Project:P2

Ding Zhao
24-677 Special Topics: Linear Control Systems

Due: Nov 19, 2019, 5:00 pm

- You need to upload your solution to Gradescope (https://www.gradescope.com/) to be graded. The link is on the panel of CANVAS. If you are not familiar about the tool, post your questions on Piazza or ask during the office hours. We will use the online submission time as the timestamp.
- Submit **Q1.py**, **controller.py** and **BuggyStates.npz** to Gradescope under **Programming_P2** and your solutions in **.pdf** format to **Project-P2**. Insert the Buggy Simulator performance plot image in the .pdf. We will test your controller.py and manually check all answers.
- You are recommended to test your codes in Google Colab before submission, to ensure it executes with standard python compilers. Please refer to (http://bit.ly/2rtnrcy) for documentation on how to use Colab.
- Right after each week's recitation, we have a half-hour Q&A session. You are requested to work on the assignment early and bring your questions to this session to take advantage of this support.
- You can also post your questions on the Piazza. We will try our best to give feedback within 24 hours during the workday and within 48 hours during the weekend. We will keep answering questions until 8:00 pm, Monday.
- **Note:** use **python3** for coding the executing the controller scripts.

1 Introduction

In this project, you will complete the following goals:

1. Check controllability and observability of the system.
2. For lateral control of the vehicle, design a feedback control matrix using pole-placement.

[Remember to submit the write-up and codes on the Gradescope]

2 Model

When the objective is to develop a steering control system for automatic lane keeping, it is useful to utilize a dynamic model in which the state variables are in terms of position and orientation error with respect to the road [1].
Hence the linearized model derived in **Exercise 1, P1** can be re-defined in terms of errors in the state variables.
The error-based linearized state-space system should be used for designing the controllers henceforth.

3 Resources

3.1 Buggy Simulator

A Buggy Simulator designed in python has been provided along with the assignment. The simulator takes the control command[steering, longitudinal Force] and then outputs the buggy state after the given fixed time step (fixed fps). Additional script `util.py` contains functions to help you design and execute the controller. Please design your controller in `controller.py`. After the complete run, a response plot is generated by the simulator. This plot contains visualization of the buggy trajectory and variation of states with respect to time.

3.2 Trajectory Data

The trajectory is given in `buggyTrace.csv`. It contains the coordinates of the trajectory: (x, y). The satellite map of the track is shown in Figure 4.

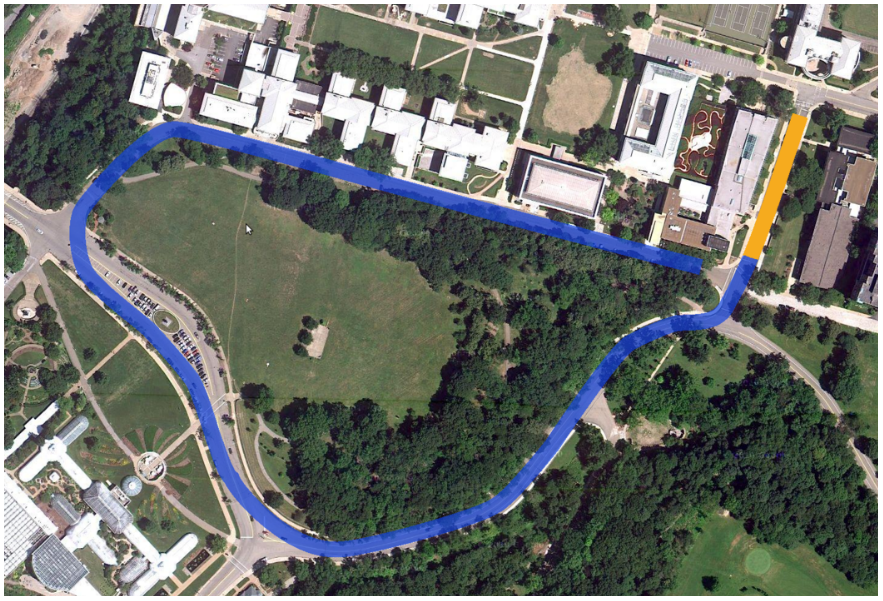


Figure 1: Buggy track[3]

4 P2:Problems [Due 5:00 PM, November 19]

Exercise 1. Consider the linearized, error-based state space system for the vehicle:

1. Check the controllability and observability of the system at the following longitudinal velocities: 2 m/s, 5 m/s and 8 m/s.

