Constant radius cornering

September 4, 2022

[ ]:
```
import sys
sys.path.append('../')
from Driver import Driver
from Race import Race
from TMEasy import Tire
from Vehicle import Vehicle
from Track import Track
from Function import Function
import numpy as np
```

[ ]:
```
%load_ext autoreload
%autoreload 2
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

1 Input data

1.1 Tires

[ ]:
```
radius = 0.35
mass = 13.44
Jz_tire = 1
cz = 392000
dfx0 = 100000
dfy0 = 80000
fxm = 3900
fym = 3650
sxm = 0.11
sym = 0.16
fxs = 3600
fys = 3600
sxs = 0.4
s_ys = 0.5
n2L0 = 0.18
sy0 = 0.19
syE = 0.35
lamb = 2/3
```

```
frr = 0.015
TMEasy = Tire(radius, mass, Jz_tire, cz, dfx0, dfy0, fxm, fym, sxm, sym, fxs,␣
↪fys, sxs, s_ys, sy0, syE, lamb, n2L0, frr)
TMEasy.all_info()
```

```
Vertical force considered: 16000N
Camber slip = 0.00
Bore slip = 0.00

Coenering stiffness = 1186.9N/
```

1.2 Driver

[ ]:
```
driver = Driver(accelerator='PIDSSCD', steering='steering2')
```

```
steering2 defined.
```

1.3 Track

[ ]:
```
hx = 0
hy = 0
wx = 0.1
wy = 100
# mean = 0
# variance = 1

zf = lambda x, y: hx * np.sin(wx * x) + hy * np.sin(wy * y) #+ 0.01 * np.
↪sin(10*x)
vzf = lambda x, vx, y, vy: hx * wx * vx * np.cos(wx * x) + hy * wy * vy * np.
↪cos(wy * y)

track = Track(zf, vzf)
# track.set_noise(mean, variance)
# track.profile(x=[0, 1000], y=[0,100], resolution=1)
```

1.4 Vehicle + Suspension

[ ]:
```
vehicle_mass = 265
Ixx = 50
Iyy = 91.8
Izz = 91.8
lf = 1.15
lr = 1.2
wf = 1.53
wr = 1.53
af = 0.9
cd = 1.34
CG_height = 0.278
```

```
car = Vehicle(TMEasy, vehicle_mass, Ixx, Iyy, Izz, lf, lr, wf, wr, af, cd,␣
↪CG_height)
```

[ ]:
```
K_sf = 55861
K_sr = 43633
C_sf = 0.3 * np.sqrt(K_sf * Iyy)
C_sr = 0.3 * np.sqrt(K_sr * Iyy)
car.set_suspension(K_sf, K_sr, C_sf, C_sr)

# G = 186e9
# df = 0.012095179
# af = 0.25
# bf = 0.54
car.set_anti_roll_bar('f', K_arz=280.51)

# dr = 0.013
# ar = 0.3
# br = 0.8
car.set_anti_roll_bar('r', K_arz=596.31)

# gamma1 = np.deg2rad(0) # O ângulo de cambagem positivo joga o carro para a␣
↪direita
# gamma2 = np.deg2rad(0)
# gamma3 = 0.6
# gamma4 = 0.63
# car.set_camber(gamma1, gamma2, gamma3, gamma4)
```

```
Anti-roll Bar (front) = 4.9 Nm/
Anti-roll Bar (rear)  = 10.4 Nm/
```

[ ]: 596.31

1.5 Simulation

[ ]:
```
ini_sol = [0,  0,    0.00002,  16.38799,  -0.,        0.,        -0.,
  -0.00023,  -0.,        -0.,        -0.,        -0.,        0.00003,   0.00003,
  -0.00003,  -0.00003,   0.,         0.,         -0.,        -0.,        46.949,
  46.949,    46.89208,  46.89209]
sim = Race(
    car,
    driver,
    track,
    max_time=35,
    rtol=1e-3,
    atol=1e-3,
    max_step=1e-3,
    initial_solution=ini_sol)
sim.post_process()
```

```
Solution Finished
steering2 defined.
```

2 Comparison with Adams

[ ]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from Function import Function
```

2.1 Auxiliar Functions

[ ]:
```python
def normalize_angle(angle):
    """
    function to reduce an angle to its equivalent value in the unit circle
    """
    unit_angle = np.modf(angle / (2 * np.pi))[0] * 2 * np.pi
    if unit_angle < 0:
        unit_angle += 2 * np.pi

    return unit_angle

def standard_angle(angle):
    """
    function to make the andgle function continuous
    """
    unit_angle = np.copy(angle)
    for i in range(1, len(unit_angle)):
        if unit_angle[i-1] > 0 and unit_angle[i] < 0:
            unit_angle[i] += 360

    return unit_angle

def calcula_dot(f, t):
    n = len(f)
    df = np.zeros(n)

    for i in range(n):
        if i == 0:
            df[i] = (-f[i+2] + 4 * f[i+1] - 3 * f[i]) / (2 * (t[i+1] - t[i]))
        elif i == n-1:
            df[i] = (3 * f[i] - 4 * f[i-1] + f[i-2]) / (2 * (t[i] - t[i-1]))
        else:
            df[i] = (f[i+1] - f[i-1]) / (2 * (t[i] - t[i-1]))

    return df
```

```python
def calcula_dot_dot(f, t):
    n = len(f)
    ddf = np.zeros(n)

    for i in range(n):
        if i == 0:
            ddf[i] = (-f[i+3] + 4 * f[i+2] - 5 * f[i+1] + 2 * f[i]) / ((t[i+1]␣
↪- t[i])**2)
        elif i == n-1:
            ddf[i] = (2 * f[i] - 5 * f[i-1] + 4 * f[i-2] - f[i-3]) / ((t[i] -␣
↪t[i-1])**2)
        else:
            ddf[i] = (f[i+1] - 2 * f[i] + f[i-1]) / ((t[i+1] - t[i])**2)

    return ddf
```

