Project 2- Exercise 1

November 6, 2024

1 Exercise 1

1.1 Question 1

[53]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy import linalg
import control as ct

#Values
m = 1888.6
Iz = 25854
lf = 1.55
lr = 1.39
C_a = 20000
```

[54]:
```
def create_matrices(v):
    A = np.array([
        [0, 1, 0, 0],
        [0, -4*C_a/(m*v), 4*C_a/m, -2*C_a*(lf-lr)/(m*v)],
        [0, 0, 0, 1],
        [0, -2*C_a*(lf-lr)/(Iz*v), 2*C_a*(lf-lr)/Iz, -2*C_a*(lf**2+lr**2)/
 ↪(Iz*v)]
    ])
    B = np.array([
        [0, 0],
        [2*C_a/m, 0],
        [0, 0],
        [2*C_a*lf/Iz, 0]
    ])
    C = np.eye(4)  # Assuming all states are observable
    D = np.zeros((4, 2))

    return ct.ss(A, B, C, D)
```

```
[62]: velocities = [2, 5, 8]
print("RESULTS (Q1)")
for v in velocities:
    sys = create_matrices(v)
    p = ct.ctrb(sys.A, sys.B)
    controllability_rank = np.linalg.matrix_rank(p)
    cond1 = controllability_rank == sys.A.shape[0]
    q = ct.obsv(sys.A, sys.C)
    observability_rank = np.linalg.matrix_rank(q)
    cond2 = observability_rank == sys.A.shape[0]


    print(f"At velocity {v} m/s:")
    print(f"Controllable: {cond1} (rank = {controllability_rank})")
    print(f"Observable: {cond2} (rank = {observability_rank})")
    print(f"Controllability Matrix P: \n {np.array2string(p, precision=3,␣
↪suppress_small=True)}")
    print(f"Observability Matrix Q: \n {np.array2string(q, precision=3,␣
↪suppress_small=True)}")
    print()
```

```
RESULTS (Q1)
At velocity 2 m/s:
Controllable: True (rank = 4)
Observable: True (rank = 4)
Controllability Matrix P:
 [[      0.           0.          21.18         0.        -452.643       0.
     9706.503       0.    ]
 [     21.18         0.        -452.643       0.        9706.503       0.
  -206189.084       0.    ]
 [      0.           0.           2.398       0.         -10.663       0.
       92.371       0.    ]
 [      2.398        0.         -10.663       0.          92.371       0.
    -1513.765       0.    ]]
Observability Matrix Q:
 [[    1.         0.         0.         0.   ]
 [    0.         1.         0.         0.   ]
 [    0.         0.         1.         0.   ]
 [    0.         0.         0.         1.   ]
 [    0.         1.         0.         0.   ]
 [    0.       -21.18      42.359     -1.694]
 [    0.         0.         0.         1.   ]
 [    0.        -0.124      0.248     -3.353]
 [    0.       -21.18      42.359     -1.694]
 [    0.       448.79    -897.58      83.927]
 [    0.        -0.124      0.248     -3.353]
 [    0.         3.036     -6.073     11.701]
 [    0.       448.79    -897.58      83.927]
 [    0.    -9515.626 19031.252 -1939.418]
 [    0.        3.036    -6.073    11.701]
 [    0.      -65.76    131.52    -50.452]]

At velocity 5 m/s:
Controllable: True (rank = 4)
Observable: True (rank = 4)
Controllability Matrix P:
 [[     0.          0.         21.18       0.      -181.057      0.
    1638.369      0.   ]
 [    21.18       0.      -181.057      0.      1638.369      0.
  -14071.089      0.   ]
 [     0.          0.          2.398      0.        -4.265      0.
      15.278      0.   ]
 [     2.398      0.        -4.265      0.        15.278      0.
    -102.661      0.   ]]
Observability Matrix Q:
 [[   1.       0.       0.       0.   ]
 [   0.       1.       0.       0.   ]
 [   0.       0.       1.       0.   ]
 [   0.       0.       0.       1.   ]
 [   0.       1.       0.       0.   ]
 [   0.      -8.472   42.359   -0.678]
 [   0.       0.       0.       1.   ]
 [   0.      -0.05     0.248   -1.341]
 [   0.      -8.472   42.359   -0.678]
 [   0.      71.806 -359.032   49.01 ]
 [   0.      -0.05     0.248   -1.341]
 [   0.       0.486   -2.429    2.08 ]
 [   0.      71.806 -359.032   49.01 ]
 [   0.    -610.762 3053.808 -473.434]
 [   0.       0.486   -2.429    2.08 ]
 [   0.      -4.219   21.095   -5.548]]

