can be achieved by analyzing the ΔABC.

We have, $\angle CBA = 90° - \delta_f$

And hence, $\angle BCA = \delta_f$ and we note that $tan(\delta_f) = \frac{l_f + l_r}{R}$

where R is the instantaneous radius of curvature of the vehicle.

But, $V = R\omega$, so $\omega = V\frac{tan(\delta_f)}{l_f + l_r}$

Finally we obtain:

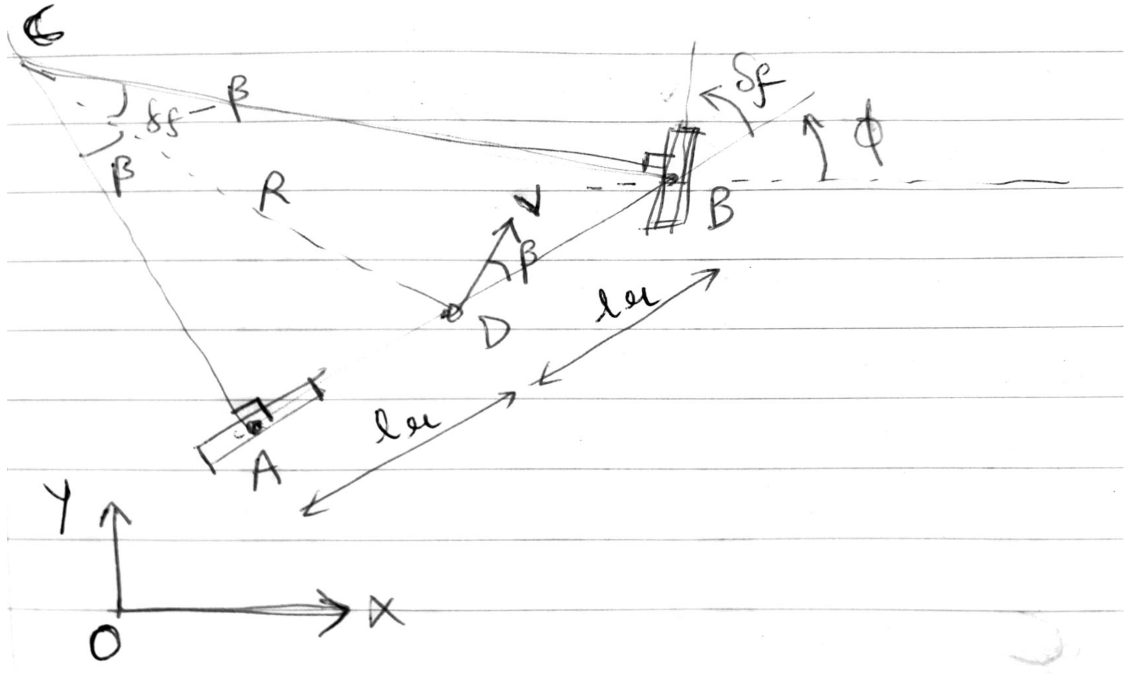
$$\frac{d\phi}{dt} = \omega = V\frac{tan(\delta_f)}{l_f + l_r}$$

Additionally we have the curvature $\frac{1}{R} = \frac{tan(\delta_f)}{l_f + l_r}$, this expression will be useful in Question 3.

Assumptions:

- Purely kinematic formulation without inclusion of any dynamic factors.
- No rear-wheel steering
- No tire-slip (velocity at the tires always along the tire)
- Real and Front axles clumped as single front and rear wheels

Question 1.2 - Kong Model Derivation



Kong Kinematic Bicycle Model Geometry

From the above figure for the Kong model. We make the following trivial observations:

$$\frac{dx}{dt} = V cos(\phi + \beta)$$

$$\frac{dy}{dt} = V sin(\phi + \beta)$$

$$\frac{d\phi}{dt} = \omega$$

But as in the case with the analysis done for **Pepy** model, we need to further express the angular velocity ω in terms of the given variables.

In ΔCAD, $\angle ACD = 180^\circ - \angle CDA - 90^\circ$, but $\angle CDA = 90 - \beta$ (i.e $180^\circ - 90^\circ - \beta$), hence $\angle ACD = \beta$

Then we have,

$$sin(\beta) = \frac{l_r}{R} \implies R = \frac{l_r}{sin(\beta)}$$

As V=Rω, we obtain $\omega = V\frac{sin(\beta)}{l_r}$ giving us $\omega = \frac{V sin(\beta)}{l_r}$

Assumptions:

- Purely kinematic formulation without inclusion of any dynamic factors.
- No rear-wheel steering
- No tire-slip (velocity at the tires always along the tire)
- Real and Front axles clumped as single front and rear wheels

Question 1.3

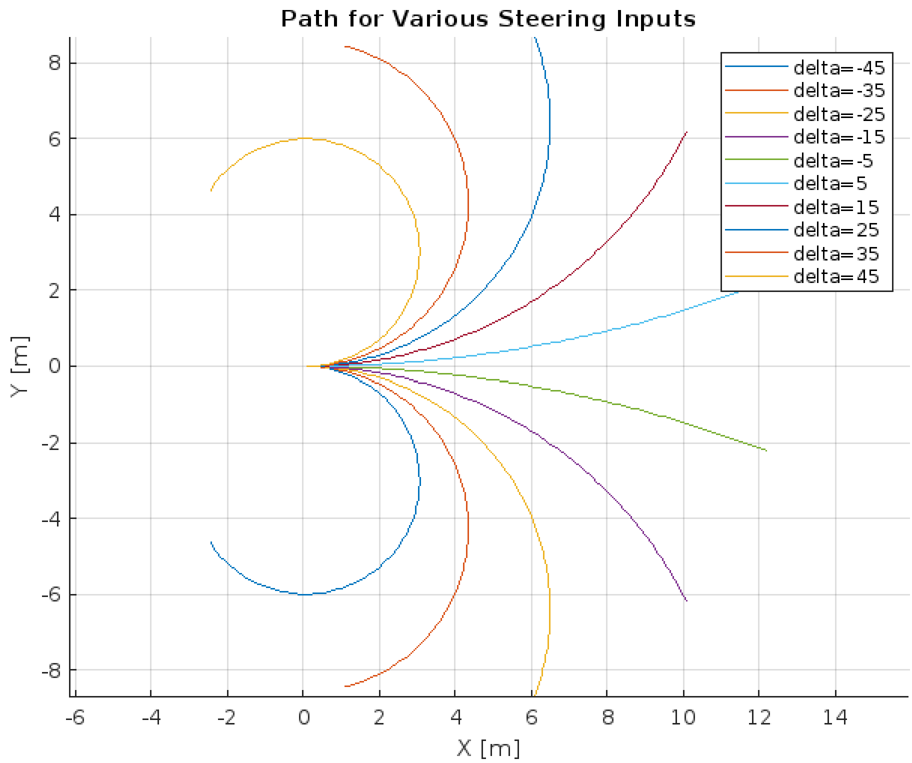
A.
   - No, this does not mean that the vehicle slip is zero. It just means that with this choice of placing the body-fixed coordinate reference, we no longer have to deal with the vehicle slip for our analysis. But in reality, the vehicle would still experience a vehicle slip when measured at the CoG. The expression for this can be obtained as: $\beta = atan(\frac{l_r\omega}{V}) \implies \beta = atan(\frac{tan(\delta_f)l_r}{lf + lr})$

B.
   - Vehicle Slip refers to the angle made by the velocity vector **w.r.t** the vehicle longitudinal axis.
   - Tire Slip refers to the angle made b/w the velocity vector at the tires and the tire longitudinal axis.
   - Yes, it is possible to have vehicle-slip without having any tire-slip, this is apparent from our analysis above for the Kong model where we have neglected any tire-slip affects and still ended up with a non-zero vehicle-slip. This is because vehicle-slip depends on the geometry of the vehicle and the choice of reference frame.

Question 1.4 - Pepy Model Simulation

Common Simulation Parameters: V = 2.50 m/sec, Tf = 5.0 sec and dt = 0.05 sec

Part A : Constant Steering Inputs (units in degrees)

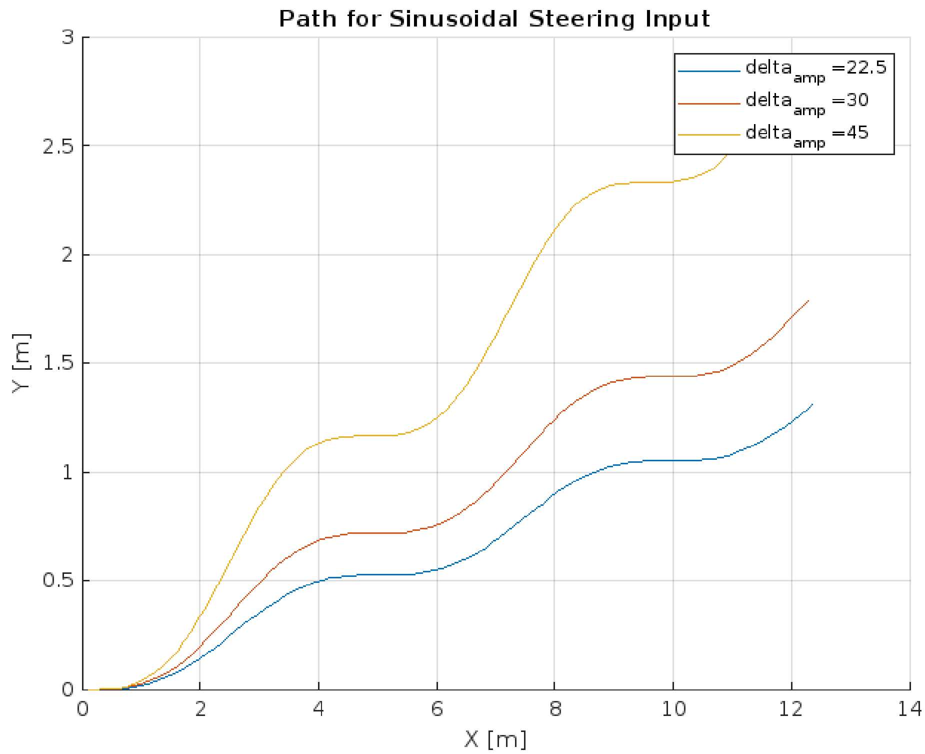


The mathematical expression for the path radius of curvature can be obtained from the expression derived in Q1.1, and it is as follows:

$$R = \frac{l_f + l_r}{tan(\delta_f)}$$

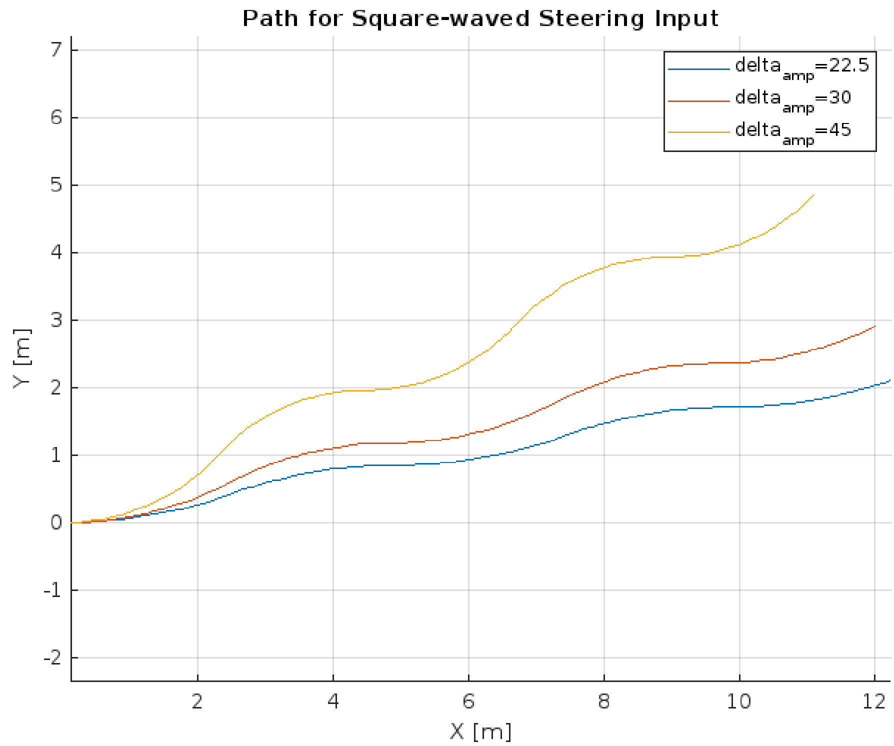
but , $l_f = l_r = 1.50$ hence, $R = \dfrac{3.0}{tan(\delta_f)}$

Case 2: Sinusoidal Steering Input (units in degrees)



- I expected the vehicle to follow a sinusoidal path about the global frame x-axis (as initial yaw state = 0.0), but we observe a different behavior from the plots (the path diverging from the x-axis). This can be explained by the fact that the vehicle position and heading/yaw is continually changing, causing a non-zero Hysteresis effect over the entire input cycle.