2.2 Importing data

[ ]:
```python
'''
A.   .EPR_FV01.ISOv60_dlc.chassis_displacements.lateral (mm)
B.   .EPR_FV01.ISOv60_dlc.chassis_displacements.longitudinal (mm)
C.   .EPR_FV01.ISOv60_dlc.chassis_displacements.pitch (deg)
D.   .EPR_FV01.ISOv60_dlc.chassis_displacements.roll (deg)
E.   .EPR_FV01.ISOv60_dlc.chassis_displacements.TIME (sec)
'''
data = pd.read_csv("FSAE_data/r100v60.txt", sep=',')#, delimiter='\n')
data.head()
```

[ ]:

	A	B	C	D	E
0	0.0	-555.2203	-0.124191	0.0	0.01
1	0.0	-391.3431	-0.124154	0.0	0.02
2	0.0	-227.4666	-0.124116	0.0	0.03
3	0.0	-63.5903	-0.124077	0.0	0.04
4	0.0	100.2863	-0.124041	0.0	0.05

[ ]:
```python
'''
A.   .EPR_FV01.ISOv60_dlc.chassis_displacements.vertical (mm)
B.   .EPR_FV01.ISOv60_dlc.chassis_displacements.yaw (deg)
'''
data2 = pd.read_csv("FSAE_data/r100v60_2.txt", sep=',')#, delimiter='\n')
data2.head()
```

[ ]:

	A	B
0	240.7241	0.0
1	240.7241	0.0
2	240.7242	0.0
3	240.7242	0.0

```
4  240.7243  0.0
```

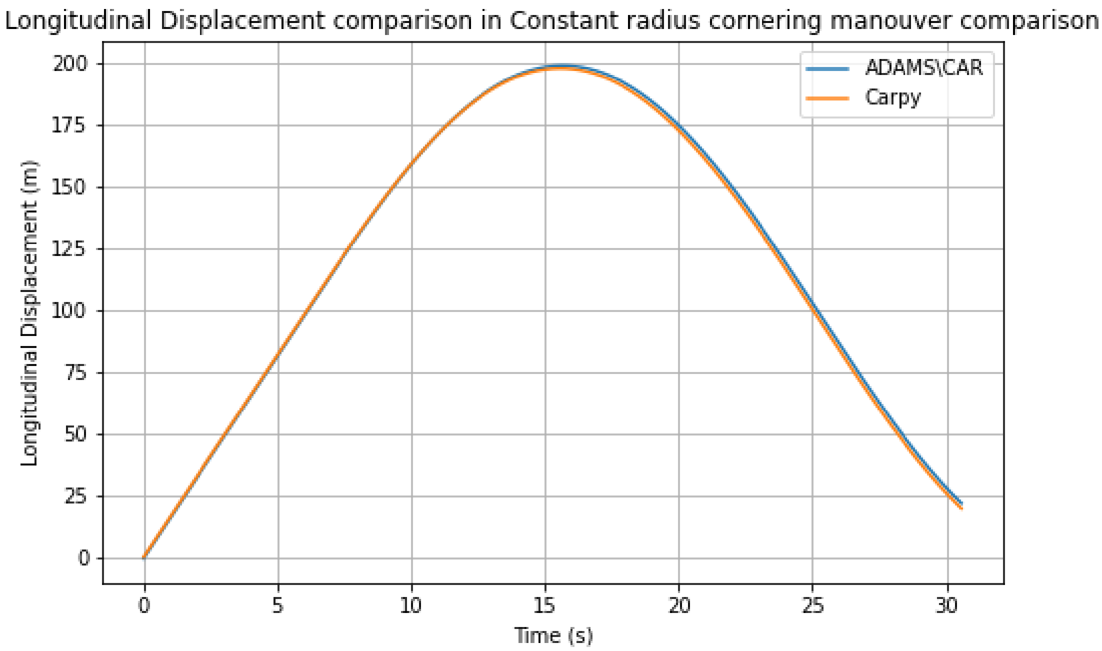
[ ]:
```
t_a = np.array(data['E'])
x = np.array(data['B']) / 1000
y = np.array(data['A']) / 1000
z = np.array(data2['A']) / 1000
roll = np.array(data['D'])
pitch = np.array(data['C'])
yaw = standard_angle(np.array(data2['B']))
```