At velocity 8 m/s:
Controllable: True (rank = 4)
Observable: True (rank = 4)
Controllability Matrix P:
 [[    0.        0.       21.18      0.     -113.161     0.      701.889
      0.   ]
 [   21.18      0.     -113.161     0.      701.889     0.    -3832.047
      0.   ]
 [    0.        0.        2.398     0.       -2.666     0.        6.33
      0.   ]
 [    2.398     0.       -2.666     0.        6.33      0.      -27.685
      0.   ]]
Observability Matrix Q:
 [[   1.       0.       0.       0.   ]
```

```
[    0.       1.       0.       0.    ]
[    0.       0.       1.       0.    ]
[    0.       0.       0.       1.    ]
[    0.       1.       0.       0.    ]
[    0.      -5.295   42.359   -0.424]
[    0.       0.       0.       1.    ]
[    0.      -0.031    0.248   -0.838]
[    0.      -5.295   42.359   -0.424]
[    0.      28.049 -224.395   44.957]
[    0.      -0.031    0.248   -0.838]
[    0.       0.19    -1.518    0.963]
[    0.      28.049 -224.395   44.957]
[    0.    -149.91  1199.284 -273.964]
[    0.       0.19    -1.518    0.963]
[    0.      -1.035    8.277   -2.406]]
```

1.2 Question 2

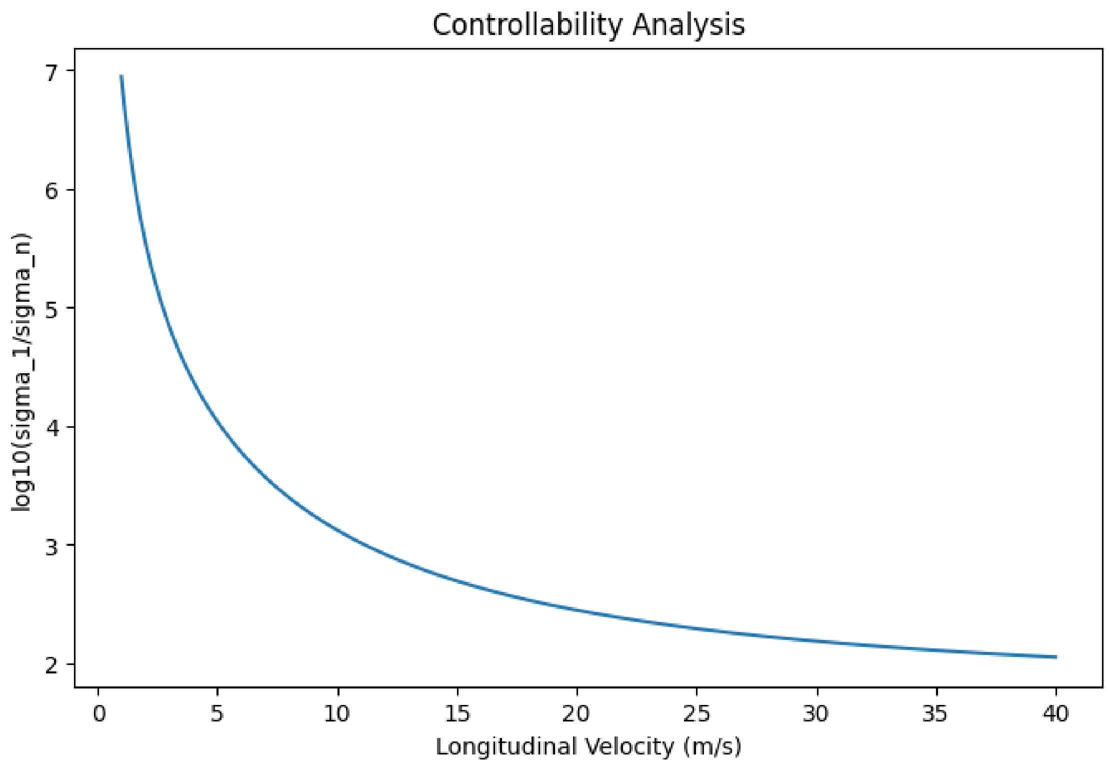
```
[ ]: def create_matrices(v):
         A = np.array([
             [0, 1, 0, 0],
             [0, -4*C_a/(m*v), 4*C_a/m, -2*C_a*(lf-lr)/(m*v)],
             [0, 0, 0, 1],
             [0, -2*C_a*(lf-lr)/(Iz*v), 2*C_a*(lf-lr)/Iz, -2*C_a*(lf**2+lr**2)/
      ↪(Iz*v)]
         ])
         B = np.array([
             [0, 0],
             [2*C_a/m, 0],
             [0, 0],
             [2*C_a*lf/Iz, 0]
         ])
         return A, B

     def calculate_controllability(A, B):
         C = control.ctrb(A, B)
         return np.linalg.matrix_rank(C)

     def analyze_system(v_range):
         log_ratios = []
         poles = [[] for _ in range(4)]    #Create list with 4 empty lists

         for v in v_range:
             A, B = create_matrices(v)