2. For longitudinal velocities V from 1m/s to 40 m/s, plot the following:

   (a) $log_{10}(\frac{\sigma_1}{\sigma_n})$ vs V (m/s), where σ_i: i'th singular value of the controllability matrix P (i=1,2,.......n).

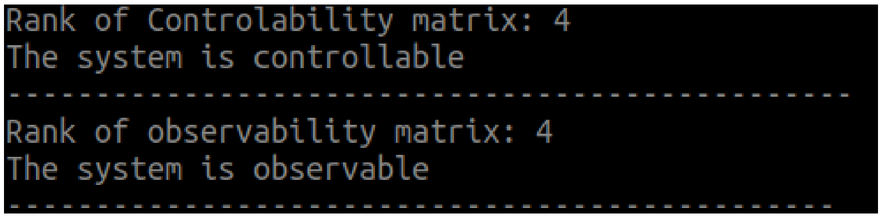
   (b) $Re(p_i)$ vs V (m/s), where Re:Real part, p_i: i'th pole of the continuous state space system. [Use 4 subplots, one for each of the 4 poles]

   Also, what can you conclude about the controllability and stability of the system, by observing these two plots.

[Submit you answers in the pdf file and also the python script. The python script should be named **Q1.py**]

Solution

1.
```python
"""
Exercise: To check the controlability and observability of the system.
"""

import numpy as np
import scipy
import scipy.signal as signal
from scipy.ndimage import gaussian_filter1d
import control
import matplotlib.pyplot as plt

#  Vehicle Dynamics Parametrs:
lr=1.7
lf=1.1
Ca=15000.0
Iz=3344.0
f=0.01
m=2000.0
g=10

delT=0.05

def state_space_system(Vx):
        # State Space Equation:
        A = np.array([[0,1,0,0],[0,-4*Ca/(m*Vx),
                4*Ca/m,2*Ca*(lr-lf)/(m*Vx)],[0,0,0,1],
                [0,2*Ca*(lr-lf)/(Iz*Vx),2*Ca*(lf-lr)/Iz,-2*Ca*(lr*lr+lf*lf)/(Iz*Vx)

        B = np.array([[0,0],[2*Ca/m,0],[0,0],[2*Ca*lf/Iz,0]])
        C=np.identity(4)

        # D=np.array([0,0,0,0]).reshape(-1,1)
        D=np.array([[0,0],[0,0],[0,0],[0,0]])
        return A,B,C,D

# Checking the Controllability.
A,B,C,D = state_space_system(Vx=6)        # at 6 m/s
# Controlablility matrix:
P=np.hstack((B,np.matmul(A,B),np.matmul(np.linalg.matrix_power(A,2),B)
        ,np.matmul(np.linalg.matrix_power(A,3),B)))

#Computing the rank of controlability matrix
r=np.linalg.matrix_rank(P)
print('Rank of Controlability matrix: {}'.format(r))
if(r==4):  # If the rank of the Controlability matrix =4, it is controllable
        print("The system is controllable")
print('------------------------------------------------')

# Cheking the observability
Q=np.vstack((C,np.matmul(C,A),np.matmul(np.linalg.matrix_power(C,2),A),
        np.matmul(np.linalg.matrix_power(C,3),A)))
r=np.linalg.matrix_rank(Q)
print('Rank of observability matrix: {}'.format(r))
if r==4:        # if the rank of Observability matrix = 4, the system is observabl
        print("The system is observable")
print('-----------------------------------------------')


V=np.arange(1,41,0.1)
ratio=[]
for i,vx in enumerate(V):
        A,B,C,D = state_space_system(vx)
        # Computing the controlability matrix
        P=np.hstack((B,np.matmul(A,B),np.matmul(np.linalg.matrix_power(A,2),B),
                np.matmul(np.linalg.matrix_power(A,3),B)))

        # SVD decomposition to compute the singular values
        u,s,vh=np.linalg.svd(P)
        ratio.append(s[0]/s[-1])

        sys=control.StateSpace(A,B,C,D)

        cpoles=np.sort(control.pole(sys))
        # np.argmin()
        # poles.append(min(np.real(cpoles)))
        if(i==0):
                poles=np.real(cpoles)
        else:
                poles=np.vstack((poles,np.real(cpoles)))


plt.plot(V, np.log10(ratio))
plt.ylabel('log(Ratio)')
plt.xlabel('Velocity (m/s)')
# plt.savefig('1_1.png')
plt.show()
fig,ax=plt.subplots(4,1)

for i in range(4):
        ax[i].plot(V, poles[:,i])
        ax[i].set_ylim([-30,15])

# plt.ylim([-30,15])
plt.xlabel('Velocity (m/s)')
plt.show()
# plt.savefig('1_2.png')
```



```
Rank of Controlability matrix: 4
The system is controllable
--------------------------------------------------
Rank of observability matrix: 4
The system is observable
--------------------------------------------------
```