2.3 Generating Functions

[ ]:
```
f_x = Function(t_a, x, 'Time (s)', 'Longitudinal Displacement (m)', 'x',␣
 ↪method='linear')
f_y = Function(t_a, y, 'Time (s)', 'Lateral Displacement (m)', 'y',␣
 ↪method='linear')
f_z = Function(t_a, z, 'Time (s)', 'Vertical Displacement (m)', 'z',␣
 ↪method='linear')
f_roll = Function(t_a, roll, 'Time (s)', 'Roll angle (deg)', 'roll',␣
 ↪method='linear')
f_pitch = Function(t_a, pitch, 'Time (s)', 'Pitch angle (deg)', 'pitch',␣
 ↪method='linear')
f_yaw = Function(t_a, yaw, 'Time (s)', 'Yaw angle (deg)', 'yaw',␣
 ↪method='linear')
```

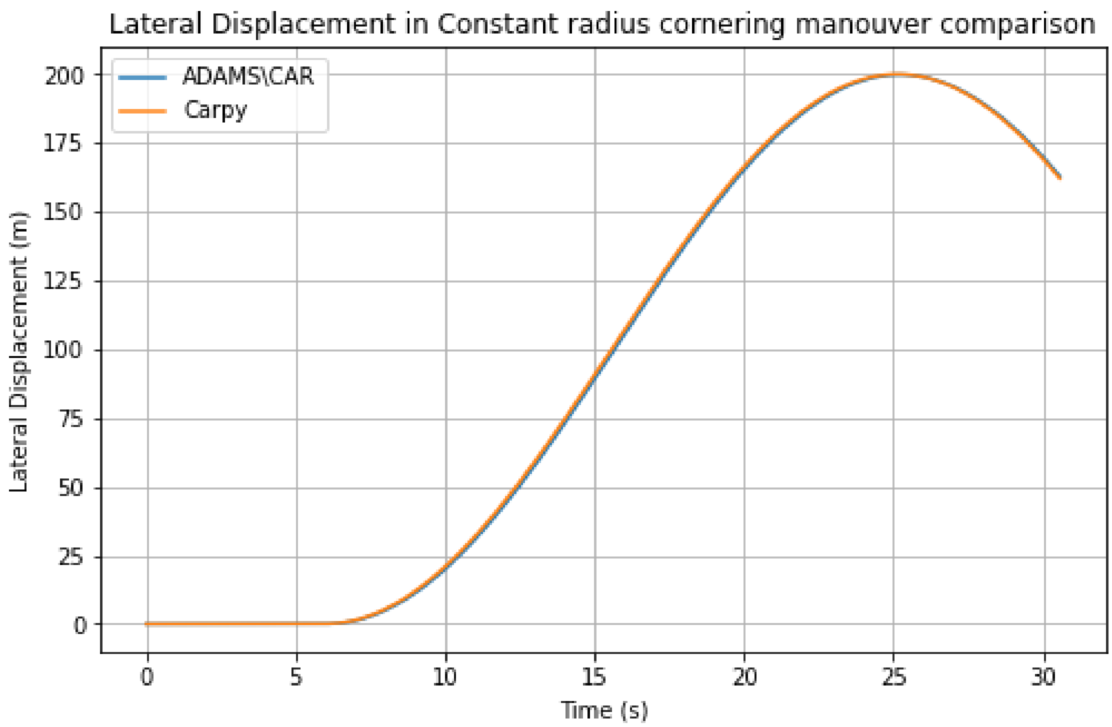
2.3.1 Longitudinal Displacement

[ ]:
```
t_c = t_a
plt.figure(figsize=(8.09, 5))
plt.plot(t_a, f_x(t_a), label='adams')
plt.plot(t_a, sim.x(t_c) - sim.x(t_c[0]), label='carpy')
plt.legend(['ADAMS\CAR', 'Carpy'])
plt.xlabel('Time (s)')
plt.ylabel('Longitudinal Displacement (m)')
plt.title('Longitudinal Displacement comparison in Constant radius cornering␣
 ↪manouver comparison')
plt.grid()
plt.show()
```