Case 3: Square-waved Steering Input (units in degrees)



- This is unrealistic because the rising and dropping edges of a perfect square wave have infinite slope implying that the desired rate of change of Yaw would be infinitely high. But in reality as physical systems have a maximum bound on this imposed by the mechanical design, dynamics, controllers and many other factors.
- Hence to alleviate this irregularity, I would instead use a trapezoidal steering profile (ramp), with the slopes of the rising and falling edges corresponding to the maximum attainable Yaw rates by the physical system.

Code

```
close all;
clear all;
clc;
% Model Parameters
lf = 1.50;
lr = 1.50;
A = [lf, lr]';
%% Part A
% Control Inputs
v = 2.50;
df = linspace(-pi/4,pi/4,10);
% Simultaion Time
dt = 0.05; Tf = 5.0;
ts = linspace(0, Tf, Tf/dt);
% Logging
X_log = zeros(3,length(ts),length(df));
X = [0,0,0]';
for i = 1:length(df)
X = [0,0,0]';
for j = 1:length(ts)
X = simulate_step(@pepyKBM,A,X,[v,df(i)],dt);
X_log(:,j,i) = X';
end
end
figure;
title('Path for Various Steering Inputs')
hold on;
grid on;
axis equal;
for i = 1:length(df)
plot(X_log(1,:,i),X_log(2,:,i),'DisplayName',strcat('delta=',num2str((df(i)*180
/pi))))
end
xlabel("X [m]")
ylabel("Y [m]")
title('Path for Various Steering Inputs')
legend('show')
figure;
hold on;
grid on;
% axis equal;
for i = 1:length(df)
plot(ts, X_log(1,:,i),'DisplayName',strcat('delta=',num2str((df(i)*180/pi))))
end
xlabel("t [s]")
ylabel("X [m]")
legend('show')
```

```
title('X Trajectory')
figure;
hold on;
grid on;
% axis equal;
for i = 1:length(df)
plot(ts, X_log(2,:,i),'DisplayName',strcat('delta=',num2str((df(i)*180/pi))))
end
xlabel("t [s]")
ylabel("Y [m]")
legend('show')
title('Y Trajectory')
figure;
hold on;
grid on;
% axis equal;
for i = 1:length(df)
plot(ts,
180/pi*X_log(3,:,i),'DisplayName',strcat('delta=',num2str((df(i)*180/pi))))
end
xlabel("t [s]")
ylabel("\Phi [deg]")
legend('show')
title('\Phi Trajectory')
%% Part B
% Simulation Time
dt = 0.05; Tf = 5.0;
ts = linspace(0, Tf, Tf/dt);
% Control Inputs
v = 2.50;
df_amp = [pi/8, pi/6, pi/4];
df = zeros(3,length(ts));
df_freq = 0.5;
for i=1:length(df_amp)
df(i,:) = df_amp(i)*sin(2*pi*df_freq*ts);
end
% Logging
X_log = zeros(3,length(ts), length(df));
X = [0,0,0]';
for i = 1:length(df_amp)
X = [0,0,0]';
for j = 1:length(ts)
X = simulate_step(@pepyKBM,A,X,[v,df(i,j)],dt);
X_log(:,j,i) = X';
end
end
figure;
hold on;
grid on;
```

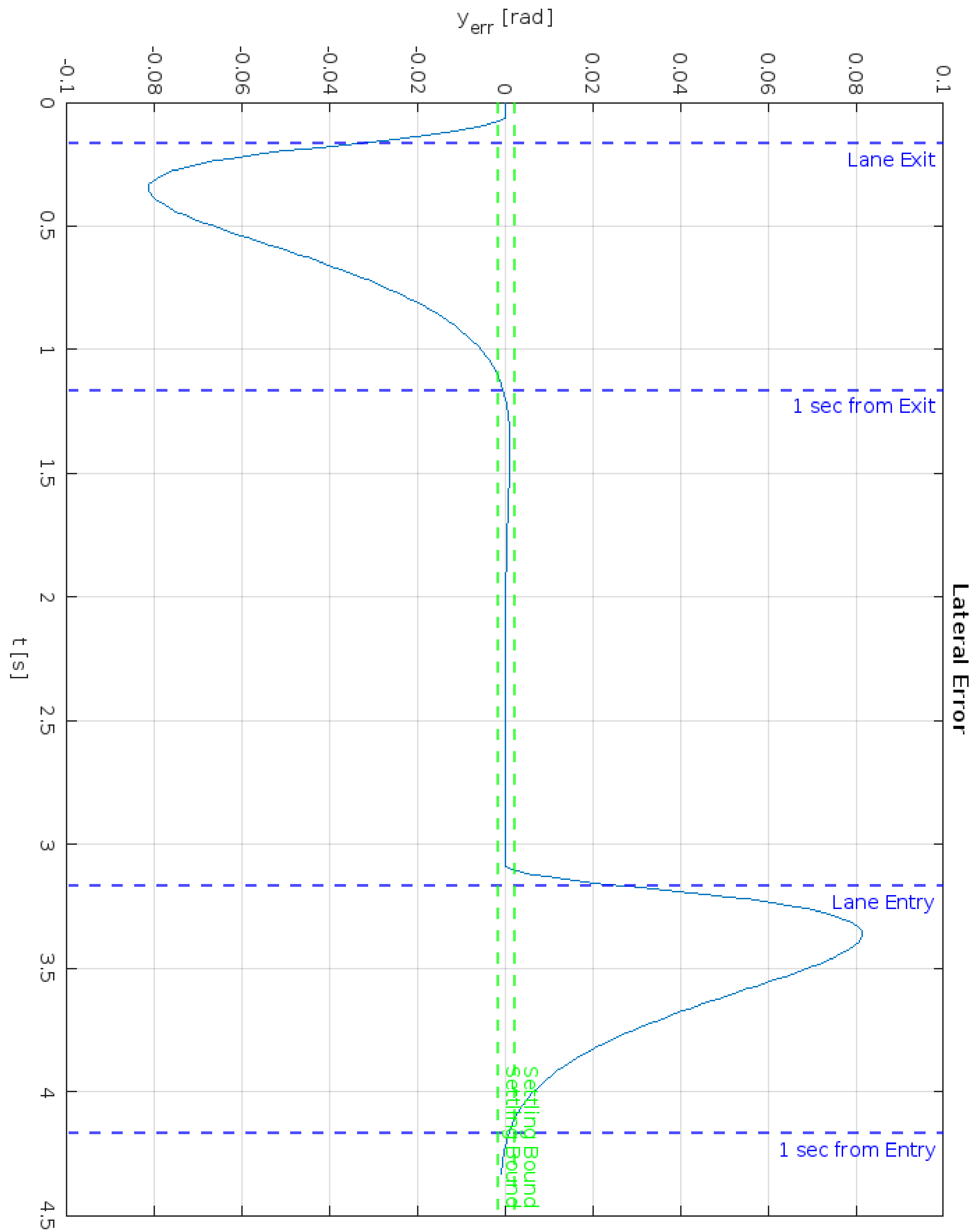
```
% axis equal;
for i = 1:length(df_amp)
plot(X_log(1,:,i),X_log(2,:,i),'DisplayName',strcat('delta_{amp}
=',num2str((df_amp(i)*180/pi))))
end
xlabel("X [m]")
ylabel("Y [m]")
legend('show')
title('Path for Sinusoidal Steering Input')
figure;
hold on;
grid on;
% axis equal;
for i = 1:length(df_amp)
plot(ts,X_log(3,:,i),'DisplayName',strcat('delta_amp=',num2str((df_amp(i)*180/p
i))))
end
ylabel("\delta_f [rad]")
xlabel("t [s]")
legend('show')
title('Steering Input (\delta_f) ')
% figure;
% plot(ts,X)
%% Part C
% Simulation Time
dt = 0.05; Tf = 5.0;
ts = linspace(0, Tf, Tf/dt);
% Control Inputs
v = 2.50;
df_amp = [pi/8, pi/6, pi/4];
df = zeros(3,length(ts));
df_freq = 0.5;
for i=1:length(df_amp)
df(i,:) = df_amp(i)*square(2*pi*df_freq*ts);
end
% Logging
X_log = zeros(3,length(ts), length(df));
X = [0,0,0]';
for i = 1:length(df_amp)
X = [0,0,0]';
for j = 1:length(ts)
X = simulate_step(@pepyKBM,A,X,[v,df(i,j)],dt);
X_log(:,j,i) = X';
end
end
figure;
hold on;
grid on;
axis equal;
```

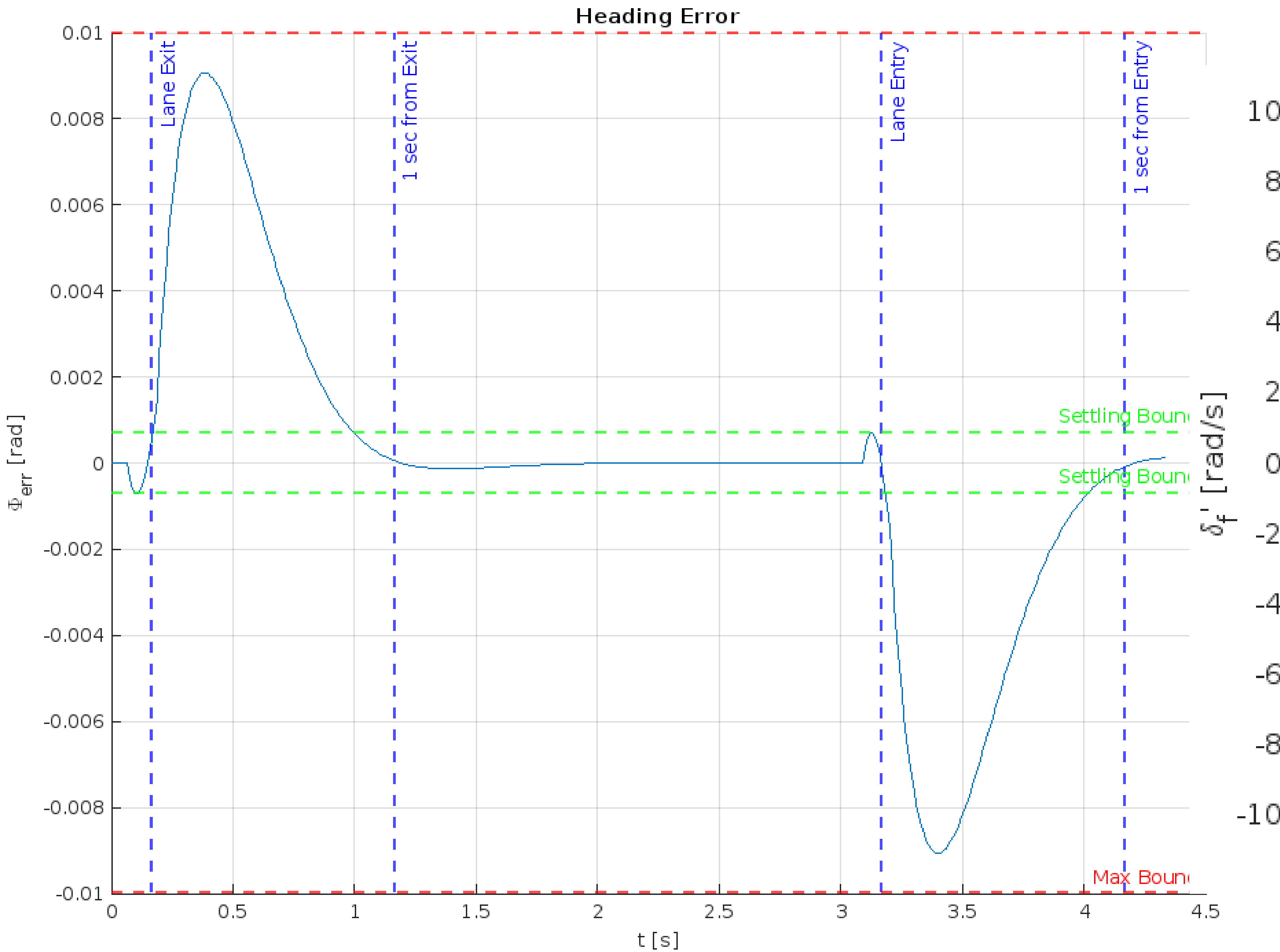
```
for i = 1:length(df_amp)
plot(X_log(1,:,i),X_log(2,:,i),'DisplayName',strcat('delta_{amp}=',num2str((df_
amp(i)*180/pi))))
end
xlabel("X [m]")
ylabel("Y [m]")
legend('show')
title('Path for Square-waved Steering Input')
figure;
hold on;
grid on;
% axis equal;
for i = 1:length(df_amp)
plot(ts,df(i,:),'DisplayName',strcat('delta_{amp}=',num2str((df_amp(i)*180/pi))
))
end
ylabel("\delta_f [rad]")
xlabel("t [s]")
legend('show')
title('Steering Input (\delta_f) ')
%%
figure
for i = 1:length(df_amp)
plot(ts,df(i,:))
hold on;
plot(ts,X_log(3,:,i),'DisplayName',strcat('delta_{amp}=',num2str((df_amp(i)*180
/pi))))
end

function Xdot = pepyKBM(A,X,U)
      Xdot = [0,0,0]';
      Xdot(1) = U(1)*cos(X(3));
      Xdot(2) = U(1)*sin(X(3));
      Xdot(3) = U(1)*tan(U(2))/(A(1)+A(2));
end

function Xn = simulate_step(dynamics, A, X, U,dt)
Xn = X + dynamics(A, X, U)*dt;
end
```