Figure 2:

2. (a) The ratio of singular values of the controllability matrix reflects the defectiveness of the system. The defectiveness of the system is inversely proportional to the ratio, i.e. smaller the ratio $\implies$ the system is less likely to be defective. Therefor, the system is comparatively more controllable in the lateral direction at higher longitudinal velocities.

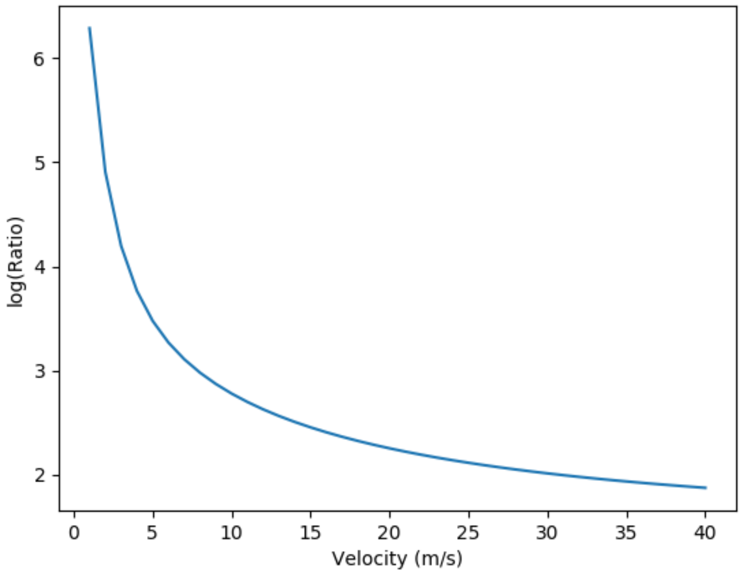


Figure 3:

(b) The system is a second order system with two zeros poles. With an increase in the longitudinal velocity, the conjugate pole pairs move closer to the imaginary axis. This represents the behavior that the system tends to be less stable as the velocity increases.

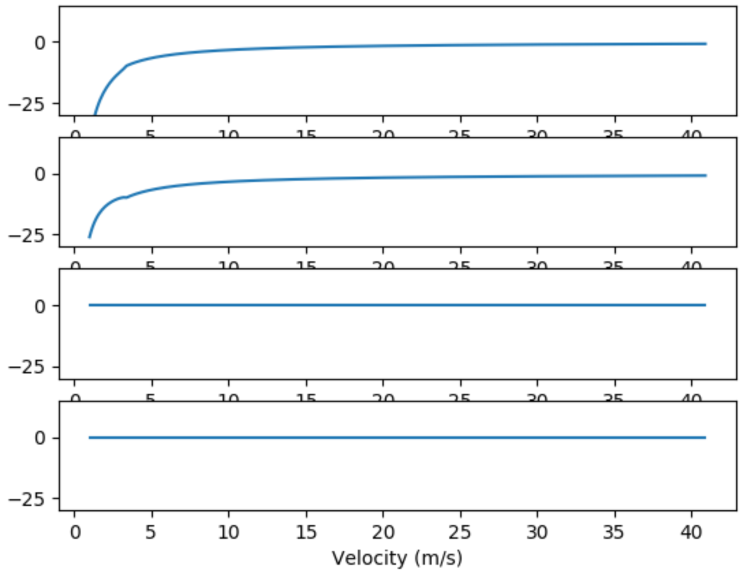


Figure 4:

In conclusion, as the longitudinal velocity increases, it is easier to make a change in the lateral state (controllability) though there is a higher risk of the system becoming unstable.