             # Part (a)
```

```
        C = control.ctrb(A, B)
        singular_values = np.linalg.svd(C)[1]
        log_ratio = np.log10(singular_values[0] / singular_values[-1])
        log_ratios.append(log_ratio)

        # Part (b)
        eigenvalues = np.linalg.eigvals(A)
        for i, pole in enumerate(eigenvalues):
            poles[i].append(pole.real)

    return log_ratios, poles


v_range = np.linspace(1, 40, 500)
log_ratios, poles = analyze_system(v_range)
```

1.2.1 Part (a)

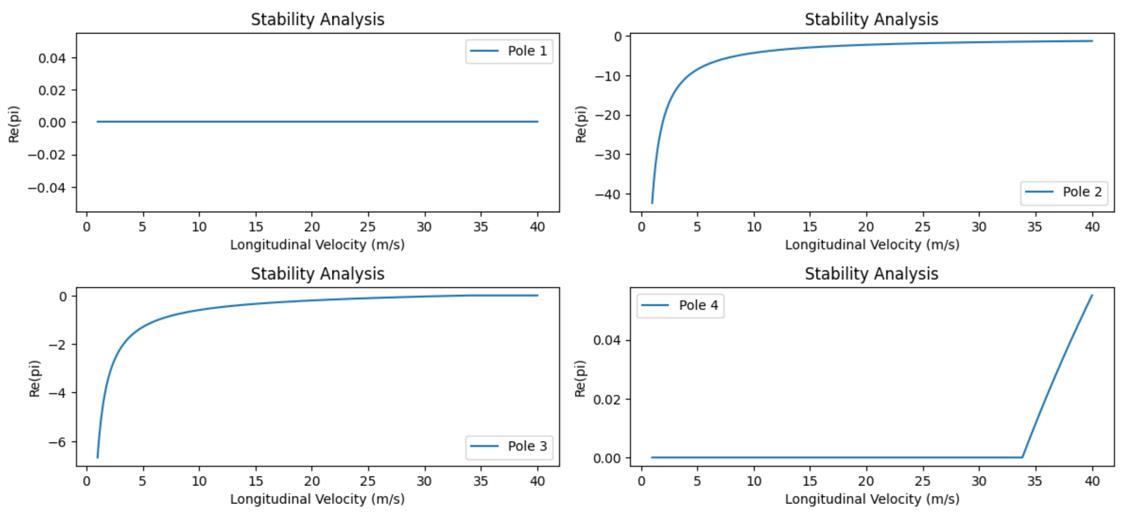
[ ]:
```
plt.figure(figsize=(8, 5))
# plt.subplot(3, 2, 1)
plt.plot(v_range, log_ratios)
plt.title('Controllability Analysis')
plt.xlabel('Longitudinal Velocity (m/s)')
plt.ylabel('log10(sigma_1/sigma_n)')
```

[ ]:
```
Text(0, 0.5, 'log10(sigma_1/sigma_n)')
```



1.2.2 Part (b)

[ ]:
```
# Plot real parts of poles
plt.figure(figsize=(12, 8))
for i, pole_set in enumerate(poles):
    plt.subplot(3, 2, 3+i)
    plt.plot(v_range, pole_set, label=f'Pole {i+1}')
    plt.title('Stability Analysis')
    plt.xlabel('Longitudinal Velocity (m/s)')
    plt.ylabel('Re(pi)')
    plt.legend()