Q2.1 Model Development

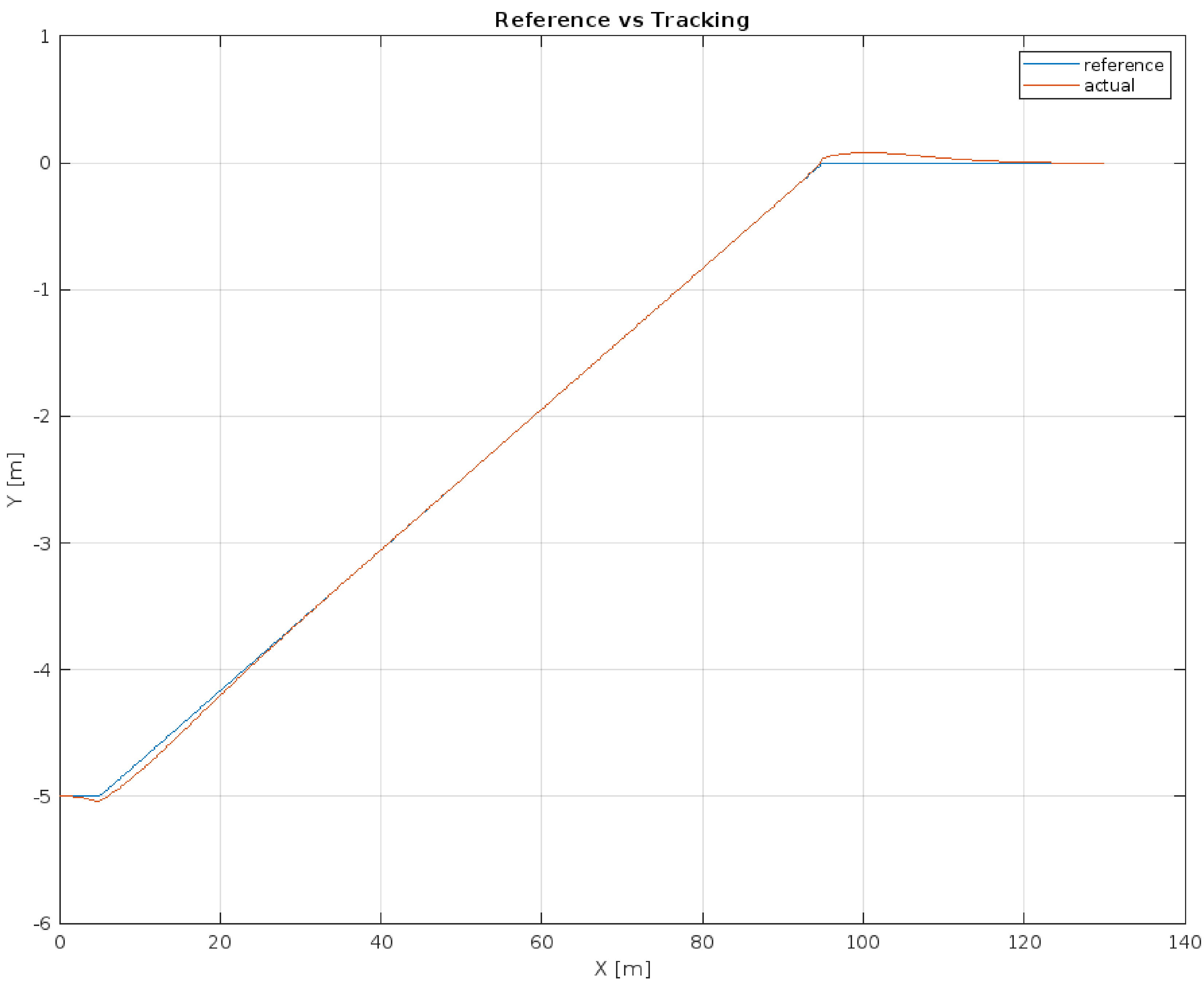
```
% Model Parameters
m = 1573;
Iz = 2873;
lf = 1.10;
lr = 1.58;
Cf = 8e4;
Cr = 8e4;
Vx = 30;
% System Matrix
A = 2*[ 0, 1/2, 0, 0;
0, -(Cf+Cr)/(m*Vx), (Cf+Cr)/m, (-Cf*lf+Cr*lr)/(m*Vx);
0, 0, 0, 1/2;
0, -(Cf*lf-Cr*lr)/(Iz*Vx), (Cf*lf-Cr*lr)/Iz, -(Cf*lf^2+Cr*lr^2)/(Iz*Vx)];
%Control Matrix
B1 = [0;
2*Cf/m;
0;
2*Cf*lf/Iz;
];
%Feed-Forward Matrix
B2 = [0;
-2*(Cf*lf-Cr*lr)/(m*Vx) - Vx;
0;
-2*(Cf*lf^2+Cr*lr^2)/(Iz*Vx)
];
```

Question 2.2 - DBM Lane Change using LQR

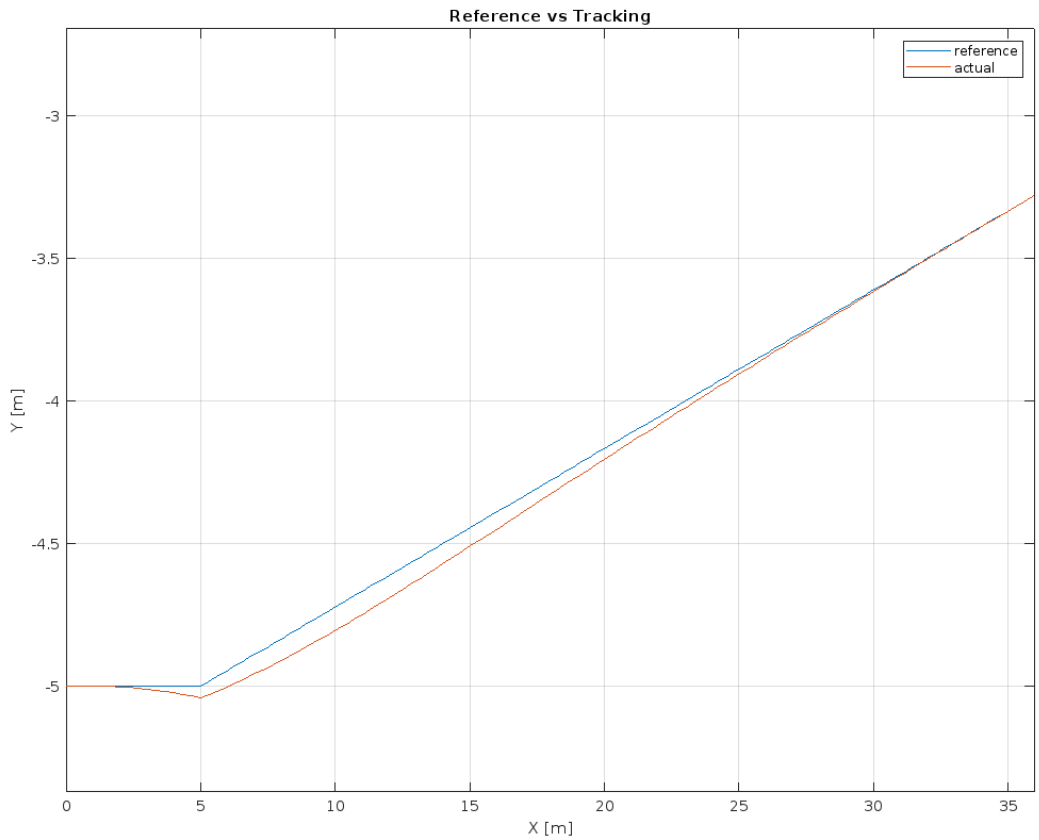



Heading Error
Lane Exit
1 sec from Exit
Lane Entry
1 sec from Entry
Settling Boun
Settling Boun
Max Boun
Φ_{err} [rad]
t [s]
δ_f' [rad/s]
0.01
0.008
0.006
0.004
0.002
0
-0.002
-0.004
-0.006
-0.008
-0.01
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
10
8
6
4
2
0
-2
-4
-6
-8
-10

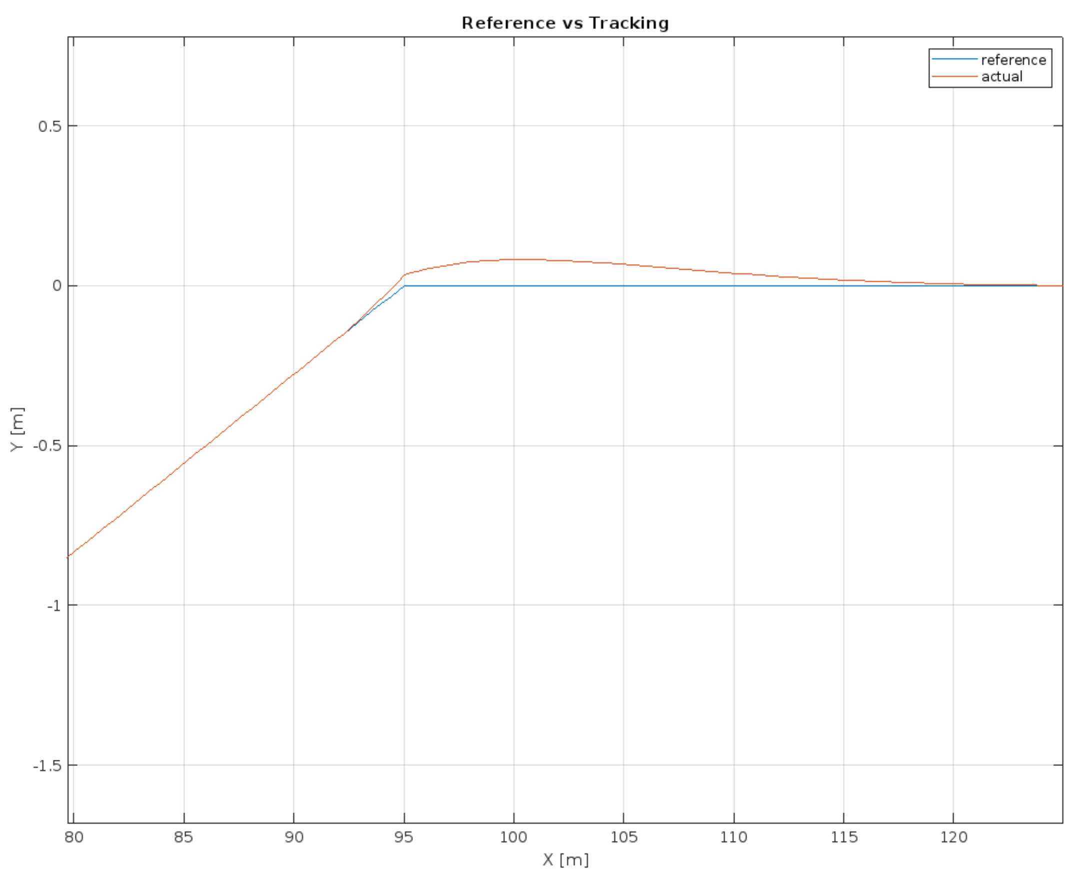
Question 2.3



Reference vs Tracking
reference
actual
Y [m]
X [m]
1
0
-1
-2
-3
-4
-5
-6
0
20
40
60
80
100
120
140



Reference vs Tracking
reference
actual
-3
-3.5
-4
-4.5
-5
0
5
10
15
20
25
30
35
Y [m]
X [m]



