**Python program using the CARLA simulator to perform an overtake maneuver of a moving vehicle in front of the ego vehicle**

**import carla**

**import random**

**import time**

**def overtake\_vehicle(world, ego\_vehicle, target\_vehicle):**

**# Get the current location of the ego vehicle**

**ego\_location = ego\_vehicle.get\_location()**

**# Get the current location of the target vehicle**

**target\_location = target\_vehicle.get\_location()**

**# Calculate the distance between the ego and target vehicles**

**distance = ego\_location.distance(target\_location)**

**# Get the velocity of the ego vehicle**

**ego\_velocity = ego\_vehicle.get\_velocity()**

**# If the ego vehicle is already in front of the target vehicle, return**

**if ego\_velocity.x > target\_vehicle.get\_velocity().x:**

**return**

**# Otherwise, calculate the time it would take to overtake the target vehicle**

**time\_to\_overtake = distance / (ego\_velocity.x - target\_vehicle.get\_velocity().x)**

**# Calculate the new location of the ego vehicle at the end of the overtake maneuver**

**new\_x = ego\_location.x + (ego\_velocity.x \* time\_to\_overtake) + (3 \* random.random())**

**new\_y = ego\_location.y**

**new\_z = ego\_location.z**

**new\_location = carla.Location(x=new\_x, y=new\_y, z=new\_z)**

**# Set the new location of the ego vehicle and wait for the maneuver to complete**

**ego\_vehicle.set\_location(new\_location)**

**time.sleep(time\_to\_overtake)**

**# Connect to the CARLA simulator**

**client = carla.Client('localhost', 2000)**

**client.set\_timeout(2.0)**

**# Load the town map**

**world = client.load\_world('Town01')**

**# Set the simulation settings**

**settings = world.get\_settings()**

**settings.fixed\_delta\_seconds = 0.05**

**settings.synchronous\_mode = True**

**world.apply\_settings(settings)**

**# Spawn the ego vehicle at a random location**

**spawn\_points = world.get\_map().get\_spawn\_points()**

**ego\_bp = random.choice(world.get\_blueprint\_library().filter('vehicle.tesla.model3'))**

**ego\_vehicle = world.spawn\_actor(ego\_bp, random.choice(spawn\_points))**

**# Spawn the target vehicle at a random location in front of the ego vehicle**

**target\_bp = random.choice(world.get\_blueprint\_library().filter('vehicle.nissan.micra'))**

**target\_spawn\_location = carla.Location(x=10, y=ego\_vehicle.get\_location().y, z=1)**

**target\_vehicle = world.try\_spawn\_actor(target\_bp, target\_spawn\_location)**

**# Run the simulation for 30 seconds**

**for i in range(600):**

**world.tick()**

**# Perform the overtake maneuver if the target vehicle is in front of the ego vehicle**

**if target\_vehicle and ego\_vehicle.get\_location().y < target\_vehicle.get\_location().y:**

**overtake\_vehicle(world, ego\_vehicle, target\_vehicle)**

**# Destroy the actors and stop the simulation**

**if target\_vehicle:**

**target\_vehicle.destroy()**

**if ego\_vehicle:**

**ego\_vehicle.destroy()**

**world.apply\_settings(carla.WorldSettings(synchronous\_mode=False))**

**2. import carla**

**def main():**

**# Connect to the Carla server**

**client = carla.Client('localhost', 2000)**

**client.set\_timeout(2.0)**

**# Load a map**

**world = client.load\_world('Town01')**

**# Create a blueprint for a vehicle to spawn**

**vehicle\_blueprint = world.get\_blueprint\_library().find('vehicle.tesla.model3')**

**# Set the initial location and orientation of the vehicle**

**spawn\_point = carla.Transform(carla.Location(x=100.0, y=50.0, z=2.0), carla.Rotation(yaw=180.0))**

**# Spawn the vehicle**

**vehicle = world.spawn\_actor(vehicle\_blueprint, spawn\_point)**

**# Set the target speed of the vehicle**

**target\_speed = 50 # km/h**

**vehicle.set\_target\_speed(target\_speed)**

**# Get the location of the vehicle in front**

**vehicle\_in\_front = world.get\_actor(0)**

**location\_in\_front = vehicle\_in\_front.get\_location()**