plt.tight_layout()
plt.show()
```

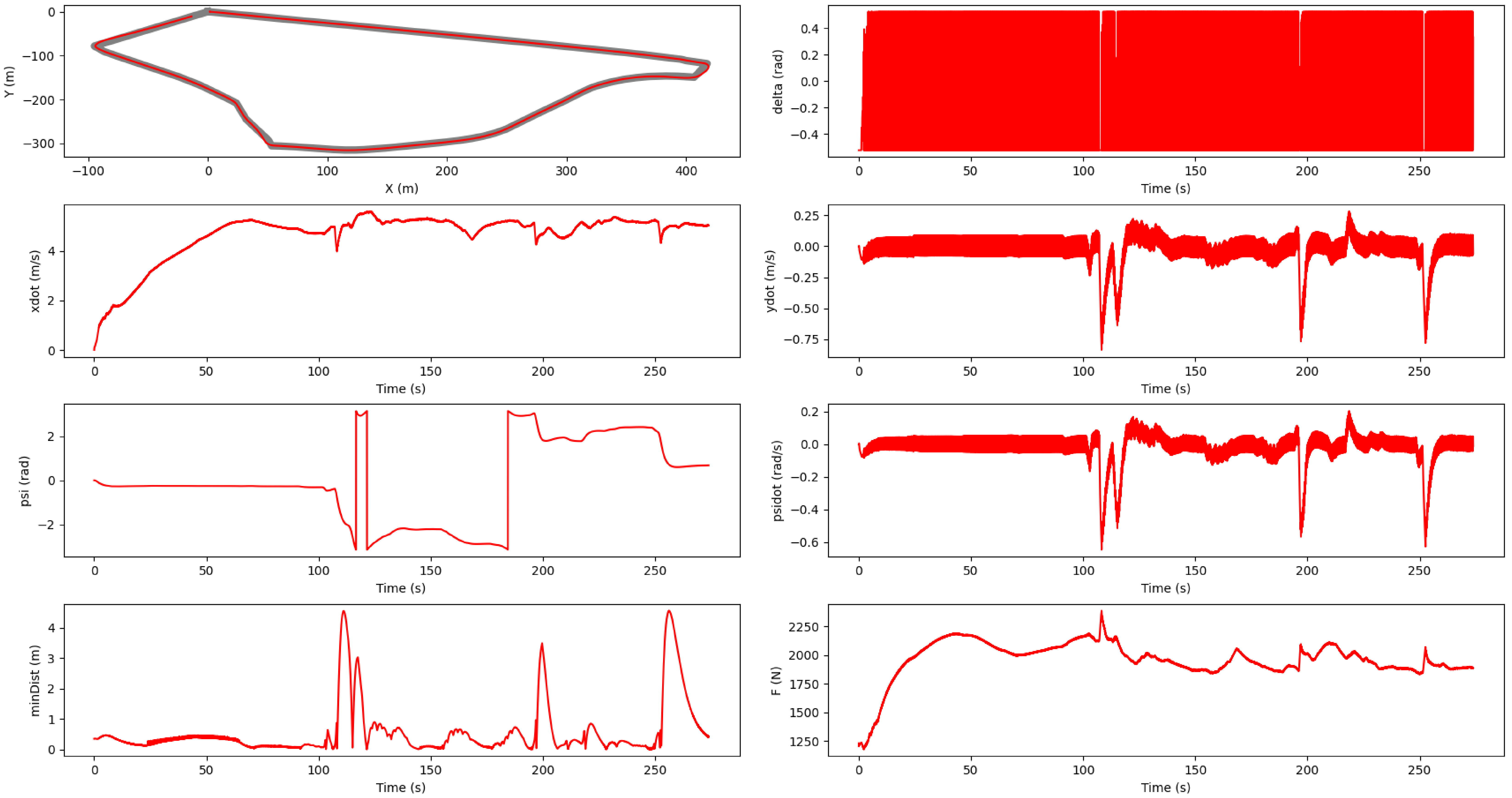


1.2.3 Conclusions

Controllability Analysis: - The plot of log10(1/ n) versus longitudinal velocity shows a decreasing trend. This indicates that as velocity increases, the system's controllability improves. A lower value of log10(1/ n) suggests better conditioning of the controllability matrix, meaning the system is more controllable at higher velocities.

Stability Analysis: - Pole 1: This pole remains near zero across all velocities, indicating that it has little effect on the stability. - Pole 4: This pole also remains near zero up until around 35 m/s, after which is begins to increase and become positive, indicating instability at speeds greater than 35 m/s. - Pole 2 and Pole 3: These poles start with negative real parts and move toward zero as velocity increases. This suggests that the system becomes less stable at higher speeds, as poles approaching zero can lead to slower convergence or potential instability.

Overall Conclusions: - The system becomes more controllable as velocity increases, which is advantageous for control design. While the system is stable at lower velocities (poles have negative real parts), increasing velocity reduces stability margins. Proper control design is needed to ensure stability at higher speeds. These observations highlight the trade-offs in designing control systems for vehicles operating across a range of speeds. Improved controllability at higher speeds must be balanced against potential reductions in stability.


Y (m)
X (m)
delta (rad)
Time (s)
xdot (m/s)
Time (s)
ydot (m/s)
Time (s)
psi (rad)
Time (s)
psidot (rad/s)
Time (s)
minDist (m)
Time (s)
F (N)
Time (s)