Exercise 2. Consider the linearized, error-based state space model of the vehicle:

For the lateral control of the vehicle, design a state feedback control matrix F using pole placement. Tune the poles of the closed loop system such that it can achieve the performance criteria mentioned below.
For the longitudinal control, use a PID controller.
Design the controllers in **controller.py**.
[You have to edit only the controller.py python script]
Execute the `main.py` python script to check your controller. It generates a performance plot and saves the vehicle states in a .npz file. Submit the Buggy states in .npz file, the response plots in pdf file, and your controller in the controller.py script.
Your controller is required to achieve the following performance criteria:

1. Time to complete the loop = 410 s
2. Maximum deviation from the reference trajectory = 8.0 m
3. Average deviation from the reference trajectory = 4.5 m

Solution

```
from BuggySimulator import *
import numpy as np
import scipy
import cmath
from scipy.ndimage import gaussian_filter1d
from util import *
from scipy import signal

class controller():

    def __init__(self,traj,vehicle):
        self.vehicle=vehicle
        self.traj=traj
        self.prev_vx_error=0
        self.integral_vx_error=0
        self.curv=self.compute_curvature()

    def compute_curvature(self):
        """
        Function to compute and return the curvature of trajectory.
        """
        sigma_gaus = 10
        traj=self.traj
        xp = scipy.ndimage.filters.gaussian_filter1d(input=traj[:,0],
            sigma=sigma_gaus,order=1)
        xpp = scipy.ndimage.filters.gaussian_filter1d(input=traj[:,0],
            sigma=sigma_gaus,order=2)
        yp = scipy.ndimage.filters.gaussian_filter1d(input=traj[:,1],
            sigma=sigma_gaus,order=1)
        ypp = scipy.ndimage.filters.gaussian_filter1d(input=traj[:,1],
            sigma=sigma_gaus,order=2)
        curv=np.zeros(len(traj))

        for i in range(len(xp)):
            curv[i] = (xp[i]*ypp[i] - yp[i]*xpp[i])/(xp[i]**2 + yp[i]**2)**1.5

        return curv

    def control_update(self):

        traj=self.traj
        vehicle=self.vehicle

        lr = vehicle.lr
        lf = vehicle.lf
        Ca = vehicle.Ca
        Iz = vehicle.Iz
        f = vehicle.f
        m = vehicle.m
        g = vehicle.g

        delT = 0.05

        #reading current vehicle states
        X = vehicle.state.X
        Y = vehicle.state.Y
        xdot = vehicle.state.xd
        ydot = vehicle.state.yd
        phi = vehicle.state.phi
        phidot = vehicle.state.phid
        delta = vehicle.state.delta

        mindist, index = closest_node(X, Y, traj)

        Vx = 3.5  # constant desired longitudinal velocity of the vehicle

        # Computing the curvature of trajectory
        curv=self.curv

        # ------------|Lateral Controller|---------------------------------
```

```
#Ref Eq (2.45) Vehicle Dynamics and Control by Rajesh Rajamani
A = [[0,1,0,0],[0,-4*Ca/(m*Vx),4*Ca/m,2*Ca*(lr-lf)/(m*Vx)],[0,0,0,1],
[0,2*Ca*(lr-lf)/(Iz*Vx),2*Ca*(lf-lr)/Iz,-2*Ca*(lr*lr+lf*lf)/(Iz*Vx)]]
B = [[0],[2*Ca/m],[0],[2*Ca*lf/Iz]]

C = np.identity(4)

D = [[0],[0],[0],[0]]
#D = [[0,0],[0,0],[0,0],[0,0]]

#State space system (continuous)
syscont = signal.StateSpace(A,B,C,D)

#Discretizing state space system
sysdisc = syscont.to_discrete(delT)
Ad = sysdisc.A
Bd = sysdisc.B

#Computing the Feedback control matrix by pole placement

# poles=np.array([-1, -0.5, 0.1, 1])
poles=np.array([-0.2,-0.1,0.3,1.2])

fsf=signal.place_poles(Ad,Bd,poles)
K=fsf.gain_matrix

if index<len(traj)-100:
    idx_fwd = 100
else:
    idx_fwd = len(traj)-index-1

phides = np.arctan2((traj[index+idx_fwd][1]-Y),(traj[index+idx_fwd][0]
    X))
phidesdot = xdot*curv[index+idx_fwd]

e = np.zeros(4)

#Ref p34 Vehicle Dynamics and Control by Rajesh Rajamani
e[0] = (Y - traj[index+idx_fwd][1])*np.cos(phides) - (X - traj[index+
    idx_fwd][0])*np.sin(phides)
e[2] = wrap2pi(phi - phides)
e[1] = ydot + xdot*e[2]
e[3] = phidot - phidesdot

error = np.matrix(e)
```

```
deltades = float(-K*np.transpose(error))
#deltades = float(-K[0,:]*np.transpose(error))
# print(deltades)
deltad = (deltades - delta)/0.05

#--------|Longitudinal Controller|------------------------------
kp=400
kd=300
ki=-0.1

# Computing the errors
vx_error=Vx-xdot
self.integral_vx_error+=vx_error
derivative_error=vx_error-self.prev_vx_error
F=kp*vx_error + ki*self.integral_vx_error*delT +
kd*derivative_error/delT
# -----------------------------------------------------------------

controlinp = vehicle.command(F,deltad)

# Storing the parameters for the next epoch
self.prev_vx_error=vx_error

return controlinp
```