Reference vs Tracking
reference
actual
0.5
0
-0.5
-1
-1.5
80
85
90
95
100
105
110
115
120
Y [m]
X [m]

```
Max Abs. Lateral Error: 0.081435 m
Max Abs. Heading Error: 0.009063 rad
Max Steering Rate: 8.884825 rad/s
>>
```

Error Stats

Poles obtained from LQR, LQR Weights and LQR Gain

Poles obtained from LQR:

P =

1.0e+02 *

-0.0411 + 0.0307i
-0.0411 - 0.0307i
-0.0605 + 0.0000i
-1.1127 + 0.0000i

State Weight Matrix:

Q =

7.5000 0 0 0
0 0.4200 0 0
0 0 25.0000 0
0 0 0 17.0000

Control Penalty Weight:

R =

5.5000

LQR Gain:

K =

1.1677 0.2031 9.1127 1.4891

Code

```
close all;
clear all;
clc;
% Model Parameters
m = 1573;
Iz = 2873;
lf = 1.10;
lr = 1.58;
Cf = 8e4;
Cr = 8e4;
Vx = 30;
% System Matrix
A = 2*[ 0, 1/2, 0, 0;
0, -(Cf+Cr)/(m*Vx), (Cf+Cr)/m, (-Cf*lf+Cr*lr)/(m*Vx);
0, 0, 0, 1/2;
0, -(Cf*lf-Cr*lr)/(Iz*Vx), (Cf*lf-Cr*lr)/Iz, -(Cf*lf^2+Cr*lr^2)/(Iz*Vx)];
%Control Matrix
B1 = [0;
2*Cf/m;
0;
2*Cf*lf/Iz;
];
%Feed-Forward Matrix
B2 = [0;
-2*(Cf*lf-Cr*lr)/(m*Vx) - Vx;
0;
-2*(Cf*lf^2+Cr*lr^2)/(Iz*Vx)
];
%%
% Reference Path
strt_seg_len1_x = 5.0;
strt_seg_len2_x = 35.0;
slant_seg_len_x = 90;
total_path_len = strt_seg_len2_x+strt_seg_len1_x+slant_seg_len_x;
slant_seg_len_y = 5.0;
str_seg1_y = -5.0;
str_seg2_y = 0.0;
slant_ang = atan2(slant_seg_len_y,slant_seg_len_x);
slant_seg_len = norm([slant_seg_len_x,slant_seg_len_y]);
%% Time Parametrization
dt = 0.01;
tf = (slant_seg_len + strt_seg_len2_x + strt_seg_len1_x)/Vx;
t_strt1 = strt_seg_len1_x/Vx;
t_slant = t_strt1+slant_seg_len/Vx;
ts1 = linspace(0,t_strt1,floor(t_strt1/dt));
ts2 = linspace(t_strt1+dt,t_slant,floor((t_slant-t_strt1)/dt));
```

```
ts3 = linspace(t_slant+dt,tf,floor((tf-t_slant)/dt));
ts = [ts1, ts2, ts3];
dphi_wind = 10*dt;
x_ref = zeros(1,length(ts));
y_ref = zeros(1,length(ts));
dy_ref = zeros(1,length(ts));
phi_ref = zeros(1,length(ts));
dphi_ref = zeros(1,length(ts));
X_ref = zeros(4,length(ts));
x_act = zeros(1,length(ts));
x_act(1) = 0;
x_ref(1:length(ts1)) = linspace(0,strt_seg_len1_x,length(ts1));
x_ref(length(ts1)+1:length(ts1)+length(ts2)) = strt_seg_len1_x +
linspace(Vx*dt*cos(slant_ang),slant_seg_len_x, length(ts2));
x_ref(length(ts1)+length(ts2)+1:length(ts)) =
linspace(Vx*dt+strt_seg_len1_x+slant_seg_len_x,strt_seg_len1_x+slant_seg_len_x
+strt_seg_len2_x,length(ts3));
y_ref(1:length(ts1)) = str_seg1_y;
y_ref(length(ts1)+1:length(ts1)+length(ts2)) =
linspace(Vx*dt*sin(slant_ang)+str_seg1_y,str_seg2_y, length(ts2));
y_ref(length(ts1)+length(ts2)+1:length(ts)) = str_seg2_y;
dy_ref(1:length(ts1)) = 0.0;
dy_ref(length(ts1)+1:length(ts1)+length(ts2)) = Vx*sin(slant_ang);
dy_ref(length(ts1)+length(ts2)+1:length(ts)) = 0.0;
dphi1_start = floor((t_strt1-dphi_wind)/dt);
dphi1_end = floor((t_strt1)/dt);
dphi2_start = floor((t_slant-dphi_wind)/dt);
dphi2_end = floor((t_slant)/dt);
dphi_ref(dphi1_start:dphi1_end) = slant_ang/dphi_wind;
dphi_ref(dphi2_start:dphi2_end) = -slant_ang/dphi_wind;
% for i=1:length(dphi_ref)-1
% dphi_ref(i) = (phi_ref(i+1)-phi_ref(i))/dt;
% end
%
% dphi_ref(length(dphi_ref)) = dphi_ref(length(dphi_ref)-1);
% for i=1:length(phi_ref)-1
% phi_ref(i) =
atan2(path_ref(2,i+1)-path_ref(2,i),path_ref(1,i+1)-path_ref(1,i));
% end
%
for i=2:length(phi_ref)
phi_ref(i) = phi_ref(i-1) + dphi_ref(i-1)*dt;
end
% phi_ref(length(phi_ref)) = phi_ref(length(phi_ref)-1);
% dphi_ref(dphi1_start:floor((dphi1_end-dphi1_start)/2)) =
2*slant_ang/dphi_wind*linspace(0,1,floor((dphi1_end-dphi1_start)/2)+1);
% dphi_ref(floor((dphi1_end-dphi1_start)/2):dphi1_end) =
2*slant_ang/dphi_wind*linspace(1,0,floor((dphi1_end-dphi1_start)/2)+1);
% dphi_ref(dphi2_start:floor((dphi2_end-dphi2_start)/2)) =
-2*slant_ang/dphi_wind*linspace(0,1,floor((dphi2_end-dphi2_start)/2)+1);
```

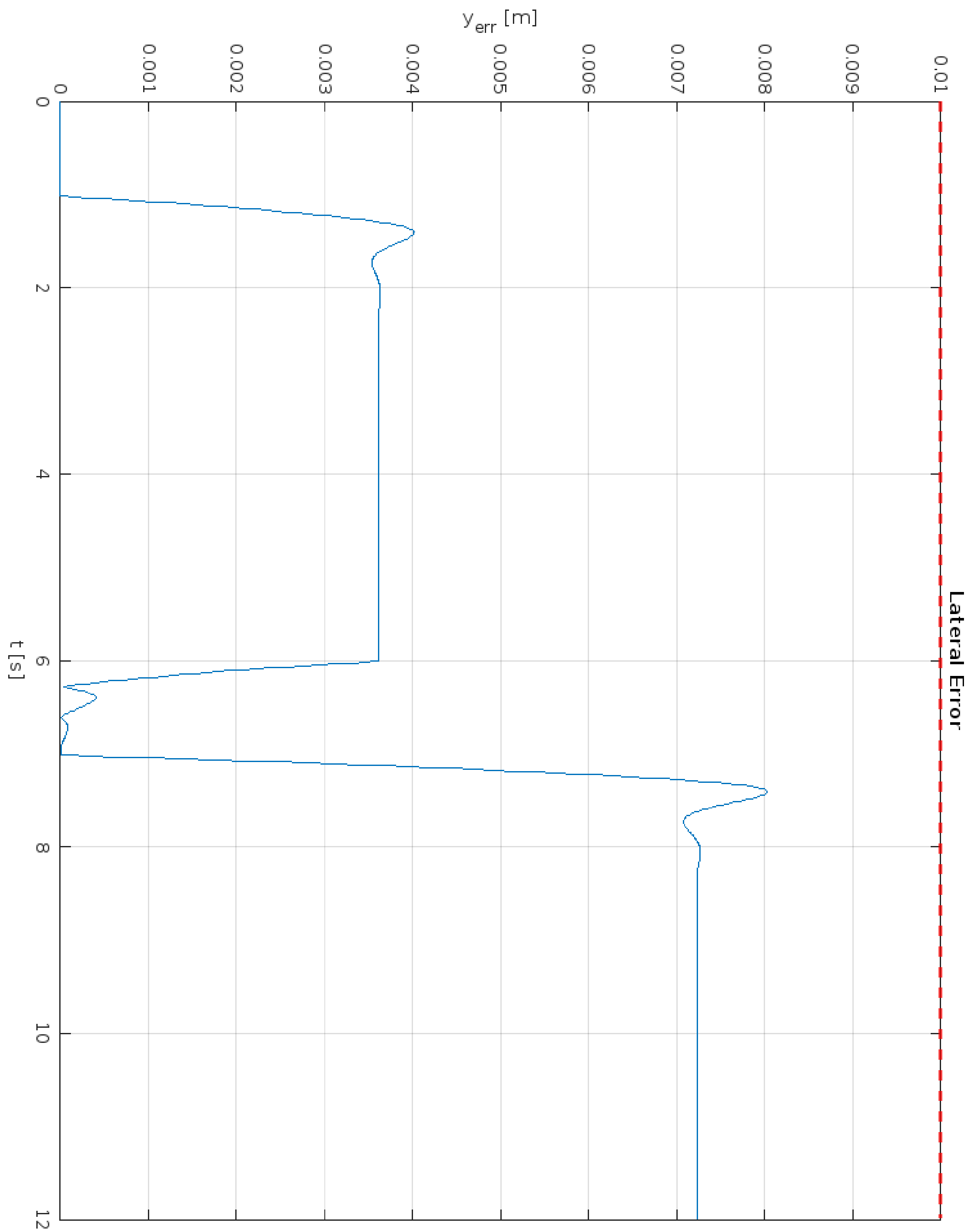
```
% dphi_ref(floor((dphi2_end-dphi2_start)/2):dphi2_end) =
-2*slant_ang/dphi_wind*linspace(1,0,floor((dphi2_end-dphi2_start)/2)+1);
X_ref(1,:) = y_ref;
X_ref(2,:) = dy_ref;
X_ref(3,:) = phi_ref;
X_ref(4,:) = dphi_ref;
%% Controller
Q = eye(4);
R = 5.5;
Q(1,1) = 7.50;
Q(2,2) = 0.42;
Q(3,3) = 25;
Q(4,4) = 17.0;
[K,S,P] = lqr(A,B1,Q,R);
% K = place(A,B1,[-20,-5.20+2.35i,-5.2-2.35i,-55]);
%% Simulation 1
X0 = [str_seg1_y,0,0,0]';
E = [0,0,0,0]';
traj_act = zeros(2, length(ts));
E_log = zeros(4,length(ts));
U_log = zeros(1,length(ts));
E_log(:,1) = [0,0,0,0]';
traj_act(1,1) = 0;
traj_act(2,1) = str_seg1_y;
Edot = zeros(4,1);
for i=2:length(ts)
curr_yaw = (phi_ref(i)+E(3));
traj_act(1,i) = x_ref(i) - E(1)*sin(curr_yaw) - E(2)*sin(curr_yaw)*dt;
traj_act(2,i) = y_ref(i) + E(1)*cos(curr_yaw) + E(2)*cos(curr_yaw)*dt;
E_log(:,i) = E;
U_log(i) = -K*E;
Edot = A*E-B1*K*E+B2*dphi_ref(i);
E = E + Edot*dt;
end
% for i = 2:length(ts)
% x_act(i) = x_act(i-1)+Vx*cos(X_log(3,i-1))*dt;
% end
fprintf('Max Abs. Lateral Error: %f m\n',max(abs(E_log(1,:))))
fprintf('Max Abs. Heading Error: %f rad\n',max(abs(E_log(3,:))))
fprintf('Max Steering Rate: %f rad/s\n',max(diff(U_log)/dt))
% fprintf('Max Abs. Lateral Error: %f m\n',max(abs(X(1,:)'-y_ref)))
% fprintf('Max Abs. Heading Error: %f rad\n',max(abs(X(3,:)'-phi_ref)))
% fprintf('Max Steering Rate: %f rad/s\n',max(diff(-K*X')/dt))
%% Plotting
figure;
plot(x_ref, y_ref)
hold on;
grid on;
plot(traj_act(1,:), traj_act(2,:));
title('Reference vs Tracking')
```

```
xlabel('X [m]')
ylabel('Y [m]')
legend('reference', 'actual')
figure;
plot(ts, E_log(1,:));
grid on;
title('Lateral Error')
xlabel('t [s]')
ylabel('y_{err} [rad]')
% legend('reference', 'actual')
xline(t_strt1,'--','Lane Exit','LineWidth',1.50,'Color','b')
xline(t_strt1+1.0,'--','1 sec from Exit','LineWidth',1.50,'Color','b')
xline(t_slant,'--','Lane Entry','LineWidth',1.50,'Color','b')
xline(t_slant+1.0,'--','1 sec from Entry','LineWidth',1.50,'Color','b')
yline(0.002,'--','Settling Bound','LineWidth',1.5,'Color','g')
yline(-0.002,'--','Settling Bound','LineWidth',1.5,'Color','g')
% yline(0.01,'--','Bound','LineWidth',2.0,'Color','r')
figure;
plot(ts, U_log)
title('Steering Angle')
xlabel('t [s]')
ylabel('\delta_f [rad]')
grid on;
% legend('reference', 'actual')
% yline(25,'--','Bound','LineWidth',2.0,'Color','r')
figure
plot(ts(2:length(ts)),diff(U_log)/dt)
hold on;
grid on;
title('Steering Rate')
xlabel('t [s]')
ylabel('\delta_f'' [rad/s]')
% legend('reference', 'actual')
% yline(25,'--','Bound','LineWidth',1.50,'Color','r')
xline(t_strt1,'--','Lane Exit','LineWidth',1.50,'Color','b')
xline(t_strt1+1.0,'--','1 sec from Exit','LineWidth',1.50,'Color','b')
xline(t_slant,'--','Lane Entry','LineWidth',1.50,'Color','b')
xline(t_slant+1.0,'--','1 sec from Entry','LineWidth',1.50,'Color','b')
figure
% plot(ts, X_log(3,:))
hold on;
% plot(ts,phi_ref)
plot(ts, E_log(3,:));
title('Heading Error')
xlabel('t [s]')
ylabel('\Phi_{err} [rad]')
% legend('reference', 'actual')
xline(t_strt1,'--','Lane Exit','LineWidth',1.50,'Color','b')
xline(t_strt1+1.0,'--','1 sec from Exit','LineWidth',1.50,'Color','b')
xline(t_slant,'--','Lane Entry','LineWidth',1.50,'Color','b')
```

