EVER 2024 Autonomous Track

Institution /Team Identification..... **Zagazig /IEEE Zagazig SB** .....

Overview

In this milestone, we have successfully achieved the required objective by controlling the vehicle in a simulated environment with no feedback control. The process took some time from the team but nevertheless it was perfectly implemented.

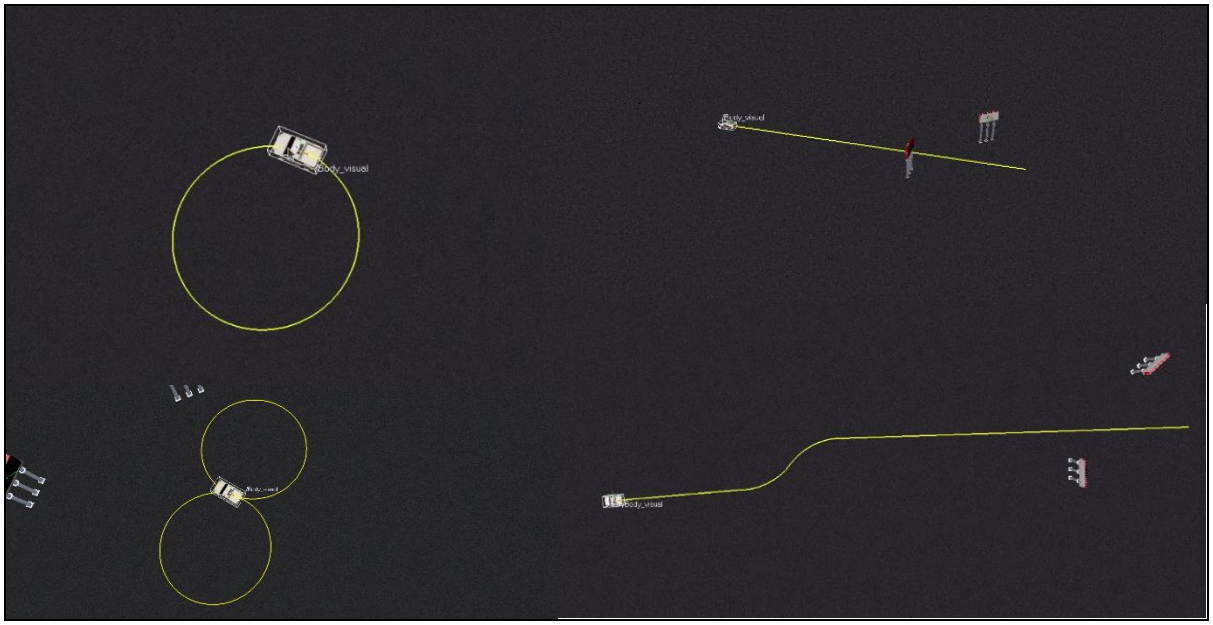
Methodology Used

First of all, we began studying vehicle controller types and what could be considered open-loop. Initially, we attempted using a Stanley controller with a predefined path. However, since Stanley requires feedback such as current position and acceleration, we handled the current position using bicycle model equations to calculate the vehicle's next position in X and Y. To obtain acceleration without IMU feedback, we explored two methods: deriving it from dynamics equations and measuring it by drawing graphs at constant pedal pressure. Unfortunately, both methods resulted in poor acceleration accuracy.