5 Appendix

(Already covered in P1)

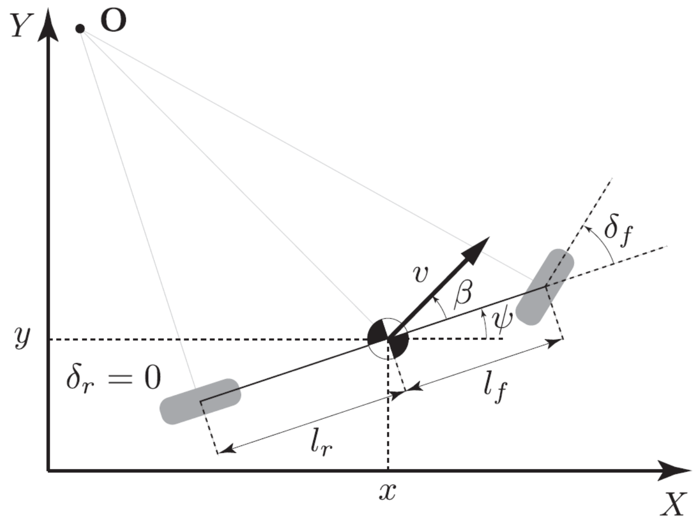


Figure 5: Bicycle model[2]

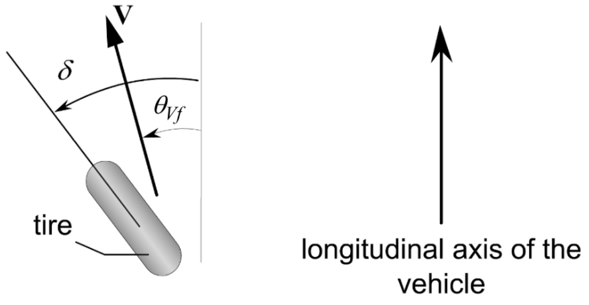


Figure 6: Tire slip-angle[2]

Here you will use the bicycle model for the vehicle, which is a popular model in the study of vehicle dynamics. Shown in Figure 5, the car is modeled as a two-wheel vehicle in two degree of freedom, described in longitudinal and lateral dynamics separately. The model parameters are defined in Table 1.

5.1 Lateral dynamics

Ignoring road bank angle and applying Newton's second law of motion along the y axis

$$ma_y = F_{yf} \cos \delta_f + F_{yr}$$

where $a_y = \left(\frac{d^2y}{dt^2}\right)_{inertial}$ is the inertial acceleration of the vehicle at the center of geometry in the direction of the y axis, F_{yf} and F_{yr} are the lateral tire forces of the front and rear wheels respectively and δ_f is the front wheel angle which will be denoted as δ later. Two terms contribute to a_y: the acceleration $\ddot{y}$ which is due to motion along the y axis and the centripetal acceleration . Hence

$$a_y = \ddot{y} + \dot{\psi}\dot{x}$$

Combining the two equations, the equation for the lateral translational motion of the vehicle is obtained as

$$\ddot{y} = -\dot{\psi}\dot{x} + \frac{1}{m}(F_{yf}\cos\delta + F_{yr})$$

Moment balance about the axis yields the equation for the yaw dynamics as

$$\ddot{\psi} I_z = l_f F_{yf} - l_r F_{yr}$$

The next step is to model the lateral tire forces F_{yf} and F_{yr}. Experimental results show that the lateral tire force of a tire is proportional to the slip-angle for small slip-angles when vehicle's speed is large enough, let's say when $\dot{x} \geq 0.5$ m/s. The slip angle of a tire is defined as the angle between the orientation of the tire and the orientation of the velocity vector of the vehicle. the slip angle of the front and rear wheel is

$$\alpha_f = \delta - \theta_{Vf}$$
$$\alpha_r = -\theta_{Vr}$$

where θ_{Vp} is the angle between the velocity vector and the longitudinal axis of the vehicle, for $p \in \{f, r\}$. A linear approximation of the tire forces are given by

$$F_{yf} = 2C_\alpha \left(\delta - \frac{\dot{y} + l_f\dot{\psi}}{\dot{x}}\right)$$
$$F_{yr} = 2C_\alpha \left(-\frac{\dot{y} - l_r\dot{\psi}}{\dot{x}}\right)$$

where C_α is called the cornering stiffness of tires. If $\dot{x} < 0.5$ m/s, we just set F_{yf} and F_{yr} both to zeros.