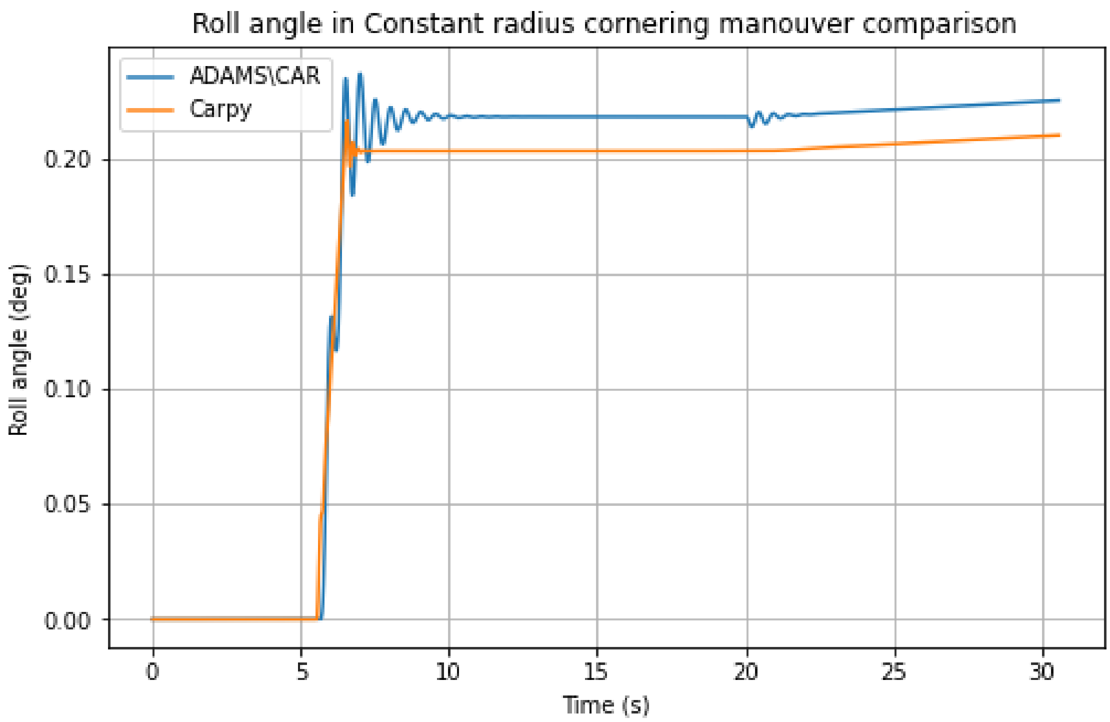
2.3.2 Lateral Displacement

[ ]:
```
plt.figure(figsize=(8.09, 5))
plt.plot(t_a, -f_y(t_a), label='adams')
plt.plot(t_a, sim.y(t_c), label='carpy')
plt.legend(['ADAMS\CAR', 'Carpy'])
plt.xlabel('Time (s)')
plt.ylabel('Lateral Displacement (m)')
plt.title('Lateral Displacement in Constant radius cornering manouver␣
 ↪comparison')
plt.grid()
plt.show()
```



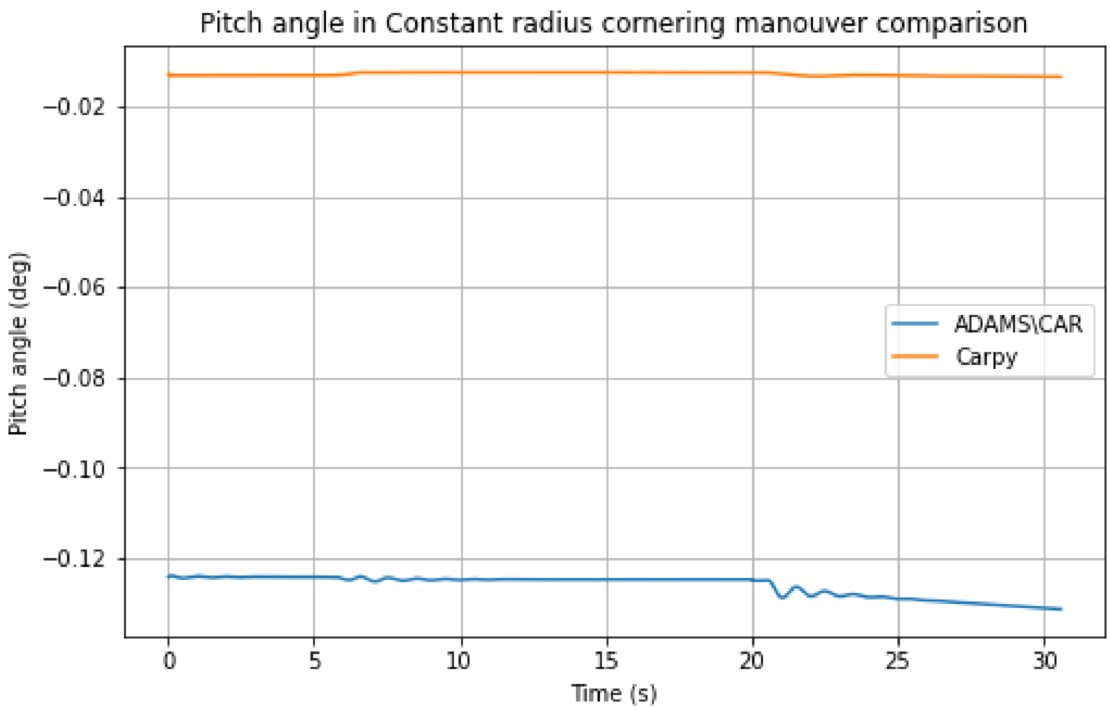
2.3.3 Roll Angle

[ ]:
```
plt.figure(figsize=(8.09, 5))
plt.plot(t_a, -f_roll(t_a), label='adams')
plt.plot(t_a, sim.phi(t_c), label='carpy')
plt.legend(['ADAMS\CAR', 'Carpy'])
plt.xlabel('Time (s)')
plt.ylabel('Roll angle (deg)')
plt.title('Roll angle in Constant radius cornering manouver comparison')
plt.grid()
plt.show()
```



