EVER 2024 Autonomous Track

Institution /Team Identification (Assiut Motorsport)

Overview

For an open-loop control, we conducted a system that depends on a timer to take action. After making basic dynamic calculations, the time is determined. The system uses no feedback to make its decisions.
As a start, we tried to conduct the dynamic calculations for the car based on the given parameters. Then, we applied the results to the given workspace. At this moment we noticed a big difference between our calculations and the results obtained from CoppeliaSim. Thus, we made many trials to identify the problems. Our trials aimed to deeply understand the limits of the simulation environment and input parameters. Also, we observed the simulation while giving the max and min parameters. Then we tried closer inputs to the dynamic analysis, resulting in the final paths.
Moreover, we concluded that there is a specific steering angle parameter must be given to the simulation even in a straight line. Also, the laptop used to simulate can affect the performance of car simulation. The car will not follow the 4 paths with the given plane area, so we scaled the plane area. The performance of any simulation trail must be predicted based on vehicle dynamics formulas to be able to detect the right approach for solving the track. The simulation environment accepts the same input for changing the torque and velocity, which are inversely proportional parameters. So, we made our scenarios torque-based in some paths and velocity-based in others. More importantly, the timer set in Lua code was always flagged to one and it must be set to zero thus we made our external timer.
Finally, we are honored to participate in such a track for the first time in Egypt. And for each participant from our team, we polished our experience in ROS and Python.

Methodology Used

Dynamics Calculations

path 1:

we started with finding the maximum torque to be applied to wheels to prevent rear wheel slippage, which is critical for passenger safety. So, we used the following approach.

1. Total Tractive force:

- Rolling Resistance (RR): the force required to propel a vehicle over the surface.

Friction force per wheel = μ* N
= 0.9 * (600 * 9.81)/4 = **1324 Newton**
Which is the max force NOT to be exceeded by wheel torque.
Torque $_{max}$ = F* R $_{wheel}$
= 1324 * 0.34 = **450.3 Newton*meter**
Note: at this torque, the wheel is about to slip thus taking a reduction percentage by 5%.
Thus, safely applied torque to the wheel is about **425 Newton** as a start while accelerating from rest.
Now, let's calculate the force generated by this torque:
$F = T/R_{wheel}$
= (425 *2) / 0.34 = **2500 Newton**
This is the total force from to wheels to be applied to the whole system.

Knowing Newton's second law, we get the max acceleration:

$a_{max}=F/m$
= 2500 / 600 = **4.167 m/s²**

At a distance of 75 meters the car will accelerate the first half of the track and then decelerate the second half with the same force.
Using Kinematics equations:

$V_f{}^2 = V_i^2 + 2 * a * d$
V_f = (2*4.17*37.5)$^{1/2}$ m/s = **17.68 m/s**

This is the Max final velocity that the car can reach to achieve the required distance.

$V_f = V_i + a\,t$
17.68 = 0 + 4.167 * t
t = **4.24 s**
This is the time taken by the vehicle to reach half of the distance.
The total time taken to finish path 1 = **8.48s**

CoppeliaSim Implementation:

We made an outsource timer to send specific values to the simulation at precalculated time intervals.
Most Importantly, we set the gas pedal value while accelerating from rest at a max value of 0.034 to prevent wheel slippage at a start. This ensures we give the maximum force to our car system using the static coefficient of friction (0.9). after moving the car the full torque of the motors is used to reach max possible speed.
Then as time passes the limit of the gas pedal increases since the gas pedal parameter is a time-dependent function.
At the final stage of the path1 (braking), we sent the max value to reach the minimum time to finish the path.

Path 2:

In this path, we need to calculate the max velocity for the car taking into consideration the effect of centrifugal force.

Steering Angle:

Dealing with the bicycle model and the rear wheel as a reference point:

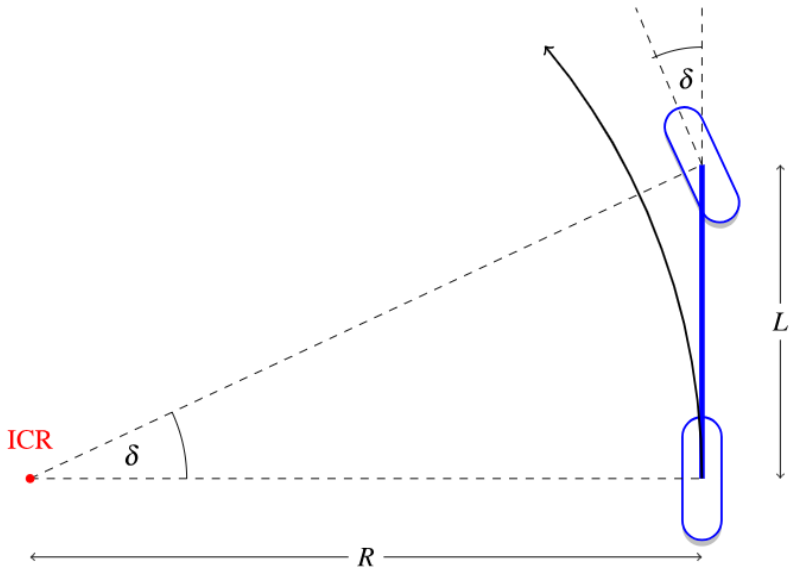

Figure 1

R = 6 + half of track width = 6+0.5*1.65 = 6.825 ≈ **6.8 m**

δ = arctan (2.269/6.8)

= 18.45 ≈ **18.5°**

Is the required steering angle to follow the path of a circle with a radius of 6m.