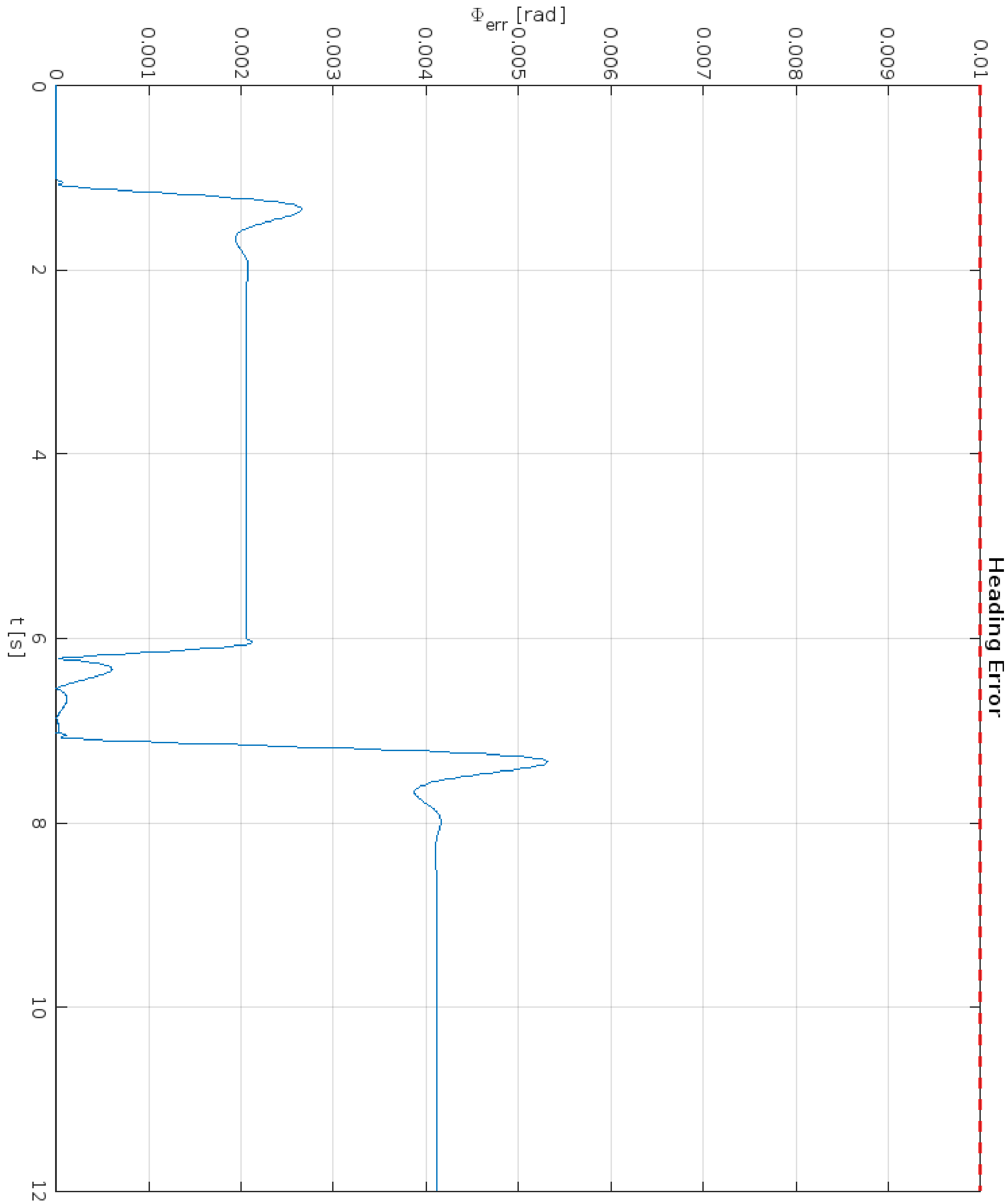
```
xline(t_slant+1.0,'--','1 sec from Entry','LineWidth',1.50,'Color','b')
yline(0.01,'--','Max Bound','LineWidth',1.5,'Color','r')
yline(-0.01,'--','Max Bound','LineWidth',1.5,'Color','r')
yline(0.0007,'--','Settling Bound','LineWidth',1.5,'Color','g')
yline(-0.0007,'--','Settling Bound','LineWidth',1.5,'Color','g')
% max(abs(phi_ref-X_log(3,:)))
grid on;
% % plot(ts, X_log(1,:));
% figure;
%
% % plot(ts,x_ref)
% % subplot(3,1,1);
% plot(x_ref,y_ref);
% hold on;
% plot(x_act, X_log(1,:))
% grid on;
% title('Reference vs Actual Path')
% xlabel('x [m]')
% ylabel('y [m]')
% legend('reference', 'actual')
% yline(0.01,'--','Bound','LineWidth',2.0,'Color','r')
%% Simulation 2
% [r, tout, sv] = lsim(ss(A-B1*K,B2,C,0), dphi_ref,
linspace(0,tf,length(dphi_ref)), X);
function U = control(X,B1,K,B2,dphi_des)
U = -B1*K*X + B2*dphi_des;
end
```

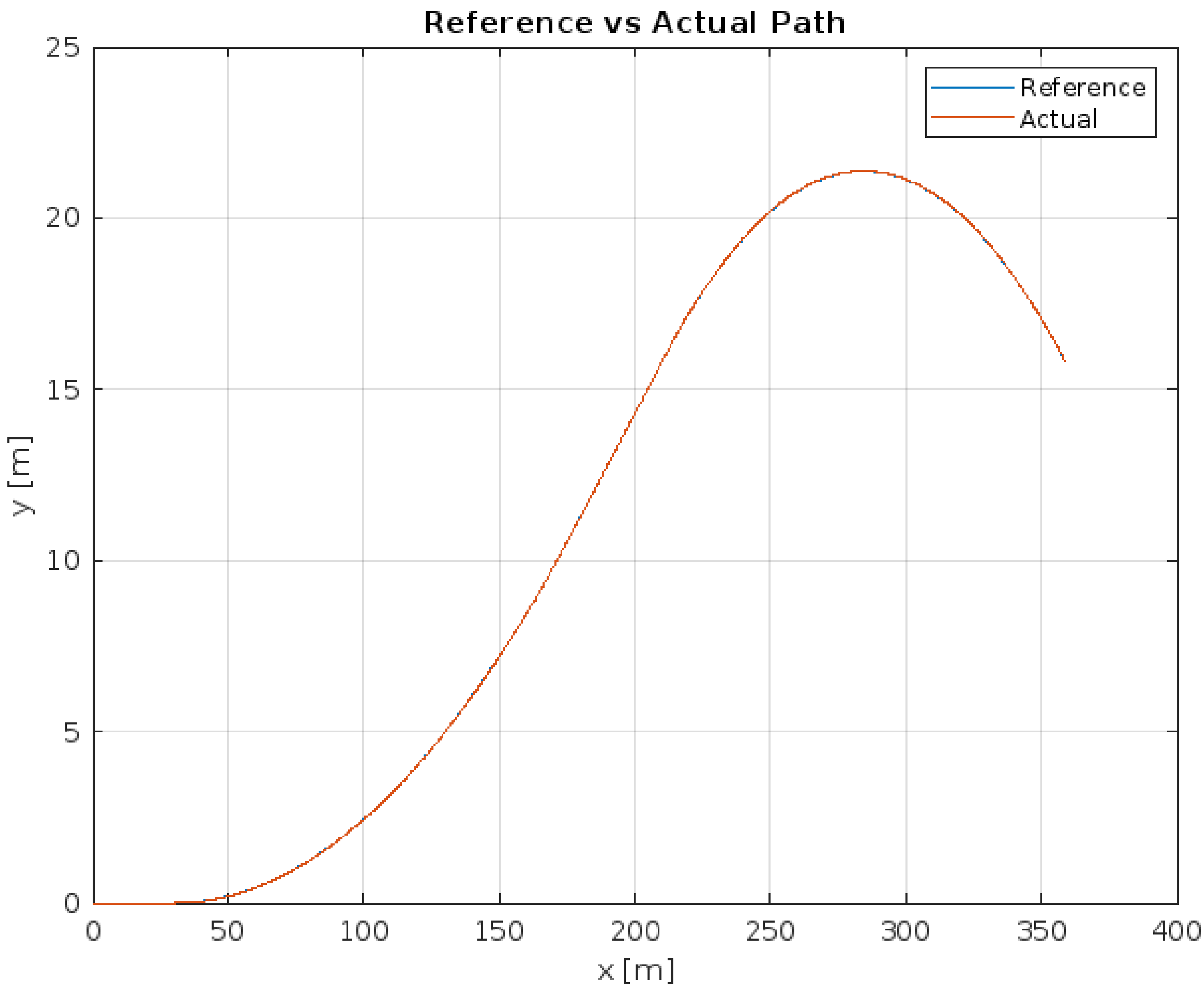
Q2.4 DBM Curve Tracking

Absolute Lateral Error



Absolute Heading Error




Reference vs Actual Path
Reference
Actual
y [m]
x [m]
0
5
10
15
20
25
0
50
100
150
200
250
300
350
400

Error Stats

```
Max Abs. Lateral Error: 0.008028 m
Max Abs. Heading Error: 0.005315 rad
Max Steering Rate: 0.220246 rad/s
>>
```

Poles obtained from LQR, LQR Weights and LQR Gain

Poles from LQR:

```
P =

-5.3027 + 9.1697i
-5.3027 - 9.1697i
-17.8412 +14.3428i
-17.8412 -14.3428i
```

State Weight Matrix

```
Q =

15.0000 0 0 0
0 0.0050 0 0
0 0 30.0000 0
0 0 0 0.0050
```

Control Penalty Weight:

```
R =

1
```

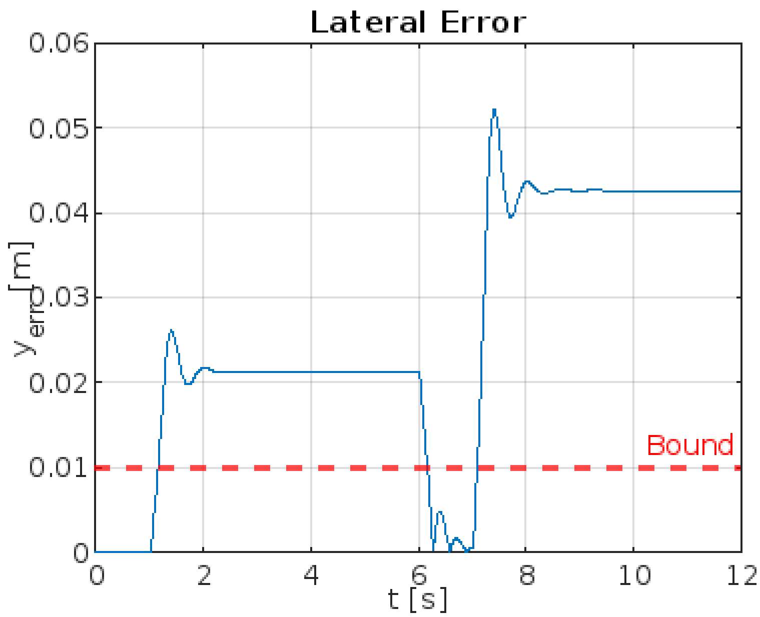
LQR Gain:

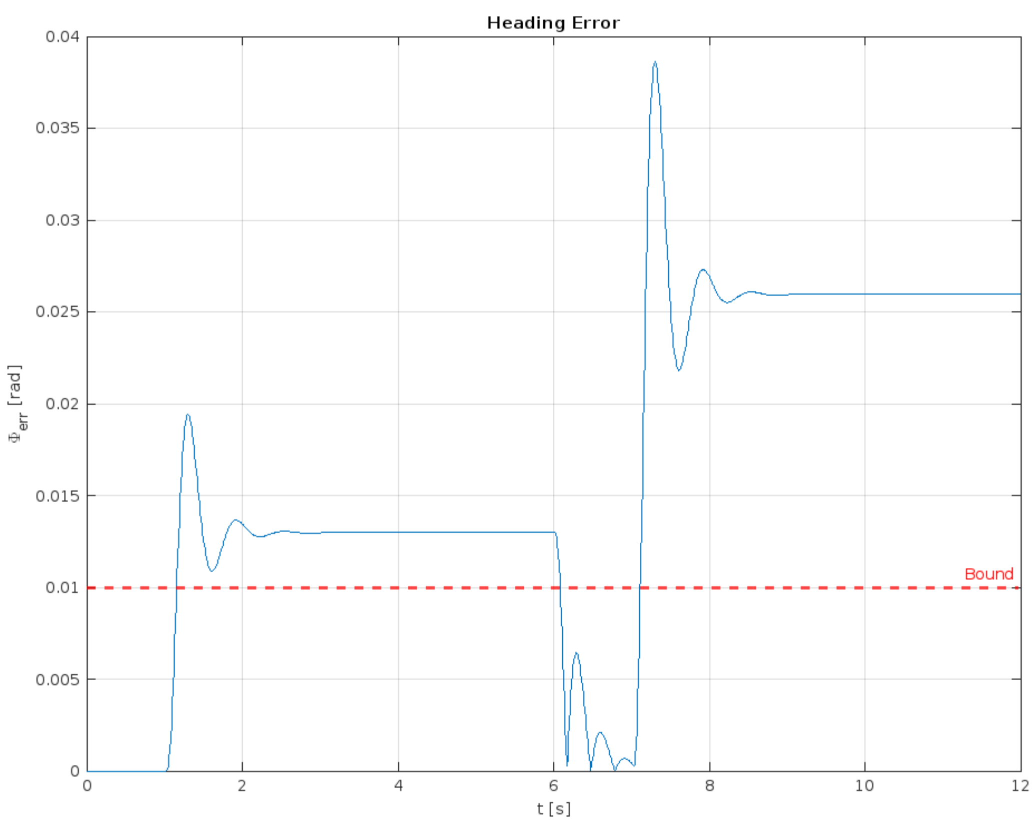
```
K =

3.8730 0.2679 4.7365 0.0877
```

Question 2.6 Vx = 60

Note: Error Plots are Absolute values

```
Max Abs. Lateral Error: 0.052119 m
Max Abs. Heading Error: 0.038643 rad
Max Steering Rate: 1.062970 rad/s
>> |
```

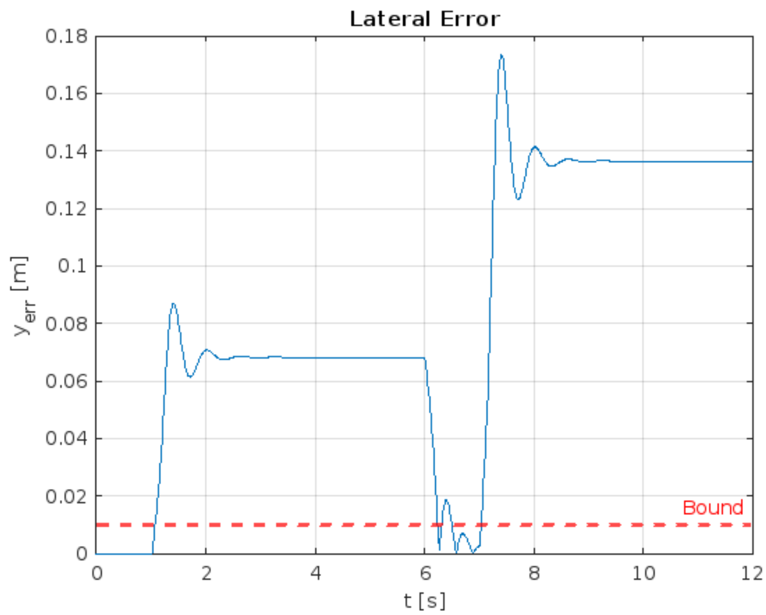


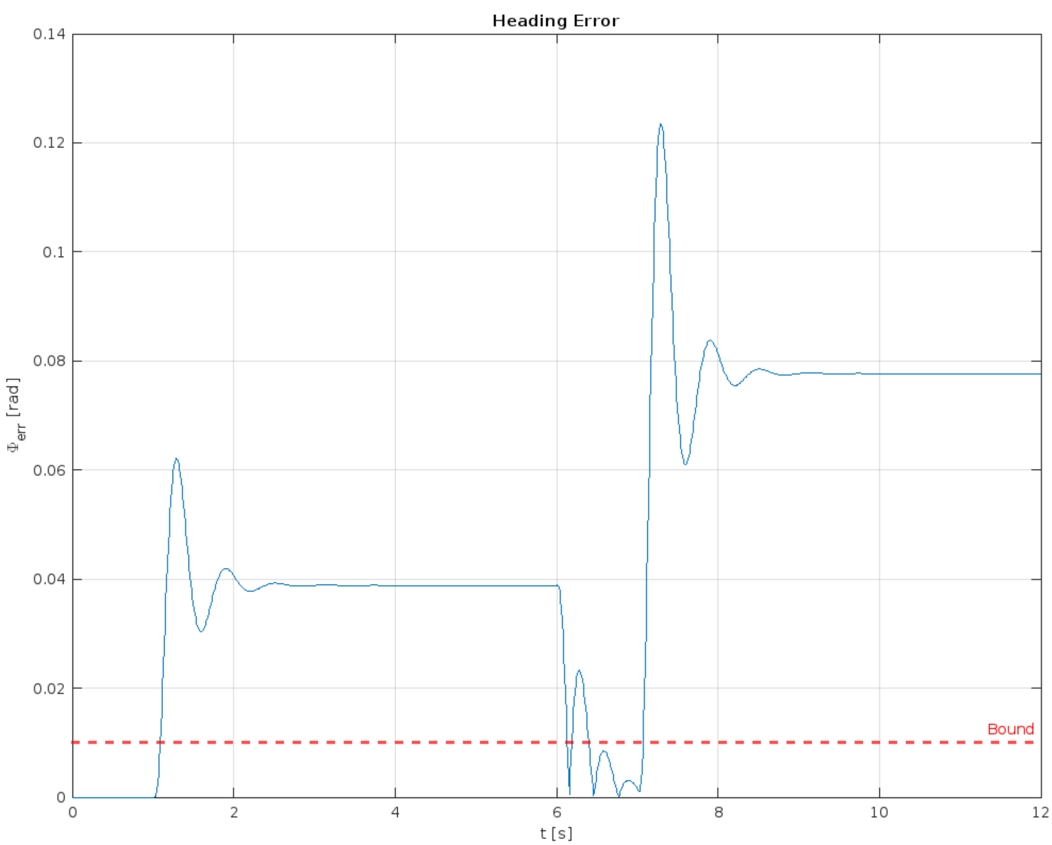


Question 2.6 Vx = 100

Note: Error Plots are Absolute values

```
Max Abs. Lateral Error: 0.173252 m
Max Abs. Heading Error: 0.123584 rad
Max Steering Rate: 3.338304 rad/s
>>
```





Explanation / Observation

We observe that upon increasing the longitudinal velocity of the vehicle (Vx), the Lateral and Heading errors also appear to increase.