5.2 Longitudinal dynamics

Similarly, a force balance along the vehicle longitudinal axis yields

$$\ddot{x} = \dot{\psi}\dot{y} + a_x$$
$$ma_x = F - sign(\dot{x})F_f$$
$$F_f = fmg$$

where F is the total tire force along x axis, F_f is the force due to rolling resistance at the tires, and f is the friction coefficient. *sign* function returns +1 when $\dot{x} \geq 1$ otherwise -1.

5.3 Global coordinates

In the global frame we have

$$\begin{aligned} \dot{X} &= \dot{x}\cos\psi - \dot{y}\sin\psi \\ \dot{Y} &= \dot{x}\sin\psi + \dot{y}\cos\psi \end{aligned}$$

5.4 System equation

Gathering all the equations, if $\dot{x} \geq 0.5$ m/s we have:

$$\begin{aligned} \ddot{y} &= -\dot{\psi}\dot{x} + \frac{2C_\alpha}{m}(\cos\delta\left(\delta - \frac{\dot{y} + l_f\dot{\psi}}{\dot{x}}\right) - \frac{\dot{y} - l_r\dot{\psi}}{\dot{x}}) \\ \ddot{x} &= \dot{\psi}\dot{y} + \frac{1}{m}(F - fmg) \\ \ddot{\psi} &= \frac{2l_fC_\alpha}{I_z}\left(\delta - \frac{\dot{y} + l_f\dot{\psi}}{\dot{x}}\right) - \frac{2l_rC_\alpha}{I_z}\left(-\frac{\dot{y} - l_r\dot{\psi}}{\dot{x}}\right) \\ \dot{X} &= \dot{x}\cos\psi - \dot{y}\sin\psi \\ \dot{Y} &= \dot{x}\sin\psi + \dot{y}\cos\psi \end{aligned}$$

otherwise since the lateral tire forces are zeros we only consider the longitudinal model.

5.5 Measurements

The observable states are with some Gaussian noise $\epsilon = N(0, \sigma)$, where

$$y = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \\ X \\ Y \\ \psi \end{bmatrix} + \epsilon,\ \sigma = \begin{bmatrix} 0.5 & & & \cdots & & 0 \\ & 0.5 & & & & \\ & & 0.05 & & & \\ \vdots & & & 0.05 & & \\ & & & & 1 & \\ 0 & & & & & 0.5 \end{bmatrix}$$

.

5.6 Physical constraints

The system satisfies the constraints that:

$$\begin{aligned} &|\delta| \leqslant \tfrac{\pi}{6}\ rad/s \\ &|\dot{\delta}| \leqslant \tfrac{\pi}{6}\ rad/s \\ &|F| \leqslant 10000\ N \\ &0 \text{ m/s} \leqslant \dot{x} \leqslant 100 \text{ m/s} \\ &|\dot{y}| \leqslant 10m/s \end{aligned}$$

Table 1: Model parameters.

Name	Description	Unit	Value
$(\dot{x}, \dot{y})$	Vehicle's velocity along the direction of vehicle frame	m/s	State
(X, Y)	Vehicle's coordinates in the world frame	m	State
ψ, $\dot{\psi}$	Body yaw angle, angular speed	rad	State
δ or δ_f	Front wheel angle	rad	State
$\dot{\delta}$	Steering Rate	rad	Input
F	Total input force	N	Input
l_r	Length from front tire to the center of mass	m	1.7
l_f	Length from front tire to the center of mass	m	1.1
C_α	Cornering stiffness of each tire	N	15000
I_z	Yaw intertia	kg m^2	3344
F_{pq}	Tire force, $p \in \{x, y\}$,$q \in \{f, r\}$	N	Depend on input force
m	vehicle mass	Kg	2000
f	Friction coefficient	1	0.01

6 Reference

1. Rajamani Rajesh. Vehicle dynamics and control. Springer Science & Business Media, 2011.

2. Kong Jason, et al. "Kinematic and dynamic vehicle models for autonomous driving control design." Intelligent Vehicles Symposium, 2015.

3. cmubuggy.org, `https://cmubuggy.org/reference/File:Course_hill1.png`