**# Calculate the location of the overtake maneuver**

**location\_overtake = carla.Location(x=location\_in\_front.x - 20.0, y=location\_in\_front.y + 10.0, z=2.0)**

**# Set the destination of the vehicle to the overtake location**

**vehicle.set\_destination(location\_overtake)**

**# Wait for the vehicle to reach the overtake location**

**while vehicle.get\_location().distance(location\_overtake) > 2.0:**

**world.tick()**

**# Perform the overtake maneuver**

**vehicle.set\_target\_speed(target\_speed \* 1.5)**

**vehicle.set\_destination(location\_in\_front)**

**# Wait for the vehicle to reach the location in front**

**while vehicle.get\_location().distance(location\_in\_front) > 2.0:**

**world.tick()**

**# Destroy the vehicle**

**vehicle.destroy()**

**if \_\_name\_\_ == '\_\_main\_\_':**

**main()**

Python program that uses the CARLA simulator to perform an overtake maneuver of a vehicle in front and then come back to the same lane after crossing

import carla

import random

import time

def overtake\_vehicle(world, ego\_vehicle, target\_vehicle):

# Get the current location of the ego vehicle

ego\_location = ego\_vehicle.get\_location()

# Get the current location of the target vehicle

target\_location = target\_vehicle.get\_location()

# Calculate the distance between the ego and target vehicles

distance = ego\_location.distance(target\_location)

# Get the velocity of the ego vehicle

ego\_velocity = ego\_vehicle.get\_velocity()

# If the ego vehicle is already in front of the target vehicle, return

if ego\_velocity.x > target\_vehicle.get\_velocity().x:

return

# Otherwise, calculate the time it would take to overtake the target vehicle

time\_to\_overtake = distance / (ego\_velocity.x - target\_vehicle.get\_velocity().x)

# Calculate the new location of the ego vehicle at the end of the overtake maneuver

new\_x = ego\_location.x + (ego\_velocity.x \* time\_to\_overtake) + 5.0

new\_y = ego\_location.y

new\_z = ego\_location.z

new\_location = carla.Location(x=new\_x, y=new\_y, z=new\_z)

# Set the new location of the ego vehicle and wait for the maneuver to complete

ego\_vehicle.set\_location(new\_location)

time.sleep(time\_to\_overtake)

# Change lanes

ego\_control = ego\_vehicle.get\_control()

ego\_control.steer = -1.0

ego\_vehicle.apply\_control(ego\_control)

time.sleep(1.0)

# Wait for the target vehicle to be behind the ego vehicle

while target\_vehicle.get\_location().y < ego\_vehicle.get\_location().y:

world.tick()

# Change lanes again

ego\_control = ego\_vehicle.get\_control()

ego\_control.steer = 0.0

ego\_vehicle.apply\_control(ego\_control)

time.sleep(1.0)

def main():

# Connect to the CARLA simulator

client = carla.Client('localhost', 2000)

client.set\_timeout(2.0)

# Load the town map

world = client.load\_world('Town01')

# Set the simulation settings

settings = world.get\_settings()

settings.fixed\_delta\_seconds = 0.05

settings.synchronous\_mode = True

world.apply\_settings(settings)

# Spawn the ego vehicle at a random location

spawn\_points = world.get\_map().get\_spawn\_points()

ego\_bp = random.choice(world.get\_blueprint\_library().filter('vehicle.tesla.model3'))

ego\_vehicle = world.spawn\_actor(ego\_bp, random.choice(spawn\_points))

# Spawn the target vehicle at a random location in front of the ego vehicle

target\_bp = random.choice(world.get\_blueprint\_library().filter('vehicle.nissan.micra'))

target\_spawn\_location = carla.Location(x=10, y=ego\_vehicle.get\_location().y, z=1)

target\_vehicle = world.try\_spawn\_actor(target\_bp, target\_spawn\_location)

# Run the simulation for 60 seconds

for i in range(1200):

world.tick()

# Perform the overtake maneuver if the target vehicle is in front of the ego vehicle

if target\_vehicle and ego\_vehicle.get\_location().y < target\_vehicle.get\_location().y:

overtake\_vehicle(world, ego\_vehicle, target\_vehicle)