Code

```
close all;
clear all;
clc;
% Model Parameters
m = 1573;
Iz = 2873;
lf = 1.10;
lr = 1.58;
Cf = 8e4;
Cr = 8e4;
Vx = 30;
% System Matrix
A = 2*[ 0, 1/2, 0, 0;
0, -(Cf+Cr)/(m*Vx), (Cf+Cr)/m, (-Cf*lf+Cr*lr)/(m*Vx);
0, 0, 0, 1/2;
0, -(Cf*lf-Cr*lr)/(Iz*Vx), (Cf*lf-Cr*lr)/Iz, -(Cf*lf^2+Cr*lr^2)/(Iz*Vx)];
%Control Matrix
B1 = [0;
2*Cf/m;
0;
2*Cf*lf/Iz;
];
%Feed-Forward Matrix
B2 = [0;
-2*(Cf*lf-Cr*lr)/(m*Vx) - Vx;
0;
-2*(Cf*lf^2+Cr*lr^2)/(Iz*Vx)
];
% Measurement Matrix
C = [1 0 0 0;
0 0 1 0];
%%
% Reference Path
strt_seg1_dur = 1.0;
strt_seg1_len = Vx*strt_seg1_dur;
strt_seg2_dur = 1.0;
strt_seg2_len = Vx*strt_seg2_dur;
```

```
circ_seg1_rad = 1000;
circ_seg1_dur = 5.0;
circ_seg1_dir = 1;
circ_seg2_rad = 500;
circ_seg2_dur = 5.0;
circ_seg2_dir = -1;
init_pos = [0,0]';
init_phi = 0.0;
%% Time Parametrization
dt = 0.01;
tf = strt_seg1_dur+strt_seg2_dur+circ_seg1_dur+circ_seg2_dur;
t_strt1 = strt_seg1_dur;
t_circ1 = t_strt1 + circ_seg1_dur;
t_strt2 = t_circ1 + strt_seg2_dur;
t_circ2 = t_strt2 + circ_seg2_dur;
ts1 = linspace(0,t_strt1,floor(t_strt1/dt));
ts2 = linspace(t_strt1+dt,t_circ1,floor((circ_seg1_dur)/dt));
ts3 = linspace(t_circ1+dt,t_strt2,floor((strt_seg2_dur)/dt));
ts4 = linspace(t_strt2+dt,t_circ2,floor((circ_seg2_dur)/dt));
ts = [ts1, ts2, ts3, ts4];
dy_ref = zeros(1,length(ts));
phi_ref = zeros(1,length(ts));
dphi_ref = zeros(1,length(ts));
X_ref = zeros(4,length(ts));
path_ref = zeros(2,length(ts));
%straight segment 1
path_ref(:,1:length(ts1)) = init_pos + [Vx*cos(init_phi),
Vx*sin(init_phi)]'*ts1;
dy_ref(1:length(ts1)) = Vx*sin(init_phi);
%Circular segment 1
circ1_end_phi = init_phi+Vx*circ_seg1_dur/circ_seg1_rad;
theta1 = init_phi+Vx*linspace(0, circ_seg1_dur, length(ts2))/circ_seg1_rad;
circ1_pts = circ_seg1_rad*[cos(theta1-pi/2); sin(theta1-pi/2)];
circ1_trans = path_ref(:,length(ts1)) - circ1_pts(:,1);
circ1_pts = circ1_pts + circ1_trans + [Vx*cos(init_phi),
Vx*sin(init_phi)]'*dt;
path_ref(:,length(ts1)+1:length(ts1)+length(ts2)) = circ1_pts;
dy_ref(length(ts1)+1:length(ts1)+length(ts2)) = Vx*sin(theta1);
%Straight Segment 2
path_ref(:,length(ts1)+length(ts2)+1:length(ts1)+length(ts2)+length(ts3)) =
path_ref(:,length(ts1)+length(ts2)) + [Vx*cos(circ1_end_phi),
Vx*sin(circ1_end_phi)]'*(ts3-t_circ1);
dy_ref(length(ts1)+length(ts2)+1:length(ts1)+length(ts2)+length(ts3)) =
Vx*sin(circ1_end_phi);
%Circular segment 2
circ2_end_phi = circ1_end_phi-Vx*circ_seg2_dur/circ_seg2_rad;
theta2 = circ1_end_phi - Vx*linspace(0, circ_seg2_dur,
length(ts4))/circ_seg2_rad;
circ2_pts = circ_seg2_rad*[cos(theta2+pi/2); sin(theta2+pi/2)];
```

```
circ2_trans =
path_ref(:,length(ts3)+length(ts2)+length(ts1))+[Vx*cos(circ1_end_phi),
Vx*sin(circ1_end_phi)]'*dt - circ2_pts(:,1);
circ2_pts = circ2_pts + circ2_trans;
path_ref(:,length(ts1)+length(ts2)+length(ts3)+1:length(ts1)+length(ts2)+lengt
h(ts3)+length(ts4)) = circ2_pts;
dy_ref(length(ts1)+length(ts2)+length(ts3)+1:length(ts1)+length(ts2)+length(ts
3)+length(ts4)) = Vx*sin(theta2);
% plot(path_ref(1,:), path_ref(2,:),'.')
for i=1:length(phi_ref)-1
phi_ref(i) =
atan2(path_ref(2,i+1)-path_ref(2,i),path_ref(1,i+1)-path_ref(1,i));
end
phi_ref(length(phi_ref)) = phi_ref(length(phi_ref)-1);
for i=1:length(dphi_ref)-1
dphi_ref(i) = (phi_ref(i+1)-phi_ref(i))/dt;
end
dphi_ref(length(dphi_ref)-1) = dphi_ref(length(dphi_ref)-2);
dphi_ref(length(dphi_ref)) = dphi_ref(length(dphi_ref)-2);
X_ref(1,:) = path_ref(2,:);
X_ref(2,:) = dy_ref;
X_ref(3,:) = phi_ref;
X_ref(4,:) = dphi_ref;
%% Controller
Q = eye(4);
R = 1.0;
Q(1,1) = 15;
Q(2,2) = 0.005;
Q(3,3) = 30;
Q(4,4) = 0.005;
[K,S,P] = lqr(A,B1,Q,R);
%% Simulation 1
X0 = [init_pos(1),init_pos(2),init_phi,0]';
X = [0,0,0,0]';
E = [0,0,0,0]';
X_log = zeros(4,length(ts));
E_log = zeros(4,length(ts));
U_log = zeros(1,length(ts));
Edot = zeros(4,1);
traj_act = zeros(2, length(ts));
for i=1:length(ts)
X_log(:,i) = X_ref(:,i)+E;
curr_yaw = (phi_ref(i)+E(3));
traj_act(1,i) = path_ref(1,i) - E(1)*sin(curr_yaw) - E(2)*sin(curr_yaw)*dt;
traj_act(2,i) = path_ref(2,i) + E(1)*cos(curr_yaw) + E(2)*cos(curr_yaw)*dt;
E_log(:,i) = E;
U_log(i) = -K*E;
Edot = A*E-B1*K*E+B2*dphi_ref(i);
E = E + Edot*dt;
```

```
end
fprintf('Max Abs. Lateral Error: %f m\n',max(abs(E_log(1,:))))
fprintf('Max Abs. Heading Error: %f rad\n',max(abs(E_log(3,:))))
fprintf('Max Steering Rate: %f rad/s\n',max(diff(U_log)/dt))
%% Plotting
figure;
plot(ts, abs(E_log(1,:)));
title('Lateral Error')
xlabel('t [s]')
ylabel('y_{err} [m]')
yline(0.01,'--','Bound','LineWidth',2.0,'Color','r')
grid on;
figure;
plot(ts, U_log)
title('Steering Angle')
xlabel('t [s]')
ylabel('\delta_f [rad]')
grid on;
figure;
plot(ts(1:length(ts)-1),diff(U_log)/dt)
title('Steering Rate')
xlabel('t [s]')
ylabel('\delta_f'' [rad/s]')
grid on;
grid on;
figure
plot(ts, abs(E_log(3,:)));
grid on;
title('Heading Error')
xlabel('t [s]')
ylabel('\Phi_{err} [rad]')
yline(0.01,'--','Bound','LineWidth',2.0,'Color','r')
figure;
plot(path_ref(1,:),path_ref(2,:));
hold on;
plot(path_ref(1,:), X_log(1,:))
grid on;
title('Reference vs Actual Path')
xlabel('x [m]')
ylabel('y [m]')
legend('Reference', 'Actual')
%% Simulation 2
% [r, tout, sv] = lsim(ss(A-B1*K,B2,C,0), dphi_ref,
linspace(0,tf,length(dphi_ref)), X);
function U = control(X,B1,K,B2,dphi_des)
U = -B1*K*X + B2*dphi_des;
end
```

Question 3

We use the same EoMs developed above for the Pepy model for this problem. Except we also have a longitudinal controller providing us with a longitudinal acceleration which will affect the velocity as follows:

$$\frac{dV}{dt} = a_{lon}$$

$$\frac{dx}{dt} = V cos(\phi)$$

$$\frac{dy}{dt} = V sin(\phi)$$

$$\frac{d\phi}{dt} = \omega = V\frac{tan(\delta_f)}{l_f + l_r}$$

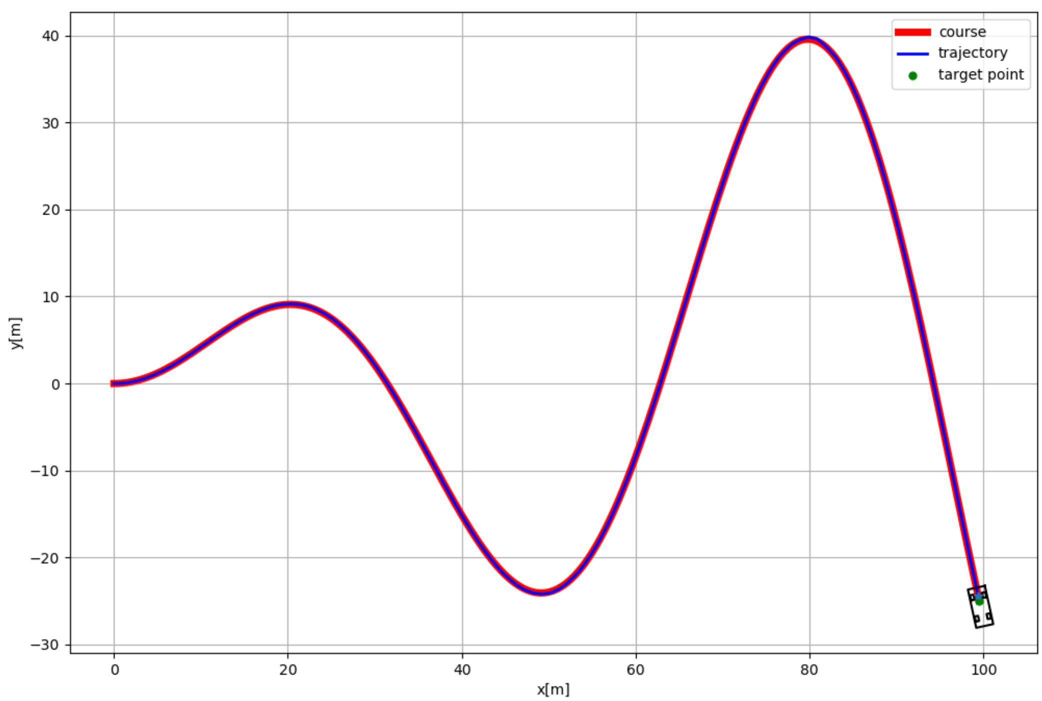
Lookahead angle: η (angle made by the lookahead vector with the global X-axis)
Heading angle: ϕ
Yaw Error: α

- Then $\eta = atan(\frac{y_t - y_v}{x_t - x_v})$, where (x_t, y_t) is the coordinate of the target point, (x_v, y_v) corresponds to the coordinates of the vehicle (both in global frame)
- Then $\alpha = \eta - \phi$ is the expression to obtain the yaw-error.
- Once we obtain the yaw-error we use,
- $x_{tB} = Lsin(\alpha)$, projection of the x_t on the body axes,
- And from the formulation of Pure-Pursuit controller, we have
- $\frac{1}{R} = \frac{2x_{tB}}{L^2}$, substituting the value of x_{tB} from above, we obtain,
- $\frac{1}{R} = \frac{2sin(\alpha)}{L}$

- But from the Pepy model developed above we have:
- $\frac{1}{R} = \frac{tan(\delta_f)}{l_f + l_r}$
- Equating the two and eliminating R, we obtain
- $\delta_f = atan(\frac{2(l_f + l_r)sin(\alpha)}{L})$

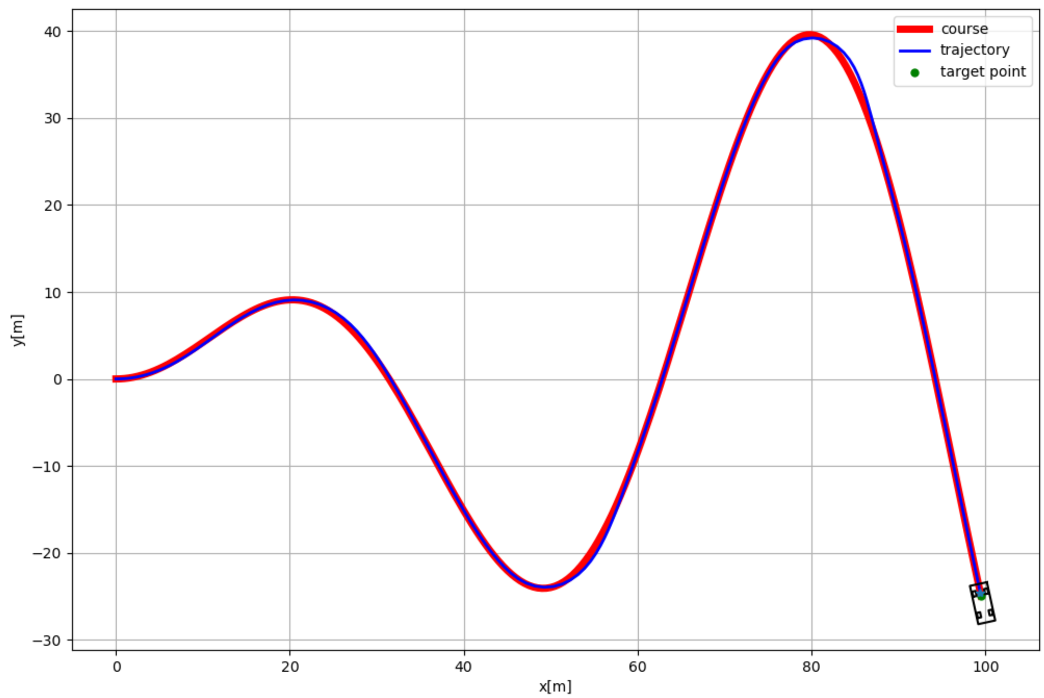
Where, $l_f + l_r$ is the wheelbase of the vehicle (WB).