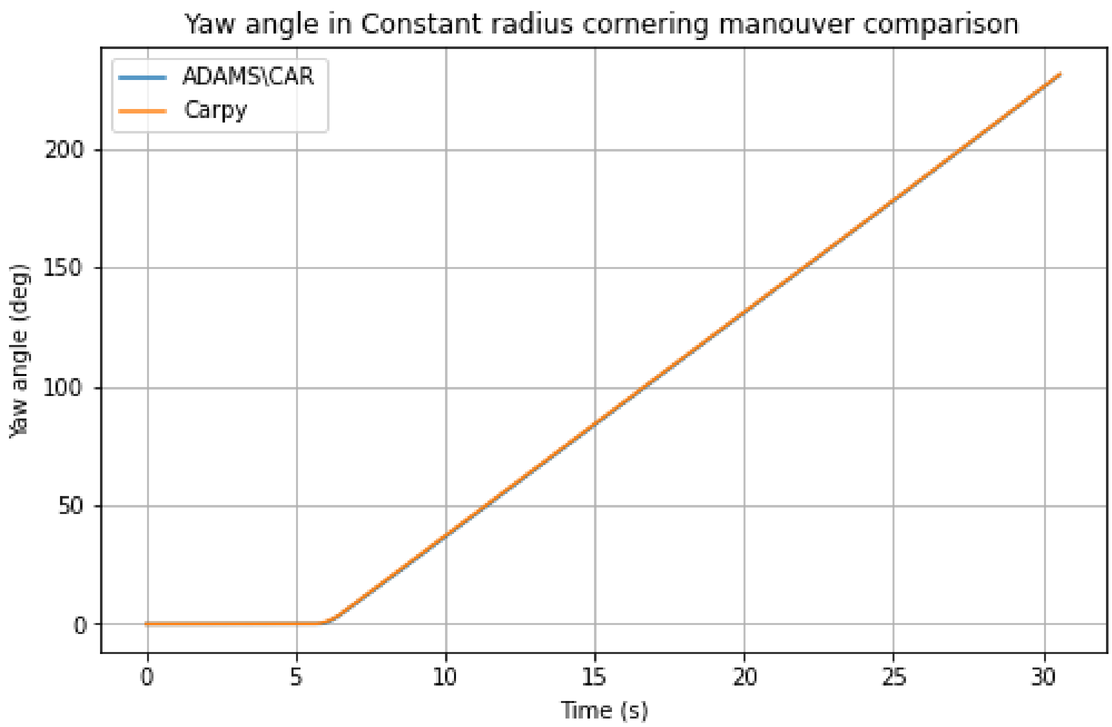
2.3.4 Pitch Angle

[ ]:
```
plt.figure(figsize=(8.09, 5))
plt.plot(t_a, f_pitch(t_a), label='adams')
plt.plot(t_a, sim.theta(t_c), label='carpy')
plt.legend(['ADAMS\CAR', 'Carpy'])
plt.xlabel('Time (s)')
plt.ylabel('Pitch angle (deg)')
plt.title('Pitch angle in Constant radius cornering manouver comparison')
plt.grid()
plt.show()
```



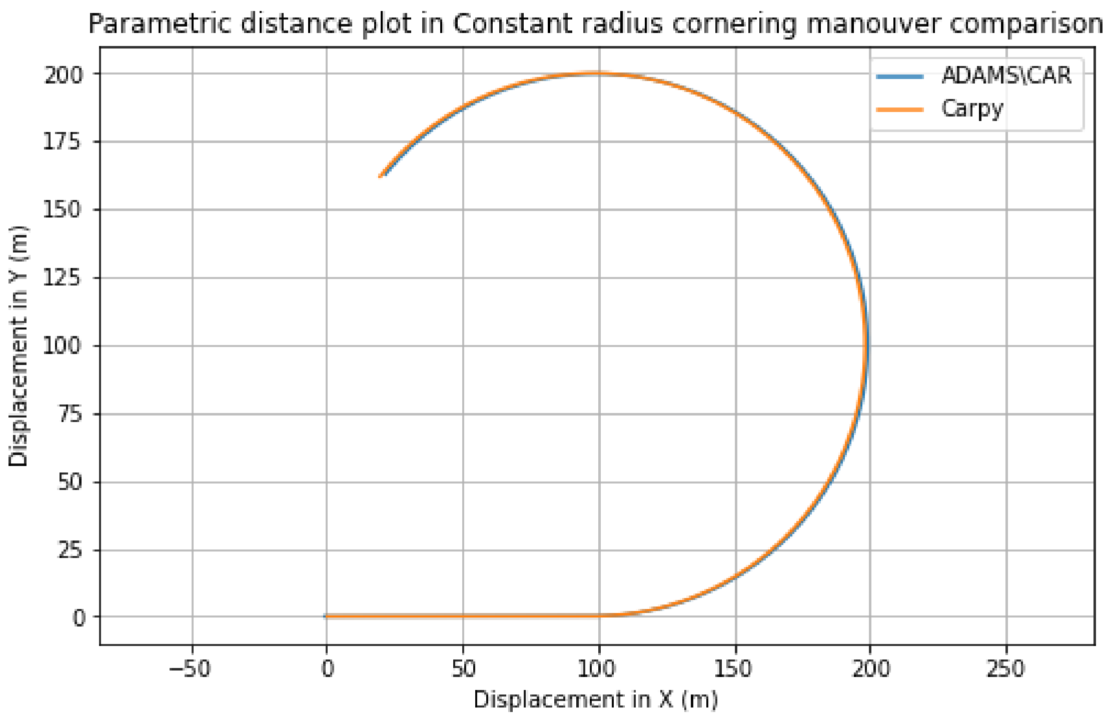
2.3.5 Yaw angle

[ ]:
```
plt.figure(figsize=(8.09, 5))
plt.plot(t_a, f_yaw(t_a), label='adams')
plt.plot(t_a, sim.psi(t_c), label='carpy')
plt.legend(['ADAMS\CAR', 'Carpy'])
plt.xlabel('Time (s)')
plt.ylabel('Yaw angle (deg)')
plt.title('Yaw angle in Constant radius cornering manouver comparison')
plt.grid()
plt.show()
```