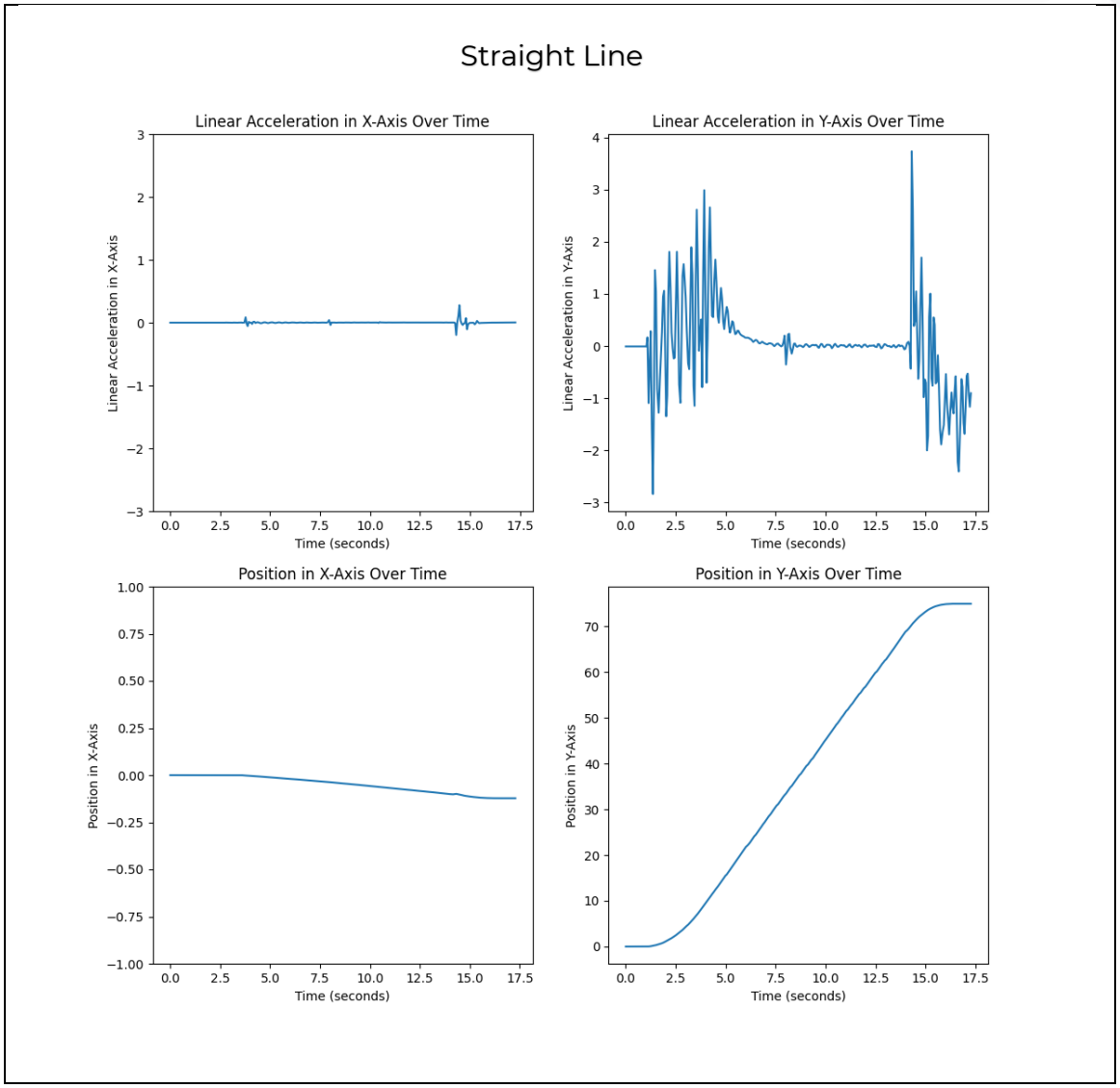
Consequently, we opted for a more challenging approach, simulating how anyone would drive a car in the real world with step-by-step instructions, like applying pressure on the pedal and adjusting steering angle. This method proved effective in achieving the desired shapes. To ensure accurate distances, we utilized multiple equations, such as R = L/tan(delta) and delta = 2Lsin(alpha)/ld. Drawing these shapes required introducing a mandatory thing which is the delay between instructions, which posed challenges in determining simulation time. To address this, we used a system delay and, through trial and error, successfully achieved accurate shapes.

It's essential to note that when running this on another PC, customization of delays is necessary, depending on the simulator's performance and real-time factor. Additionally, we utilized an odometry sensor to map the moving path, ultimately achieving our goal.