<u>Question 3.2 Plot for default parameters</u>

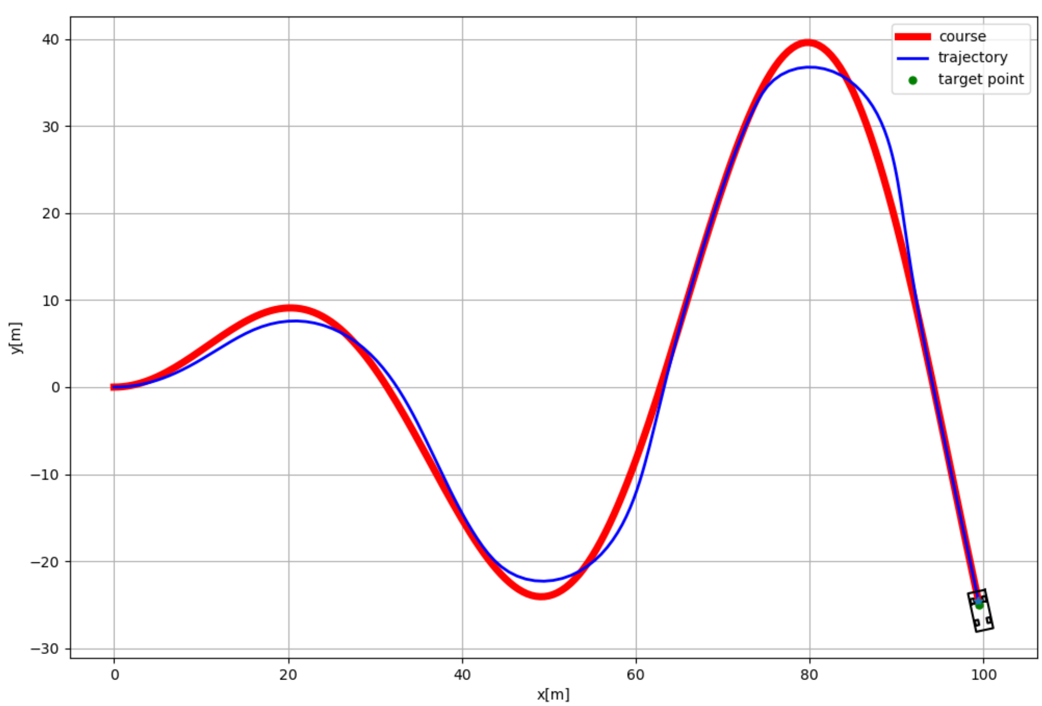


Question 3.3

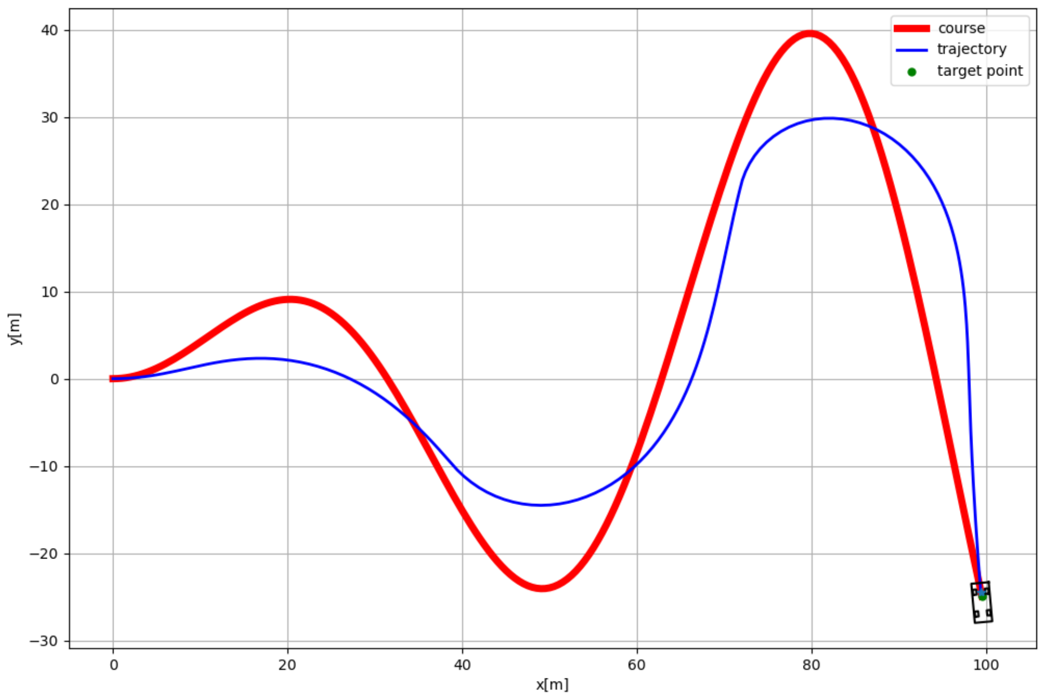
L = 5.0m



L = 10.0m



L = 20.0 m



High Lookahead Values

- Result in a smoother but over-damped response
- This causes the Vehicle to short–cut many sharp turns resulting in a poor tracking performance
- Lower control effort, as steering input is inversely proportional to the lookahead distance

Low Lookahead Values

- Results in a snappier response, good tracking accuracy even in case of sharp turns, corners
- But too low of a value can induce oscillatory response and noise in the controller
- Results in a higher control effort

Remarks

- To always have an optimal tracking performance, one could adapt the look-ahead distance online as a function of the path curvature and the vehicle speed. I have worked on such an adaptive look-ahead based non-linear 3D path controller for UAVs in my Master’s thesis at IIT Madras.

Code

```
"""

Path tracking simulation with pure pursuit steering control and PID speed control for
16-665 .

author: Rathin Shah(rsshah), Shruti Gangopadhyay (sgangopa)

"""


import math
import matplotlib.pyplot as plt
import numpy as np

# Pure Pursuit parameters
L = 1.0  # look ahead distance
dt = 0.1  # discrete time

# Vehicle parameters (m)
LENGTH = 4.5        #length of the vehicle (for the plot)
WIDTH = 2.0         #length of the vehicle (for the plot)
BACKTOWHEEL = 1.0   #length of the vehicle (for the plot)
WHEEL_LEN = 0.3     #length of the vehicle (for the plot)
WHEEL_WIDTH = 0.2   #length of the vehicle (for the plot)
TREAD = 0.7         #length of the vehicle (for the plot)
WB = 2.5            # wheel-base


def plotVehicle(x, y, yaw, steer=0.0, cabcolor="-r", truckcolor="-k"):

    outline = np.array(
        [
            [
                -BACKTOWHEEL,
                (LENGTH - BACKTOWHEEL),
                (LENGTH - BACKTOWHEEL),
                -BACKTOWHEEL,
                -BACKTOWHEEL,
            ],
            [WIDTH / 2, WIDTH / 2, -WIDTH / 2, -WIDTH / 2, WIDTH / 2],
        ]
```

```
)

fr_wheel = np.array(
    [
        [WHEEL_LEN, -WHEEL_LEN, -WHEEL_LEN, WHEEL_LEN, WHEEL_LEN],
        [
            -WHEEL_WIDTH - TREAD,
            -WHEEL_WIDTH - TREAD,
            WHEEL_WIDTH - TREAD,
            WHEEL_WIDTH - TREAD,
            -WHEEL_WIDTH - TREAD,
        ],
    ]
)

rr_wheel = np.copy(fr_wheel)

fl_wheel = np.copy(fr_wheel)
fl_wheel[1, :] *= -1
rl_wheel = np.copy(rr_wheel)
rl_wheel[1, :] *= -1

Rot1 = np.array([[math.cos(yaw), math.sin(yaw)], [-math.sin(yaw), math.cos(yaw)]])
Rot2 = np.array(
    [[math.cos(steer), math.sin(steer)], [-math.sin(steer), math.cos(steer)]]
)

fr_wheel = (fr_wheel.T.dot(Rot2)).T
fl_wheel = (fl_wheel.T.dot(Rot2)).T
fr_wheel[0, :] += WB
fl_wheel[0, :] += WB

fr_wheel = (fr_wheel.T.dot(Rot1)).T
fl_wheel = (fl_wheel.T.dot(Rot1)).T

outline = (outline.T.dot(Rot1)).T
rr_wheel = (rr_wheel.T.dot(Rot1)).T
rl_wheel = (rl_wheel.T.dot(Rot1)).T

outline[0, :] += x
outline[1, :] += y
fr_wheel[0, :] += x
fr_wheel[1, :] += y
rr_wheel[0, :] += x
```

```
    rr_wheel[1, :] += y
    fl_wheel[0, :] += x
    fl_wheel[1, :] += y
    rl_wheel[0, :] += x
    rl_wheel[1, :] += y

    plt.plot(
        np.array(outline[0, :]).flatten(), np.array(outline[1, :]).flatten(), truckcolor
    )
    plt.plot(
        np.array(fr_wheel[0, :]).flatten(),
        np.array(fr_wheel[1, :]).flatten(),
        truckcolor,
    )
    plt.plot(
        np.array(rr_wheel[0, :]).flatten(),
        np.array(rr_wheel[1, :]).flatten(),
        truckcolor,
    )
    plt.plot(
        np.array(fl_wheel[0, :]).flatten(),
        np.array(fl_wheel[1, :]).flatten(),
        truckcolor,
    )
    plt.plot(
        np.array(rl_wheel[0, :]).flatten(),
        np.array(rl_wheel[1, :]).flatten(),
        truckcolor,
    )
    plt.plot(x, y, "*")


def getDistance(p1, p2):
    """
    Calculate distance
    :param p1: list, point1
    :param p2: list, point2
    :return: float, distance
    """
    dx = p1[0] - p2[0]
    dy = p1[1] - p2[1]
    return math.hypot(dx, dy)
```

```python
class Vehicle:
    def __init__(self, x, y, yaw, vel=0):
        """
        Define a vehicle class (state of the vehicle)
        :param x: float, x position
        :param y: float, y position
        :param yaw: float, vehicle heading
        :param vel: float, velocity

        """
        # State of the vehicle

        self.x = x #x coordinate of the vehicle
        self.y = y #y coordinate of the vehicle
        self.yaw = yaw  #yaw of the vehicle
        self.vel = vel  #velocity of the vehicle

    def update(self, acc, delta):
        """
        Vehicle motion model, here we are using simple bycicle model
        :param acc: float, acceleration
        :param delta: float, heading control
        """

        # TODO- update the state of the vehicle (x,y,yaw,vel) based on simple bicycle model
        self.x += self.vel*math.cos(self.yaw)*dt
        self.y += self.vel*math.sin(self.yaw)*dt
        self.yaw += self.vel*math.tan(delta)*dt/(WB)
        self.vel += acc*dt




class Trajectory:
    def __init__(self, traj_x, traj_y):
        """
        Define a trajectory class
        :param traj_x: list, list of x position
        :param traj_y: list, list of y position
        """
        self.traj_x = traj_x
        self.traj_y = traj_y
        self.last_idx = 0
```

```
    def getPoint(self, idx):
        return [self.traj_x[idx], self.traj_y[idx]]

    def getTargetPoint(self, pos):
        """
        Get the next look ahead point
        :param pos: list, vehicle position
        :return: list, target point
        """
        target_idx = self.last_idx
        target_point = self.getPoint(target_idx)
        curr_dist = getDistance(pos, target_point)

        while curr_dist < L and target_idx < len(self.traj_x) - 1:
            target_idx += 1
            target_point = self.getPoint(target_idx)
            curr_dist = getDistance(pos, target_point)

        self.last_idx = target_idx
        return self.getPoint(target_idx)


class Controller:
    def __init__(self, kp=1.0, ki=0.1):
        """
        Define a PID controller class
        :param kp: float, kp coeff
        :param ki: float, ki coeff
        :param kd: float, kd coeff
        """
        self.kp = kp
        self.ki = ki
        self.Pterm = 0.0
        self.Iterm = 0.0
        self.last_error = 0.0

    def Longitudinalcontrol(self, error):
        """
        PID main function, given an input, this function will output a acceleration for
longitudinal error
        :param error: float, error term
        :return: float, output control
        """
        self.Pterm = self.kp * error
```

```
        self.Iterm += error * dt

        self.last_error = error
        output = self.Pterm + self.ki * self.Iterm
        return output

    def PurePursuitcontrol(self, error):
        #TODO- find delta
        delta = math.atan(2*WB*math.sin(error)/L)

        return delta

def main():
    # create vehicle
    ego = Vehicle(0, 0, 0)
    plotVehicle(ego.x, ego.y, ego.yaw)

    # target velocity
    target_vel = 10

    # target course
    traj_x = np.arange(0, 100, 0.5)
    traj_y = [math.sin(x / 10.0) * x / 2.0 for x in traj_x]
    traj = Trajectory(traj_x, traj_y)
    goal = traj.getPoint(len(traj_x) - 1)

    # create longitudinal and pure pursuit controller
    PI_acc = Controller()
    PI_yaw = Controller()

    # real trajectory
    traj_ego_x = []
    traj_ego_y = []

    plt.figure(figsize=(12, 8))

    while getDistance([ego.x, ego.y], goal) > 1:
        target_point = traj.getTargetPoint([ego.x, ego.y])

        # use PID to control the speed vehicle
        vel_err = target_vel - ego.vel
        acc = PI_acc.Longitudinalcontrol(vel_err)

        # use pure pursuit to control the heading of the vehicle
```

```
        # TODO- Calculate the yaw error
        eta = math.atan2(target_point[1]-ego.y,target_point[0]-ego.x)
        yaw_err = eta-ego.yaw   #TODO- Update the equation
        delta = PI_yaw.PurePursuitcontrol(yaw_err)  #TODO- update thr Pure pursuit
controller

        # move the vehicle
        ego.update(acc, delta)

        # store the trajectory
        traj_ego_x.append(ego.x)
        traj_ego_y.append(ego.y)

        # plots
        plt.cla()
        plt.plot(traj_x, traj_y, "-r", linewidth=5, label="course")
        plt.plot(traj_ego_x, traj_ego_y, "-b", linewidth=2, label="trajectory")
        plt.plot(target_point[0], target_point[1], "og", ms=5, label="target point")
        plotVehicle(ego.x, ego.y, ego.yaw, delta)
        plt.xlabel("x[m]")
        plt.ylabel("y[m]")
        plt.axis("equal")
        plt.legend()
        plt.grid(True)
        plt.pause(0.1)


plt.savefig('/home/kyouma/dev/academics/mobility/AD/'+str(target_vel)+'_'+str(L)+'.png')


if __name__ == "__main__":
    main()
```