2.3.6 XY parametric plot

[ ]:
```
plt.figure(figsize=(8.09, 5))
plt.plot(f_x(t_a), -f_y(t_a), label='adams')
plt.plot(sim.x(t_c), sim.y(t_c), label='carpy')
plt.legend(['ADAMS\CAR', 'Carpy'])
plt.xlabel('Displacement in X (m)')
plt.ylabel('Displacement in Y (m)')
plt.title('Parametric distance plot in Constant radius cornering manouver␣
 ↪comparison')
plt.axis('equal')
plt.grid()
plt.show()
```



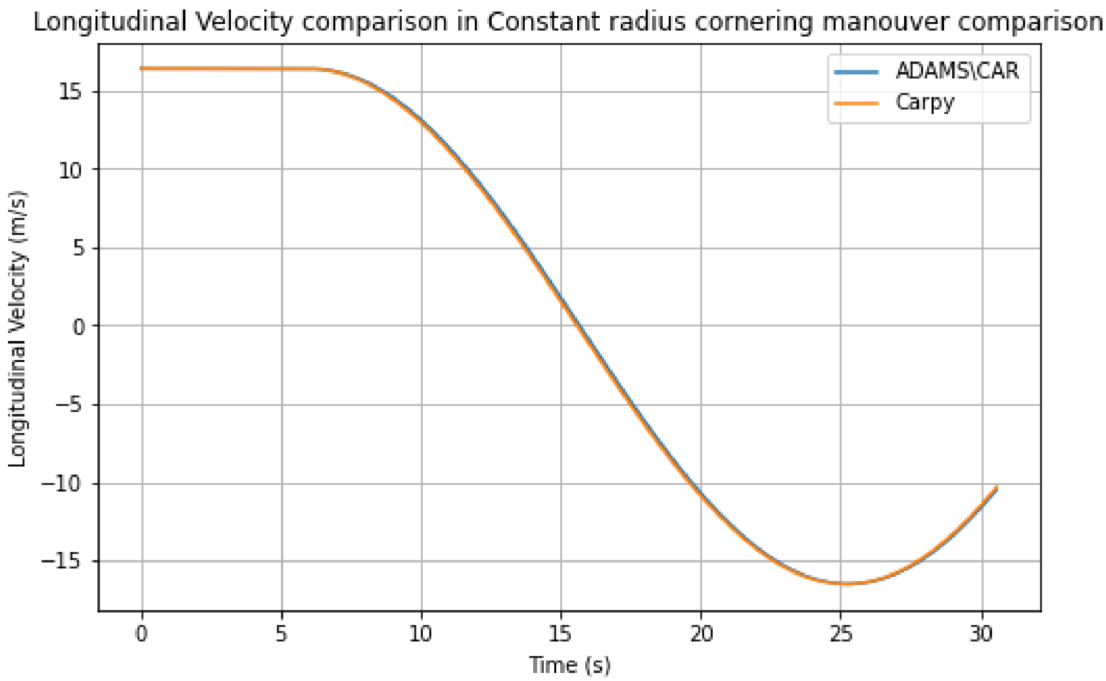
2.4 Velocities

[ ]:
```
tt = np.linspace(0, t_a[-1], 300)

vx = calcula_dot(f_x(tt), tt)
vy = calcula_dot(f_y(tt), tt)
vz = calcula_dot(f_z(tt), tt)

f_vx = Function(tt, vx, 'Time (s)', 'Longitudinal Velocity (m/s)', 'vx',␣
 ↪method='cubicSpline')
f_vy = Function(tt, vy, 'Time (s)', 'Lateral Velocity (m/s)', 'vy',␣
 ↪method='cubicSpline')
f_vz = Function(tt, vz, 'Time (s)', 'Vertical Velocity (m/s)', 'vz',␣
 ↪method='cubicSpline')
f_v = Function(tt, np.sqrt(vx**2 + vy**2 + vz**2), 'Time (s)', 'Velocity (m/
 ↪s)', 'v', method='cubicSpline')
```