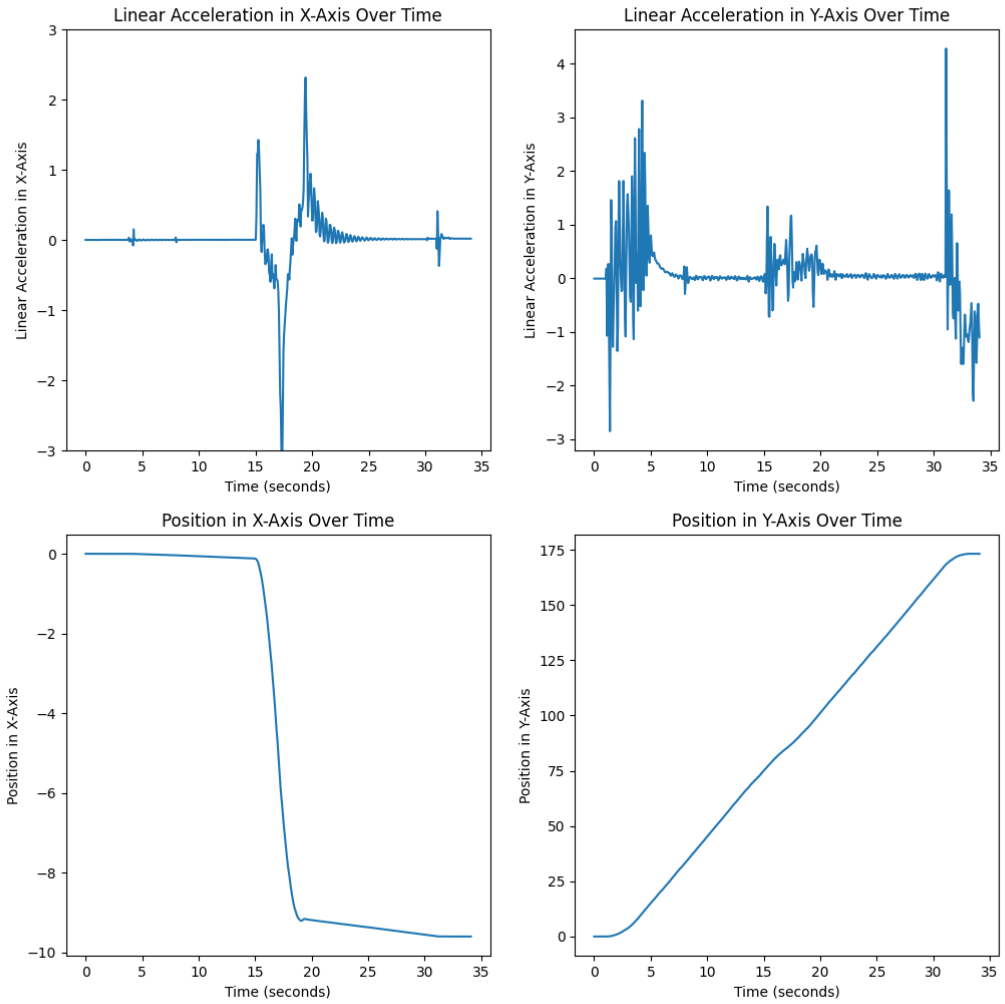
Built Tracks



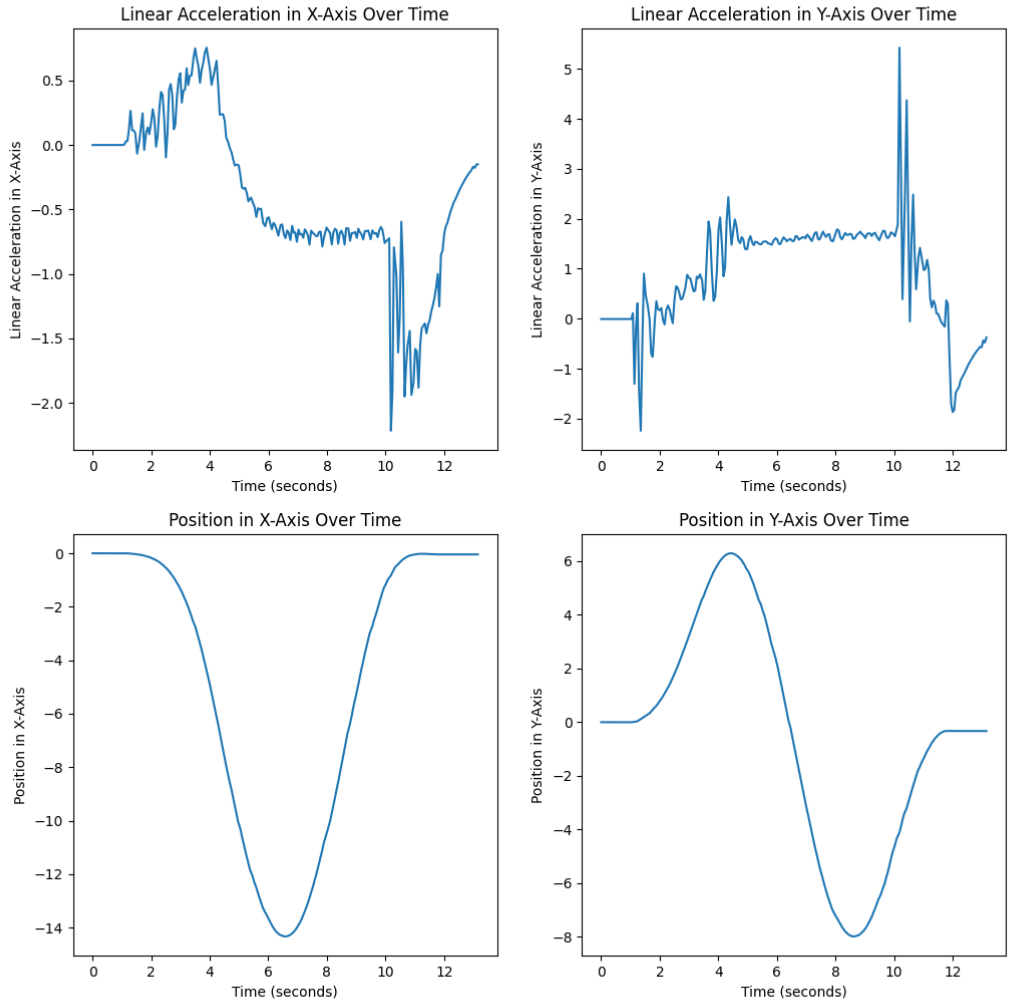
Results


Straight Line
Linear Acceleration in X-Axis Over Time
Linear Acceleration in X-Axis
Time (seconds)
Linear Acceleration in Y-Axis Over Time
Linear Acceleration in Y-Axis
Time (seconds)
Position in X-Axis Over Time
Position in X-Axis
Time (seconds)
Position in Y-Axis Over Time
Position in Y-Axis
Time (seconds)