Max Velocity:

$V_{max} = \sqrt{\mu g r} = \sqrt{0.9 * 9.81 * (6 + 0.5 * 1.65)} = \mathbf{7.76\ \frac{m}{s}}$

μ: is the static coefficient of friction.

r: is the cornering radius from the cg of the car (6m path radius + half of the track width of the car).

The total distance:

d = 2 * π * r = 2 * π * 6.8 = **42.7 m**

total time:

$$t = \frac{42.7}{7.8} = \mathbf{5.5\ s}$$

Path 3:

Dynamic calculations

We followed the same procedures as path 1 to move in a straight line with a length of 75 m. then here are the calculations to change the lane. Maintaining the same velocity while changing the lanes. Finally decelerating to stop.

We noticed -in Lua code- that the brake force is given $1.311 * 10^{50}\ N.m$. While the max allowed to avoid slippage is 425 N.m. Thus, we put the brake padel $\mathbf{3.43 * 10^{-48}}$.

Steering Angle:

Assumptions used:

1. Based on the rear wheel axis as a reference point.
2. The path followed by the car to change the lane is symmetric about the line between the two lanes.

Using SolidWorks 2D sketch as shown in Figure (2), here is a sketch for the car path to change the lanes.

With a steering angle of **24°.**

Thus, we got a distance to leave the first lane of 7.5m with a total distance of **15m** to be centered in the second lane.

Kinematics Calculations:

Based on the data collected from Path 1. We found that the max speed the car reached was **6.7 m/s.** Substituting in the next equation we got:

$V_{max} = \sqrt{\mu g r} = 6.7$

$= \sqrt{0.9 * 9.81 * (r)}$

$= \mathbf{5\ m}$

This means even with no deceleration the car can take a corner of a 5m radius.

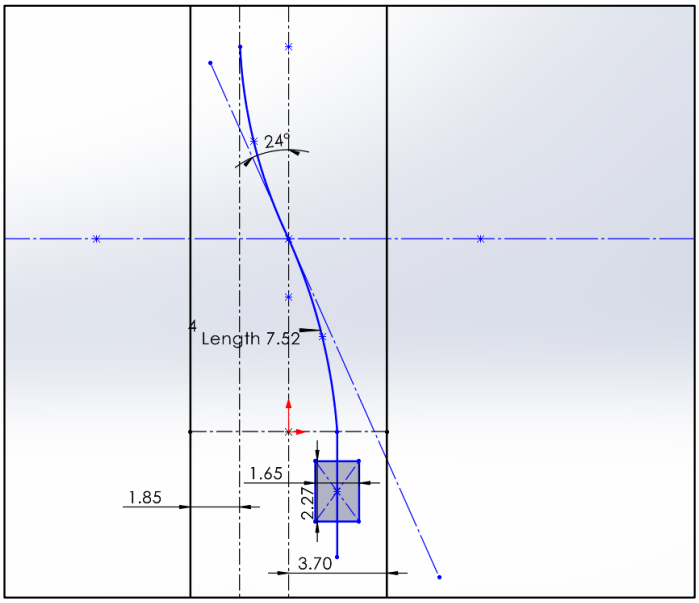

Figure 2

Time to change the lanes:

$$t_c = \frac{distance\ of\ curved\ lines}{\max speed} = \frac{7.5 * 2}{6.7} = 2.2\ \boldsymbol{s}$$

Finally, after reaching the center point of the second lane we started decelerating while maintaining the max deceleration to avoid slippage -also to avoid oversteering-.

a_{max} = -4 m/ s^2.

$$d = V_i t + 0.5 * a_{max} * t^2$$

$$75 = 6.7 * t + 0.5 * 4 * t^2$$

$$\boldsymbol{t_d = 4.6\ s}$$

Total time to finish the lane

$$\boldsymbol{t_d} = \mathbf{8}.48 + 2.2 + 4.6 = 15.28\ s$$

Path 4

path 4 actually was one of the hardest. Still enjoyable paths to make, there was different approaches to make that path, our first approach was to get the lemniscate of Bernoulli equation and turn it into a function in time, then derive it and get the velocity in X an Y axes, but that approach wasn't good enough, though we would try to use it again when we are going to be allowed for closed loop control, the other approach w took was to take path1 (line) and path 2 (circle) and combine them together, we also tried to use a cos(t) function in the steering to give our path that unique look of the curved infinity but it made the simulation take more time so we decided to relay on straight lines and semi half circles

ROS Approach:

Clock.py:

After many trials over CoppeliSim and ROS nodes, we noticed some problems with achieving the same result every time even with the same inputs. So, we approached our paths by creating a timer node that starts when the simulations begin. At the beginning the car is ordered to accelerate with a Then, after a certain period of time it sends a command to simulation to stop the vehicle, decelerate, or steer. With trial and error, we defined the time required to do our paths.

Vilocity.py

To store the collected position and time data and save it in a CSV file.

Built Tracks

Path 1:

*Figure **3***

Path 2:

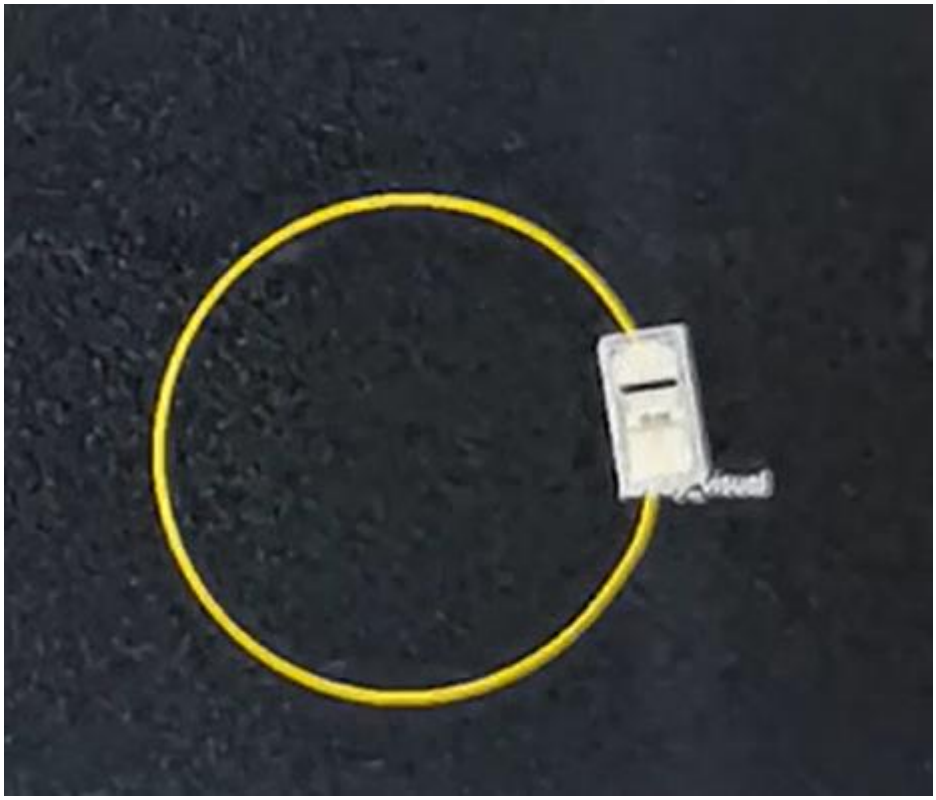

*Figure **4***

Path 3:

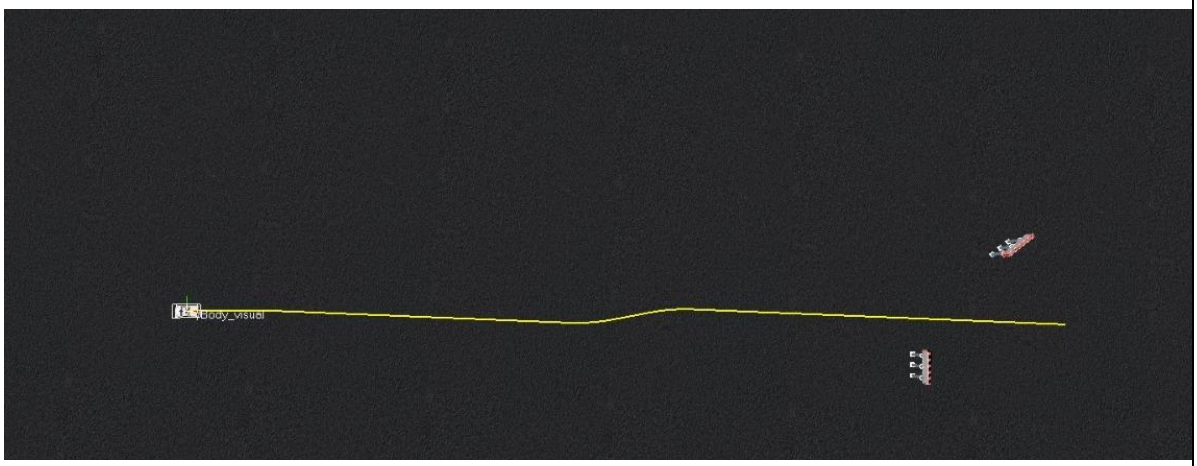

*Figure **5***

Path 4:

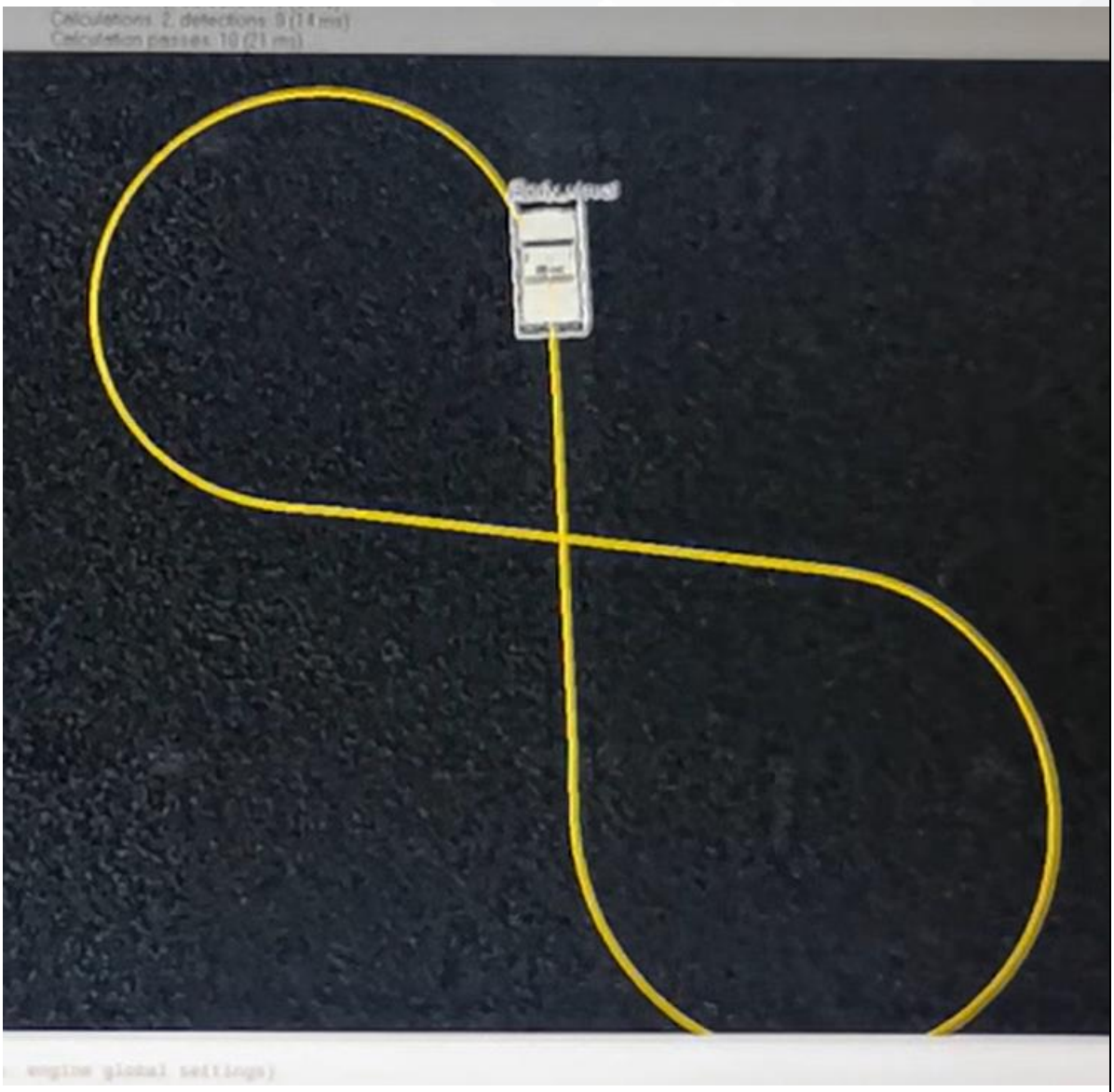

Figure 6

Results

path 1:

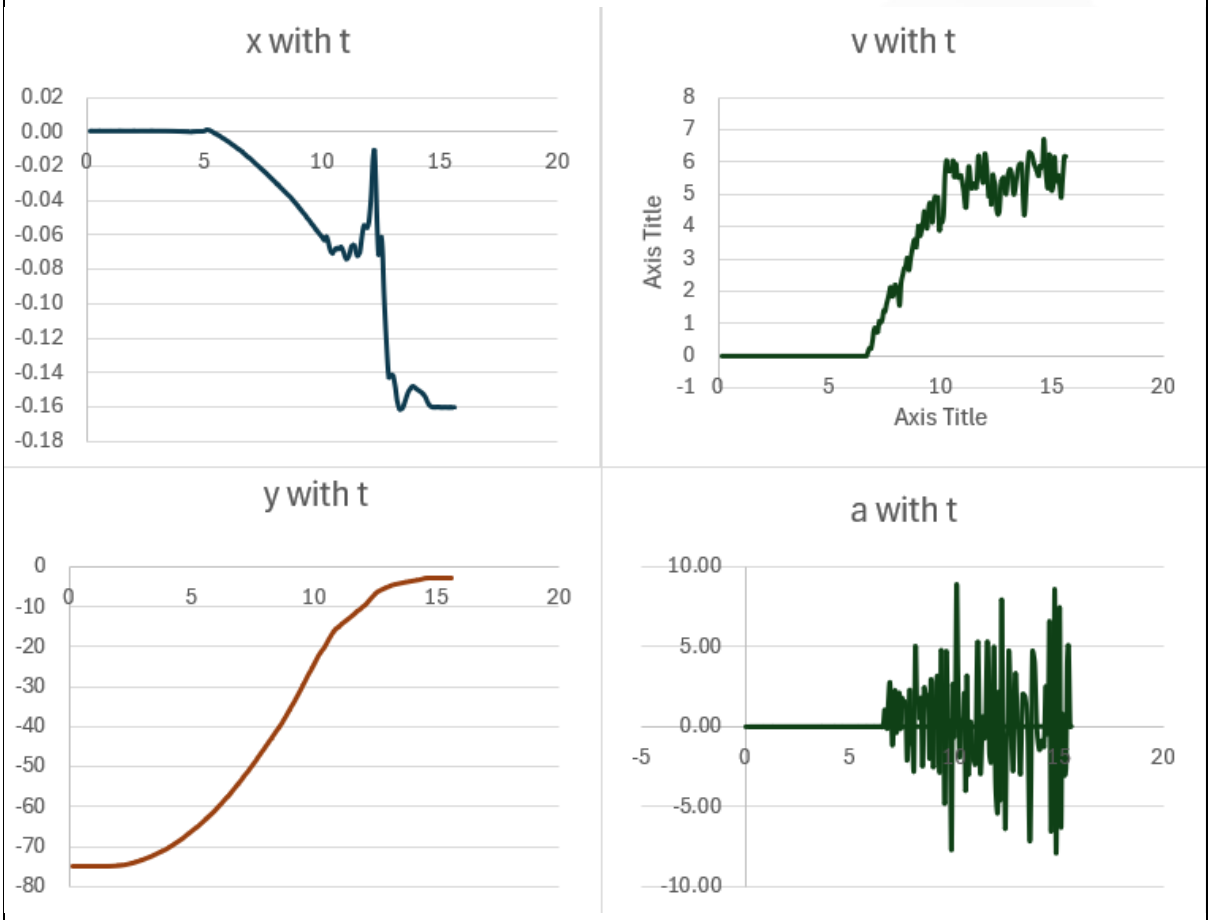

Figure 7

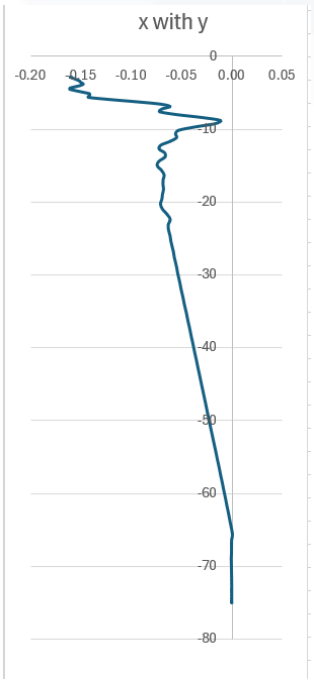

Figure 8

Path 2:

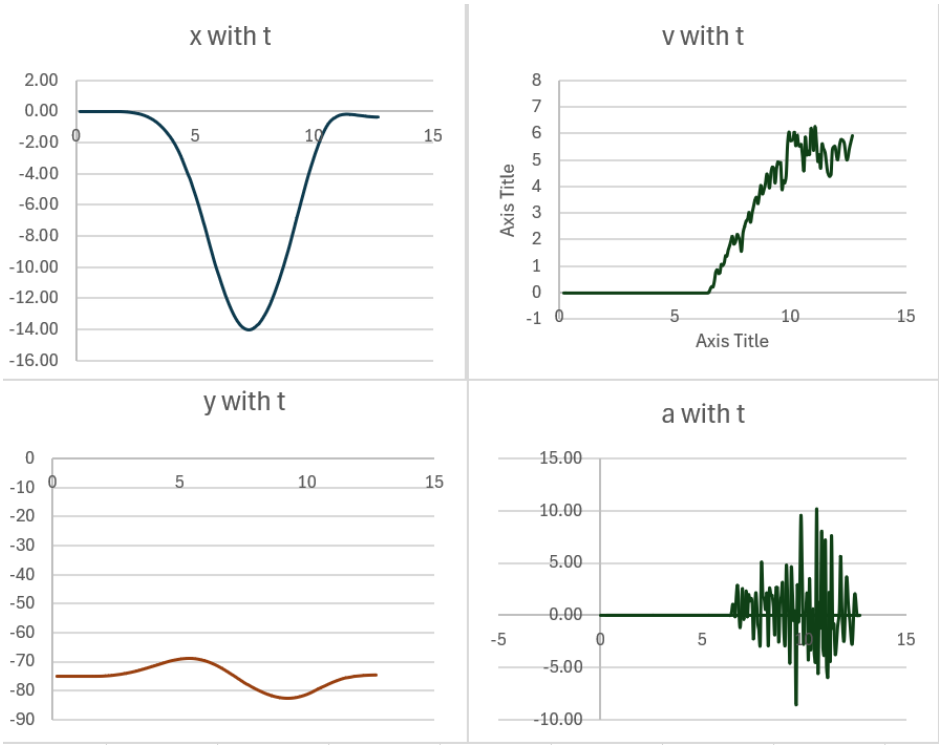

Figure 9

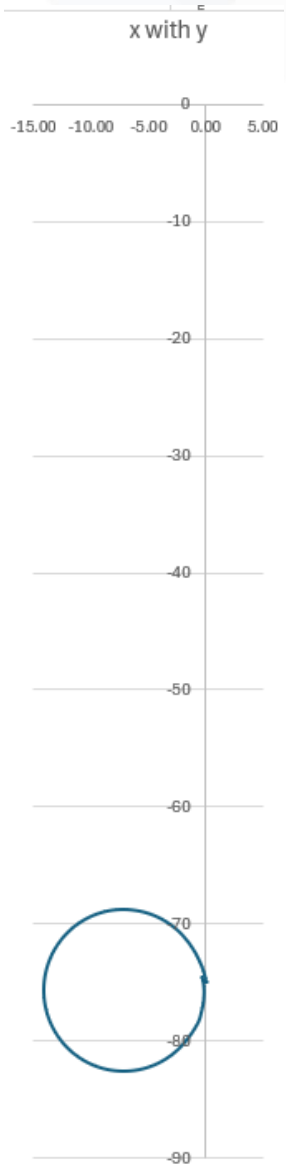

Figure 10

Path 3:

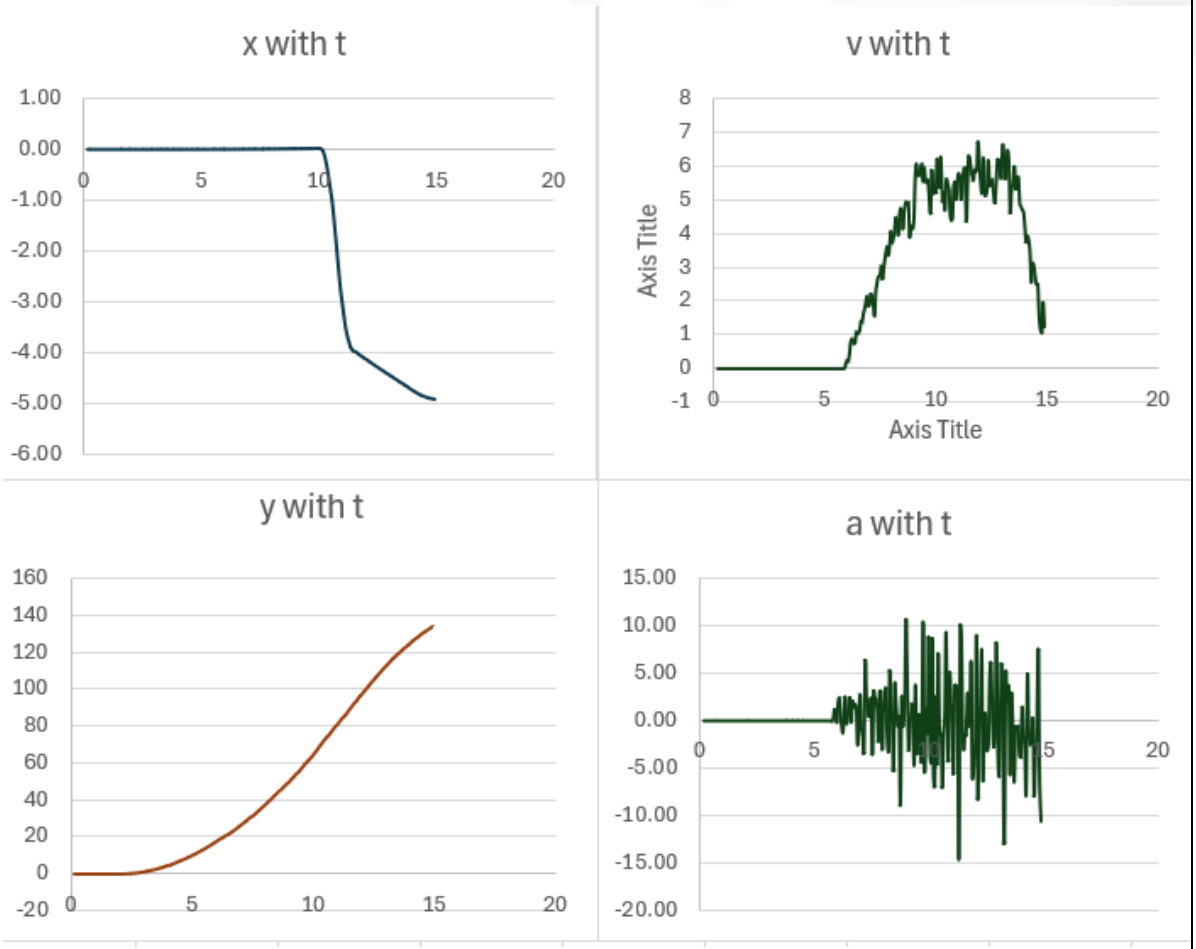

Figure 11

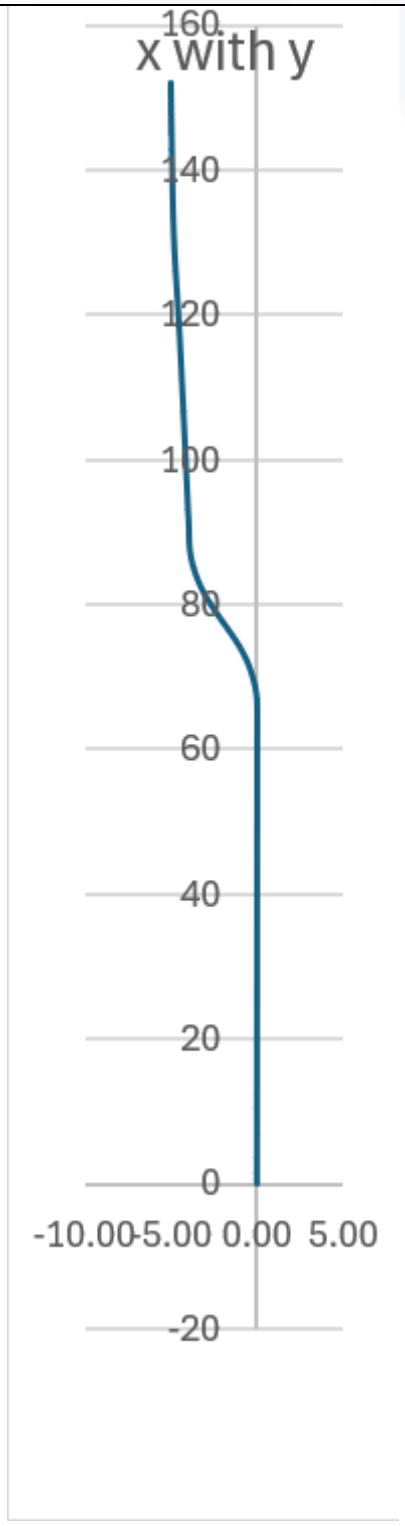

Figure 12

Path 4:

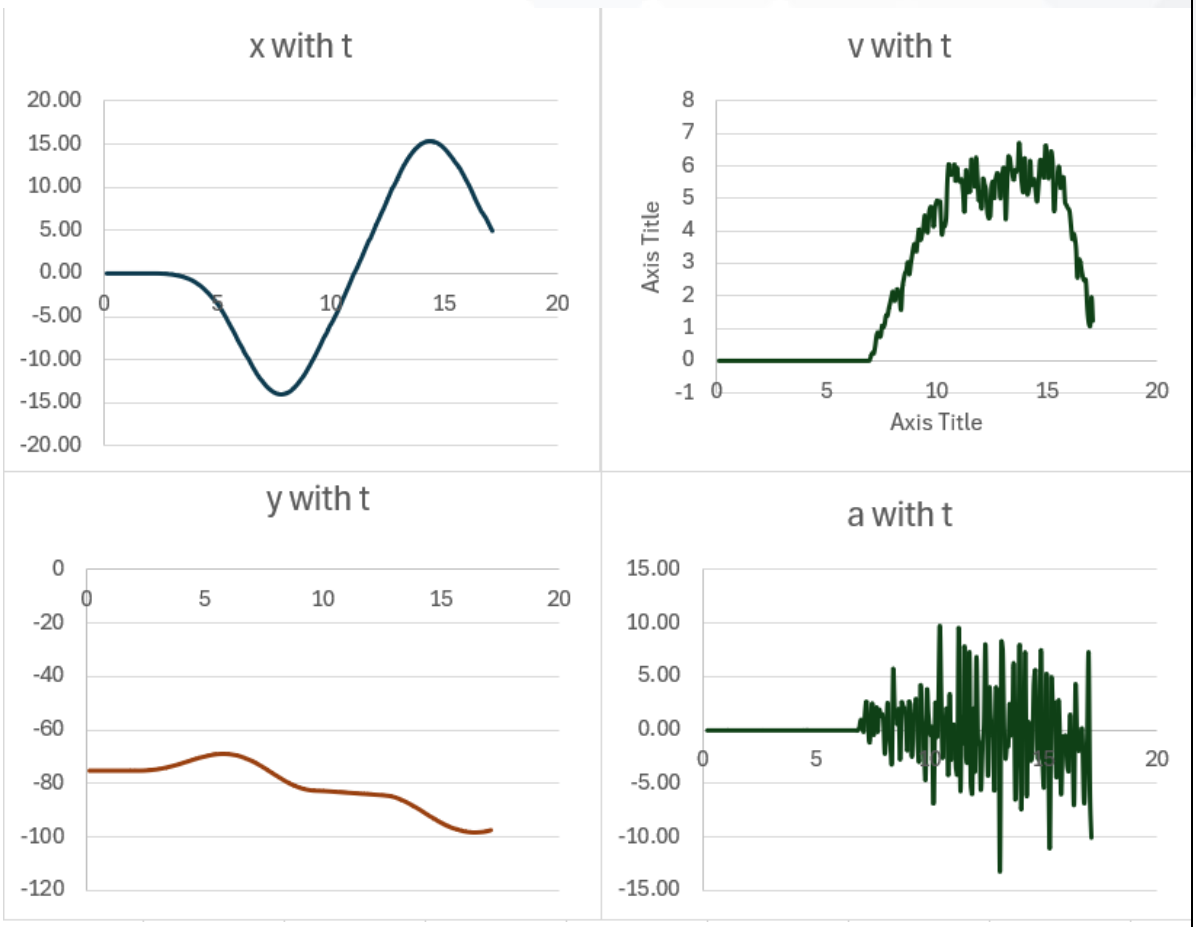

Figure 13

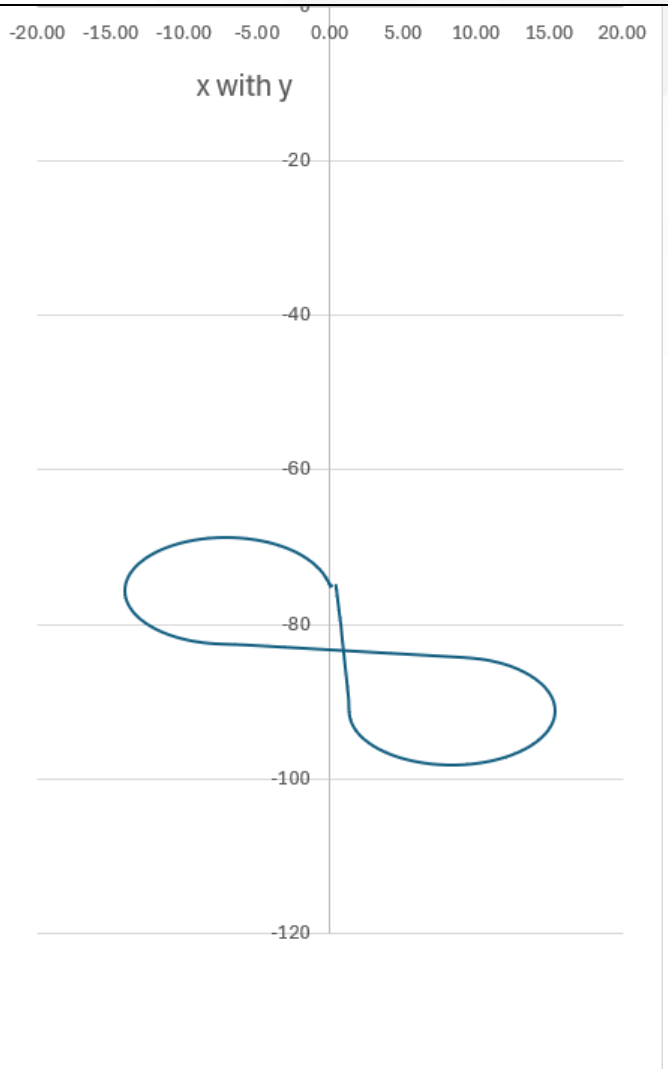
-20.00
-15.00
-10.00
-5.00
0.00
5.00
10.00
15.00
20.00
x with y
-20
-40
-60
-80
-100
-120

Figure 14

The Written Code

Path 1:

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import Float64
from std_msgs.msg import Int32
from rosgraph_msgs.msg import Clock
import time
import numpy as np
def brake():

     brake_control = rospy.Publisher("brakes", Float64, queue_size=10)

     time_brake = time.time()
     while (time.time()-start) >7.8  and (time.time()-start) <9 :
               brake_pedal = 1           # (time.time()- time_brake) *1.37* pow(10, -47)
               brake_control.publish(brake_pedal)
               steer_control.publish(angle)




def velocity():

        global steer_control
        steer_control = rospy.Publisher("SteeringAngle", Float64, queue_size=10)

        global angle
        angle = 0

        vel_control = rospy.Publisher("cmd_vel", Float64, queue_size= 10)
        rospy.init_node("control", anonymous= True)

        global start
        start = time.time()

        while (time.time()-start) <=7.8:
            gas_pedal = (((time.time()-start) ) *.15)
            vel_control.publish(gas_pedal)
            steer_control.publish(angle)

        gas_pedal =0
        vel_control.publish(gas_pedal)

        brake()

if __name__ == '__main__':

    try:
        velocity()
    except rospy.ROSInitException:
        rospy.loginfo("error occured")
```

Path 2:

```
#!/usr/bin/env python

import rospy
from std_msgs.msg import Float64
from std_msgs.msg import Int32
import time

start = time.time()



def control():


    rospy.init_node ('conrtol', anonymous=True)

    pub_SteeringAngle = rospy.Publisher('SteeringAngle', Float64, queue_size=10)
    pub_cmd_vel = rospy.Publisher('cmd_vel', Float64, queue_size=10)
    pub_brakes = rospy.Publisher('brakes', Float64, queue_size=10)



    while not rospy.is_shutdown():

        if ((time.time() - start) <=9 ):
            #rospy.loginfo("The car moving Forward")
            pub_SteeringAngle.publish(18.5)
            pub_cmd_vel.publish(0.23)
            pub_brakes.publish(0)

        elif ((time.time() - start) >9 ) and ((time.time() - start) <=9.5):
            pub_cmd_vel.publish(0)
            pub_brakes.publish(1)

        else:
            break


if __name__ == '__main__':
    try:
        control()
    except rospy.ROSInterruptException:
        pass
```

Path 3:

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import Float64
from std_msgs.msg import Int32
from rosgraph_msgs.msg import Clock
import time
import numpy as np


def velocity2():
    global x
    x=1
    brake()


def steer():
    rospy.loginfo("we are publishing steer ")
    gas_pedal = 0
    vel_control.publish(0)
    angle1 = 24
    angle2 = 0
    steer_control.publish(angle1)
    rospy.sleep(0.3)
    steer_control.publish(0)
    rospy.sleep(0.5)
    steer_control.publish(-angle1)
    rospy.sleep(0.3)
    steer_control.publish(0)
    rospy.sleep(1)
    velocity2()


def brake():
    rospy.loginfo("we are in brake")
    brake_control = rospy.Publisher("brakes", Float64, queue_size=10)
    time_brake = time.time()
    while(time.time()- time_brake <10):
        yyy = (time.time()- time_brake)
        rospy.loginfo("%i",yyy)
        brake_control.publish(yyy *3.43*pow(10,-48))
        steer_control.publish(0)




def velocity1():
        global steer_control
        steer_control = rospy.Publisher("SteeringAngle", Float64, queue_size=10)

        global angle
        angle =0

        global vel_control
        vel_control = rospy.Publisher("cmd_vel", Float64, queue_size= 10)
        rospy.init_node("control", anonymous= True)

        global start
        start = time.time()

        while (time.time()-start) <=8.5 :
            gas_pedal = (((time.time()-start) ) *.15)
            vel_control.publish(gas_pedal)
            steer_control.publish(angle)
            rospy.sleep(.5)
            rospy.loginfo("we are moving forward")

        if (time.time()-start) >7.1 :
            steer()

        gas_pedal =0
        vel_control.publish(gas_pedal)


if __name__ == '__main__':

    try:
        velocity1()
    except rospy.ROSInitException:
        rospy.loginfo("error occured")
```

Path 4:

```
#!/usr/bin/env python

import rospy
from std_msgs.msg import Float64
from std_msgs.msg import Int32
import time
import math as m

angle =0

def control():


    rospy.init_node ('conrtol', anonymous=True)

    pub_SteeringAngle = rospy.Publisher('SteeringAngle', Float64, queue_size=10)
    pub_cmd_vel = rospy.Publisher('cmd_vel', Float64, queue_size=10)
    pub_brakes = rospy.Publisher('brakes', Float64, queue_size=10)

    start = time.time()
    while (time.time()- start) <20:

        if ((time.time() - start) <= 8) :
                #curve
                pub_SteeringAngle.publish(18.5)
                pub_cmd_vel.publish(0.23)
                pub_brakes.publish(0)


        elif ((time.time() - start) > 8) and ((time.time() - start) <= 11)  :
                    #straight
                    pub_SteeringAngle.publish(0)
                    pub_cmd_vel.publish(0.23)
                    pub_SteeringAngle.publish(0)
                    pub_brakes.publish(0)

        elif ((time.time() - start) > 11) and ((time.time() - start) <= 16.9) :
                #curve
                pub_SteeringAngle.publish(-18.5)
                pub_cmd_vel.publish(0.23)
                pub_brakes.publish(0)


        elif ((time.time() - start) >16,9) and ((time.time() - start) <=18):
                #straight
                pub_SteeringAngle.publish(0)
                pub_cmd_vel.publish(0)




if __name__ == '__main__':
    try:
        control()
    except rospy.ROSInterruptException:
        pass
```

Conclusion

Regarding path 1, the results obtained from coppeliaSim are near the calculated. Finishing the path in almost 10 sec.

For path 2, in the first trials, we noticed that the car follows the circular path and then drifts. However, implementing vehicle dynamic inputs helped us to reach the exact required path. Finally, we adjusted the inputs to minimize the error between predicted performance and simulation. The car followed the path with a max speed of 7 m/s in almost 9.5 sec.

In the path3, we mixed the approach of the previous two paths, which made it easier for us to conduct the correct parameters. Also, we tried to minimize the time consumed to change the lane with the maximum safe speed. That left us to finish the first 75m in 8.48 sec reaching a max speed of 6.7 m/s, and changing the lane with the same speed. Reaching the left lane in 2.2 sec, finally finishing another 75m while decelerating for 4.6 sec.

In path 4 we pursuit that the car will follow a semi circle twice connected by a straight line.