# Destroy the actors and stop the simulation

if target\_vehicle:

target\_vehicle.destroy()

if

Python program that uses the CARLA simulator to perform a smoother overtake maneuver of a vehicle in front and then come back to the same lane after crossing

import carla

import random

import time

import math

def overtake\_vehicle(world, ego\_vehicle, target\_vehicle):

# Get the current location of the ego vehicle

ego\_location = ego\_vehicle.get\_location()

# Get the current location of the target vehicle

target\_location = target\_vehicle.get\_location()

# Calculate the distance between the ego and target vehicles

distance = ego\_location.distance(target\_location)

# Get the velocity of the ego vehicle

ego\_velocity = ego\_vehicle.get\_velocity()

# If the ego vehicle is already in front of the target vehicle, return

if ego\_velocity.x > target\_vehicle.get\_velocity().x:

return

# Otherwise, calculate the time it would take to overtake the target vehicle

time\_to\_overtake = distance / (ego\_velocity.x - target\_vehicle.get\_velocity().x)

# Calculate the new location of the ego vehicle at the end of the overtake maneuver

new\_x = ego\_location.x + (ego\_velocity.x \* time\_to\_overtake) + 10.0

new\_y = ego\_location.y

new\_z = ego\_location.z

new\_location = carla.Location(x=new\_x, y=new\_y, z=new\_z)

# Set the new location of the ego vehicle and wait for the maneuver to complete

ego\_vehicle.set\_location(new\_location)

time.sleep(time\_to\_overtake)

# Calculate the new angle of the ego vehicle after the turn

target\_x = target\_location.x

target\_y = target\_location.y

target\_angle = math.atan2(target\_y - ego\_location.y, target\_x - ego\_location.x)

# Turn smoothly to the new angle

current\_angle = math.radians(ego\_vehicle.get\_transform().rotation.yaw)

angle\_diff = target\_angle - current\_angle

# Use the sign of the angle difference to determine which direction to turn

direction = 1.0 if angle\_diff >= 0 else -1.0

# Determine the maximum steering angle based on the speed of the vehicle

speed = ego\_vehicle.get\_velocity().x

max\_steering\_angle = math.radians(40 / (1 + math.exp(-speed / 5)) + 5)

# Determine the steering angle based on the angle difference and the maximum steering angle

steering\_angle = direction \* min(max(abs(angle\_diff), 0.01), max\_steering\_angle)

# Set the control of the ego vehicle to turn smoothly to the new angle

ego\_control = carla.VehicleControl()

ego\_control.throttle = 0.6

ego\_control.steer = steering\_angle

ego\_control.brake = 0.0

ego\_vehicle.apply\_control(ego\_control)

# Wait for the turn to complete

time.sleep(1.0)

# Change lanes

ego\_control = carla.VehicleControl()

ego\_control.throttle = 0.6

ego\_control.steer = -1.0

ego\_vehicle.apply\_control(ego\_control)

time.sleep(2.0)

# Wait for the target vehicle to be behind the ego vehicle

while target\_vehicle.get\_location().y < ego\_vehicle.get\_location().y:

world.tick()

# Change lanes again

ego\_control = carla.VehicleControl()

ego\_control.throttle = 0.6

ego\_control.steer = 0.0

ego\_vehicle.apply\_control(ego\_control)

time.sleep(2.

2.

import carla

import random

import time

import math

def overtake\_vehicle(world, ego\_vehicle, target\_vehicle):

# Get the current location of the ego vehicle

ego\_location = ego\_vehicle.get\_location()

# Get the current location of the target vehicle

target\_location = target\_vehicle.get\_location()

# Calculate the distance between the ego and target vehicles

distance = ego\_location.distance(target\_location)

# Get the velocity of the ego vehicle

ego\_velocity = ego\_vehicle.get\_velocity()

# If the ego vehicle is already in front of the target vehicle, return

if ego\_velocity.x > target\_vehicle.get\_velocity().x:

return

# Otherwise, calculate the time it would take to overtake the target vehicle

time\_to\_overtake = distance / (ego\_velocity.x - target\_vehicle.get\_velocity().x)