```
    # Fill in the respective functions to implement the controller

# Import libraries
import numpy as np
from base_controller import BaseController
from scipy import signal, linalg
from util import *

# CustomController class (inherits from BaseController)
class CustomController(BaseController):

    def __init__(self, trajectory):

        super().__init__(trajectory)

         # PID parameters for lateral controller
        self.Kp_lateral = 1
        self.Ki_lateral = 0
        self.Kd_lateral = 0

        # PID parameters for longitudinal controller
        self.Kp_long = 250
        self.Ki_long = 20
        self.Kd_long = 3

        # Initializing error variables
        self.sum_lat_err = 0
        self.prev_lat_err = 0
        self.sum_long_err = 0
        self.prev_long_err = 0
        self.desired_speed = 5

        # Define constants
        # These can be ignored in P1
        self.lr = 1.39
        self.lf = 1.55
        self.Ca = 20000
        self.Iz = 25854
        self.m = 1888.6
        self.g = 9.81

        # Add additional member variables according to your need here.
        self.A = np.array([
            [0, 1, 0, 0],
            [0, -4*self.Ca/(self.m*self.desired_speed), 4*self.Ca/self.m, -2*self.Ca*(self.lf-self.lr)/(self.m
            [0, 0, 0, 1],
            [0, -2*self.Ca*(self.lf-self.lr)/(self.Iz*self.desired_speed), 2*self.Ca*(self.lf-self.lr)/self.Iz
                -2*self.Ca*(self.lf**2+self.lr**2)/(self.Iz*self.desired_speed)]
        ])

        self.B = np.array([
            [0],
            [2*self.Ca/self.m],
            [0],
            [2*self.Ca*self.lf/self.Iz]
        ])

        self.desired_poles = [-0.001, -100, -5, -0.01]
        # self.desired_poles = [ -0.2, -0.1, -0.001, -0.0001]
        self.K = signal.place_poles(self.A, self.B, self.desired_poles).gain_matrix


    def update(self, timestep):

        trajectory = self.trajectory

        lr = self.lr
        lf = self.lf
        Ca = self.Ca
        Iz = self.Iz
        m = self.m
        g = self.g

        # Fetch the states from the BaseController method
        delT, X, Y, xdot, ydot, psi, psidot = super().getStates(timestep)

        # Design your controllers in the spaces below.
        # Remember, your controllers will need to use the states
        # to calculate control inputs (F, delta).

        # ---------------|Lateral Controller|-------------------------
        """
        Please design your lateral controller below.
        """
```

```python
# Finding the closest point on the trajectory
mindist, closest_idx = closestNode(X, Y, trajectory)
ahead_idx = min(closest_idx + 50, len(trajectory) - 1)
target_x, target_y = trajectory[ahead_idx]
desired_psi = np.arctan2(target_y - Y, target_x - X)

vec_to_vehicle = np.array([X, Y]) - trajectory[ahead_idx]
traj_direction = trajectory[ahead_idx] - trajectory [closest_idx]
traj_heading = np.arctan2(traj_direction[1], traj_direction[0])
relative_heading = psi - traj_heading

e1 = np.linalg.norm(vec_to_vehicle)*np.sin(desired_psi-traj_heading)
e1_dot = xdot * np.sin(relative_heading) + ydot * np.cos(relative_heading)

e2 = wrapToPi(desired_psi - psi)
e2_dot = psidot
state = np.array([e1, e1_dot, e2, e2_dot])
delta = -np.dot(self.K, state)[0]
delta = clamp(delta, -np.pi/6, np.pi/6)


# ---------------|Longitudinal Controller|-------------------------
"""
Please design your longitudinal controller below.
"""
speed_error = self.desired_speed - xdot
# PID controler
self.sum_long_err += speed_error * delT
long_err_derivative = (speed_error - self.prev_long_err) / delT
F = (self.Kp_long * speed_error + self.Ki_long * self.sum_long_err + self.Kd_long * long_err_derivativ
self.prev_long_err = speed_error        # Changing prev to current for next iteration
F = clamp(F, 0, 15736)

# Return all states and calculated control inputs (F, delta)
return X, Y, xdot, ydot, psi, psidot, F, delta
```