2.4.1 Longitudinal velocity

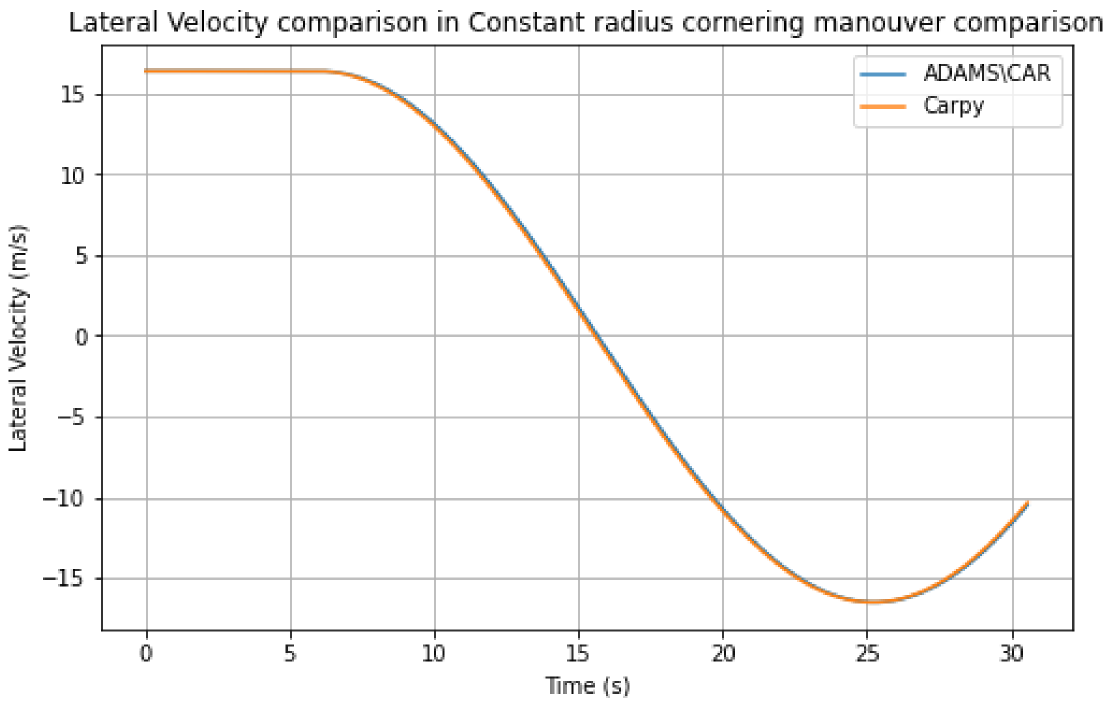
[ ]:
```
plt.figure(figsize=(8.09, 5))
plt.plot(t_a, f_vx(t_a), label='adams')
plt.plot(t_a, sim.vx(t_c), label='carpy')
plt.legend(['ADAMS\CAR', 'Carpy'])
```

```
plt.xlabel('Time (s)')
plt.ylabel('Longitudinal Velocity (m/s)')
plt.title('Longitudinal Velocity comparison in Constant radius cornering␣
 ↪manouver comparison')
plt.grid()
plt.show()
```



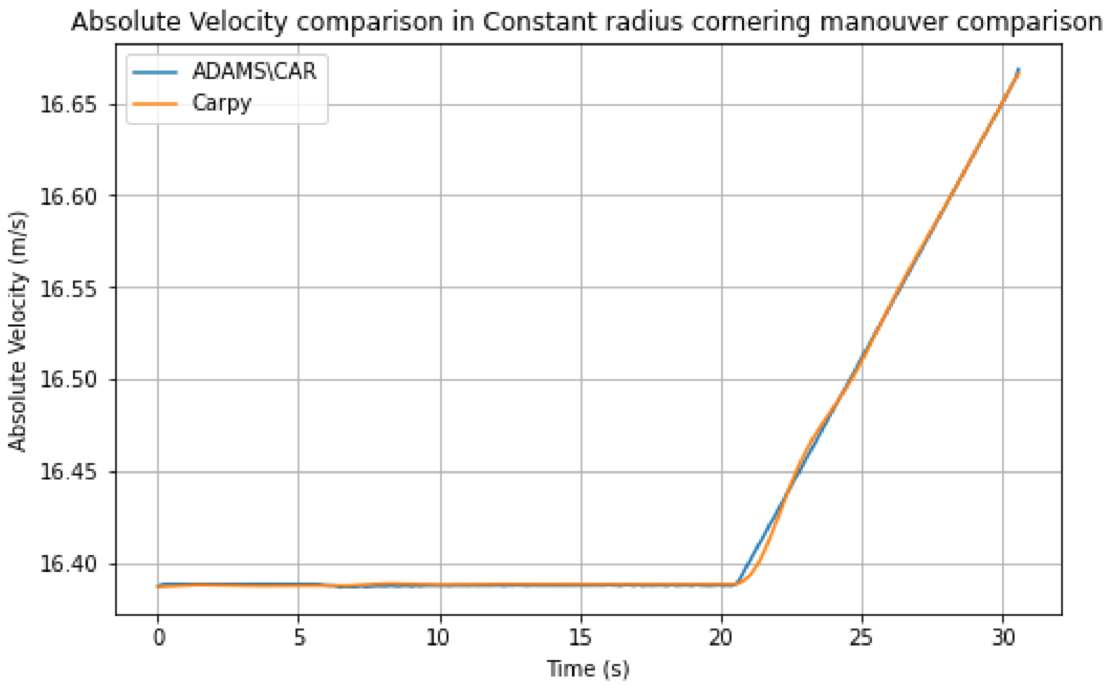
2.4.2 Lateral Velocity

[ ]:
```
plt.figure(figsize=(8.09, 5))
plt.plot(t_a, f_vx(t_a), label='adams')
plt.plot(t_a, sim.vx(t_c), label='carpy')
plt.legend(['ADAMS\CAR', 'Carpy'])
plt.xlabel('Time (s)')
plt.ylabel('Lateral Velocity (m/s)')
plt.title('Lateral Velocity comparison in Constant radius cornering manouver␣
 ↪comparison')
plt.grid()
plt.show()
```