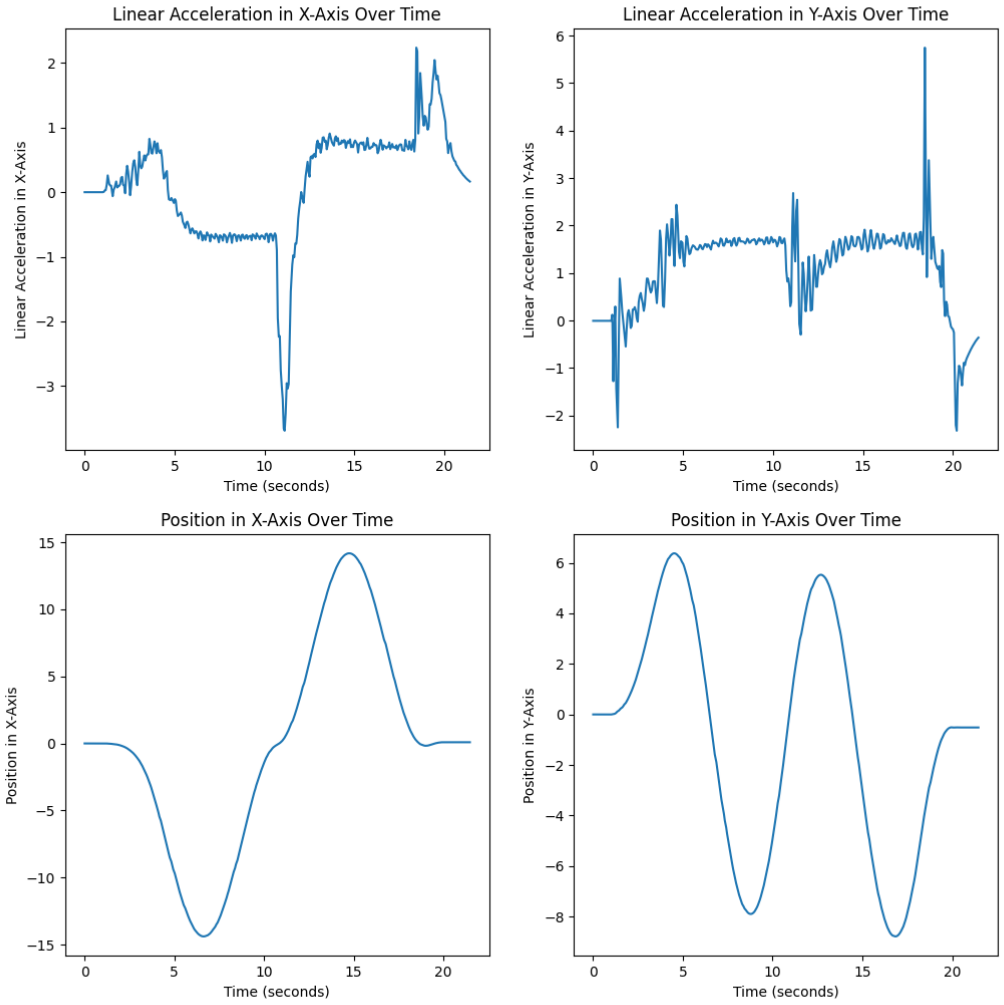
Switch Lane


Linear Acceleration in X-Axis Over Time
Linear Acceleration in X-Axis
Time (seconds)
Linear Acceleration in Y-Axis Over Time
Linear Acceleration in Y-Axis
Time (seconds)
Position in X-Axis Over Time
Position in X-Axis
Time (seconds)
Position in Y-Axis Over Time
Position in Y-Axis
Time (seconds)

One Circle


Linear Acceleration in X-Axis Over Time
Linear Acceleration in X-Axis
Time (seconds)
Linear Acceleration in Y-Axis Over Time
Linear Acceleration in Y-Axis
Time (seconds)
Position in X-Axis Over Time
Position in X-Axis
Time (seconds)
Position in Y-Axis Over Time
Position in Y-Axis
Time (seconds)

Two Circles


Linear Acceleration in X-Axis Over Time
Linear Acceleration in X-Axis
Time (seconds)
Linear Acceleration in Y-Axis Over Time
Linear Acceleration in Y-Axis
Time (seconds)
Position in X-Axis Over Time
Position in X-Axis
Time (seconds)
Position in Y-Axis Over Time
Position in Y-Axis
Time (seconds)

The Written Code

Straight Line

```python
#!/usr/bin/env python3

import rospy
from std_msgs.msg import Float64
import time

def straight_line():
    rospy.init_node('simple_car_controller', anonymous=True)

    # Define ROS publishers

    cmd_vel_pub = rospy.Publisher('/cmd_vel', Float64, queue_size=10)
    brakes_pub = rospy.Publisher('/brakes', Float64, queue_size=10)
    steering_pub = rospy.Publisher('/SteeringAngle', Float64, queue_size=10)

    rospy.sleep(1)  # Wait for publishers to register

    # Send gas pedal command and start turning
    brakes_msg = Float64()
    brakes_msg.data = 0.0
    brakes_pub.publish(brakes_msg)

    gas_pedal_msg = Float64()
    gas_pedal_msg.data = 0.2
    cmd_vel_pub.publish(gas_pedal_msg)

    steering_msg = Float64()
    steering_msg.data = 0.0  # Steering angle for turning (adjust as needed)
    steering_pub.publish(steering_msg)

    rospy.sleep(13.175)

    # Stop turning and apply brakes
    steering_pub.publish(Float64())  # Stop turning
    brakes_msg.data = 0.65
    cmd_vel_pub.publish(Float64())  # Stop gas pedal
    brakes_pub.publish(brakes_msg)


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

Switch Lane

```python
def send_commands():
    rospy.init_node('simple_car_controller', anonymous=True)
    # Define ROS publishers

    cmd_vel_pub = rospy.Publisher('/cmd_vel', Float64, queue_size=10)
    brakes_pub = rospy.Publisher('/brakes', Float64, queue_size=10)
    steering_pub = rospy.Publisher('/SteeringAngle', Float64, queue_size=10)
    rospy.sleep(1)  # Wait for publishers to register