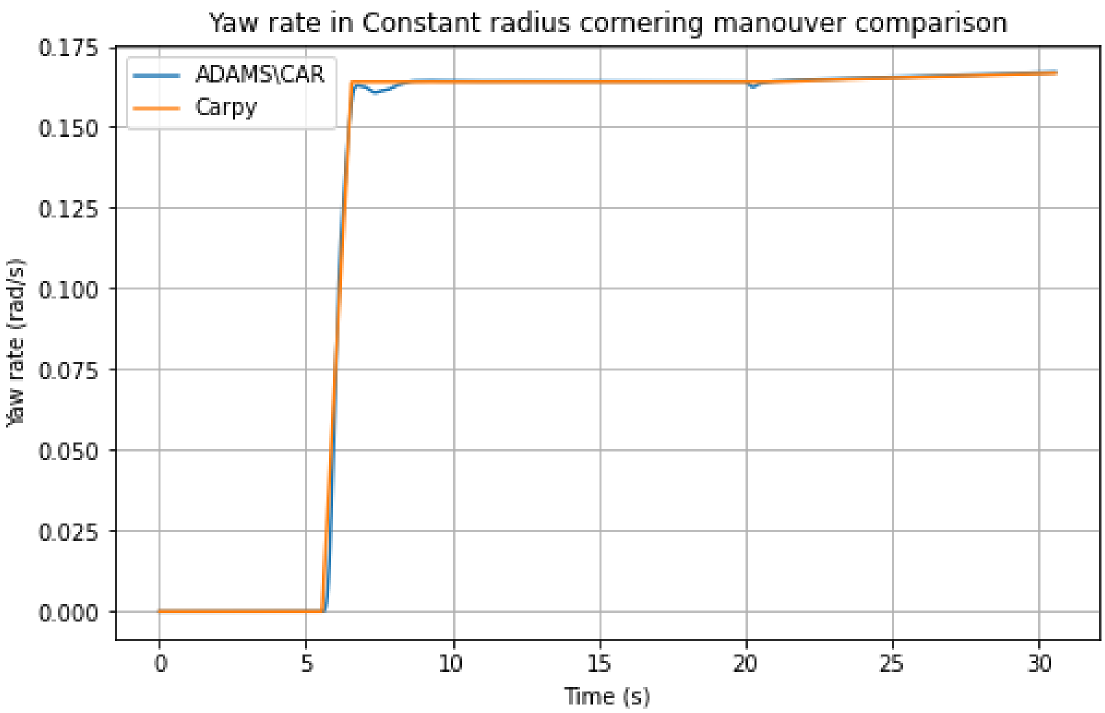
2.4.3 Absolute Velocity

```
[ ]: plt.figure(figsize=(8.09, 5))
     plt.plot(t_a, f_v(t_a), label='adams')
     plt.plot(t_a, np.sqrt(sim.vx(t_c)**2 + sim.vy(t_c)**2 + sim.vz(t_c)**2),␣
      ↪label='carpy')
     plt.legend(['ADAMS\CAR', 'Carpy'])
     plt.xlabel('Time (s)')
     plt.ylabel('Absolute Velocity (m/s)')
     plt.title('Absolute Velocity comparison in Constant radius cornering manouver␣
      ↪comparison')
     plt.grid()
     plt.show()
```



2.4.4 Yaw rate

[ ]:
```
d_yaw = np.deg2rad(calcula_dot(f_yaw(tt), tt))
f_d_yaw = Function(tt, d_yaw, 'Time (s)', 'Yaw velocity (rad/s)', 'd_yaw',␣
 ↪method='cubicSpline')

plt.figure(figsize=(8.09, 5))
plt.plot(t_a, f_d_yaw(t_a), label='adams')
plt.plot(t_a, sim.psi_dot(t_c), label='carpy')
plt.legend(['ADAMS\CAR', 'Carpy'])
plt.xlabel('Time (s)')
plt.ylabel('Yaw rate (rad/s)')
plt.title('Yaw rate in Constant radius cornering manouver comparison')
plt.grid()
plt.show()
```



[ ]:
```
# sim.animate(40, save=False)
```

[ ]:
```
# ax = calcula_dot_dot(f_x(tt), tt)
# ay = calcula_dot_dot(f_y(tt), tt)
# az = calcula_dot_dot(f_z(tt), tt)

# f_ax = Function(tt, ax, 'Time (s)', 'Longitudinal Acceleration (m/s²)', 'ax',␣
↪method='cubicSpline')
# f_ay = Function(tt, ay, 'Time (s)', 'Lateral Acceleration (m/s²)', 'ay',␣
↪method='cubicSpline')
# f_az = Function(tt, az, 'Time (s)', 'Vertical Acceleration (m/s²)', 'az',␣
↪method='cubicSpline')
```