    # Send gas pedal command and start turning
    brakes_msg = Float64()
    brakes_msg.data = 0.0
    brakes_pub.publish(brakes_msg)
    gas_pedal_msg = Float64()
    gas_pedal_msg.data = 0.2
    cmd_vel_pub.publish(gas_pedal_msg)
    steering_msg = Float64()
    steering_msg.data = 0.0  # Steering angle for turning (adjust as needed)
    steering_pub.publish(steering_msg)
    rospy.sleep(14)

    steering_msg = Float64()
    steering_msg.data = 9.0  # Steering angle for turning (adjust as needed)
    steering_pub.publish(steering_msg)
    rospy.sleep(2)

    steering_msg = Float64()
    steering_msg.data = 0.0  # Steering angle for turning (adjust as needed)
    steering_pub.publish(steering_msg)
    rospy.sleep(0.125)

    steering_msg = Float64()
    steering_msg.data = -9.0  # Steering angle for turning (adjust as needed)
    steering_pub.publish(steering_msg)
    rospy.sleep(2)

    brakes_msg = Float64()
    brakes_msg.data = 0.0
    brakes_pub.publish(brakes_msg)
    gas_pedal_msg = Float64()
    gas_pedal_msg.data = 0.2
    cmd_vel_pub.publish(gas_pedal_msg)
    steering_msg = Float64()
    steering_msg.data = 0.0  # Steering angle for turning (adjust as needed)
    steering_pub.publish(steering_msg)
    rospy.sleep(12)

    # Stop turning and apply brakes
    steering_pub.publish(Float64())  # Stop turning
    brakes_msg.data = 0.65
    cmd_vel_pub.publish(Float64())  # Stop gas pedal
    brakes_pub.publish(brakes_msg)
```

One Circle

```python
#!/usr/bin/env python3

import rospy
from std_msgs.msg import Float64
import time

def send_commands():
    rospy.init_node('simple_car_controller', anonymous=True)

    # Define ROS publishers
    cmd_vel_pub = rospy.Publisher('/cmd_vel', Float64, queue_size=10)
    brakes_pub = rospy.Publisher('/brakes', Float64, queue_size=10)
    steering_pub = rospy.Publisher('/SteeringAngle', Float64, queue_size=10)

    rospy.sleep(1)  # Wait for publishers to register

    # Send gas pedal command and start turning
    brakes_msg = Float64()
    brakes_msg.data = 0.0
    brakes_pub.publish(brakes_msg)

    gas_pedal_msg = Float64()
    gas_pedal_msg.data = 0.2
    cmd_vel_pub.publish(gas_pedal_msg)

    steering_msg = Float64()
    steering_msg.data = 17.9597315  # Steering angle for turning (adjust as needed)
    steering_pub.publish(steering_msg)

    rospy.sleep(9.1)  # Sleep for 2 seconds

    # Stop turning and apply brakes
    brakes_msg.data = 0.5211
    cmd_vel_pub.publish(Float64())  # Stop gas pedal
    brakes_pub.publish(brakes_msg)

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

Two Circle

```python
#!/usr/bin/env python3

import rospy
from std_msgs.msg import Float64
import time

def send_commands():
    rospy.init_node('simple_car_controller', anonymous=True)

    # Define ROS publishers
    cmd_vel_pub = rospy.Publisher('/cmd_vel', Float64, queue_size=10)
    brakes_pub = rospy.Publisher('/brakes', Float64, queue_size=10)
    steering_pub = rospy.Publisher('/SteeringAngle', Float64, queue_size=10)

    rospy.sleep(1)  # Wait for publishers to register

    # Send gas pedal command and start turning
    brakes_msg = Float64()
    brakes_msg.data = 0.0
    brakes_pub.publish(brakes_msg)

    gas_pedal_msg = Float64()
    gas_pedal_msg.data = 0.2
    cmd_vel_pub.publish(gas_pedal_msg)

    steering_msg = Float64()
    steering_msg.data = 17.9597315  # Steering angle for turning (adjust as needed)
    steering_pub.publish(steering_msg)

    rospy.sleep(9.6)  # Sleep for 2 seconds

    steering_msg = Float64()
    steering_msg.data = -17.9597315  # Steering angle for turning (adjust as needed)
    steering_pub.publish(steering_msg)

    rospy.sleep(7.7)  # Sleep for 2 seconds

    # Stop turning and apply brakes
    brakes_msg.data = 0.5211
    cmd_vel_pub.publish(Float64())  # Stop gas pedal
    brakes_pub.publish(brakes_msg)

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

Conclusion

Regarding the challenges, we have faced multiple problems installing the simulator at the beginning. later, we needed to integrate and find equations for each trajectory which enforced all the team to look for resources to understand the dynamics of the vehicle to output the most optimum result (that was not implemented as we are still understanding these).
Some of us then needed to work on the data analysis part to better understand our simulations and its results and generate graphs to better analyze the data.