# Calculate the new location of the ego vehicle at the end of the overtake maneuver

new\_x = ego\_location.x + (ego\_velocity.x \* time\_to\_overtake) + 5.0

new\_y = ego\_location.y

new\_z = ego\_location.z

new\_location = carla.Location(x=new\_x, y=new\_y, z=new\_z)

# Set the new location of the ego vehicle and wait for the maneuver to complete

ego\_vehicle.set\_location(new\_location)

time.sleep(time\_to\_overtake)

# Change lanes with smooth turn

steering\_angle = math.atan(0.5 / distance)

steering\_angle = steering\_angle if target\_vehicle.get\_location().y > ego\_vehicle.get\_location().y else -steering\_angle

ego\_control = ego\_vehicle.get\_control()

ego\_control.steer = steering\_angle

ego\_vehicle.apply\_control(ego\_control)

time.sleep(1.0)

# Wait for the target vehicle to be behind the ego vehicle

while target\_vehicle.get\_location().y < ego\_vehicle.get\_location().y:

world.tick()

# Change lanes again with smooth turn

ego\_control = ego\_vehicle.get\_control()

ego\_control.steer = 0.0

ego\_vehicle.apply\_control(ego\_control)

time.sleep(1.0)

def main():

# Connect to the CARLA simulator

client = carla.Client('localhost', 2000)

client.set\_timeout(2.0)

# Load the town map

world = client.load\_world('Town01')

# Set the simulation settings

settings = world.get\_settings()

settings.fixed\_delta\_seconds = 0.05

settings.synchronous\_mode = True

world.apply\_settings(settings)

# Spawn the ego vehicle at a random location

spawn\_points = world.get\_map().get\_spawn\_points()

ego\_bp = random.choice(world.get\_blueprint\_library().filter('vehicle.tesla.model3'))

ego\_vehicle = world.spawn\_actor(ego\_bp, random.choice(spawn\_points))

# Spawn the target vehicle at a random location in front of the ego vehicle

target\_bp = random.choice(world.get\_blueprint\_library().filter('vehicle.nissan.micra'))

target\_spawn\_location = carla.Location(x=10, y=ego\_vehicle.get\_location().y, z=1)

target\_vehicle = world.try\_spawn\_actor(target\_bp, target\_spawn\_location)

# Run the simulation for 60 seconds

for i in range(1200):

world.tick()

# Perform the overtake maneuver if the target vehicle is in front of the ego vehicle

if target\_vehicle and ego

3.

import carla

import random

import time

import math

def overtake\_vehicle(world, ego\_vehicle, target\_vehicle):

# Get the current location of the ego vehicle

ego\_location = ego\_vehicle.get\_location()

# Get the current location of the target vehicle

target\_location = target\_vehicle.get\_location()

# Calculate the distance between the ego and target vehicles

distance = ego\_location.distance(target\_location)

# Get the velocity of the ego vehicle

ego\_velocity = ego\_vehicle.get\_velocity()

# If the ego vehicle is already in front of the target vehicle, return

if ego\_velocity.x > target\_vehicle.get\_velocity().x:

return

# Otherwise, calculate the time it would take to overtake the target vehicle

time\_to\_overtake = distance / (ego\_velocity.x - target\_vehicle.get\_velocity().x)

# Calculate the new location of the ego vehicle at the end of the overtake maneuver

new\_x = ego\_location.x + (ego\_velocity.x \* time\_to\_overtake) + 10.0

new\_y = ego\_location.y

new\_z = ego\_location.z

new\_location = carla.Location(x=new\_x, y=new\_y, z=new\_z)

# Set the new location of the ego vehicle and wait for the maneuver to complete

ego\_vehicle.set\_location(new\_location)

time.sleep(time\_to\_overtake)

# Change lanes with a smooth turn

target\_transform = target\_vehicle.get\_transform()

target\_location = target\_transform.location

target\_yaw = target\_transform.rotation.yaw

ego\_transform = ego\_vehicle.get\_transform()

ego\_location = ego\_transform.location

ego\_yaw = ego\_transform.rotation.yaw

if target\_yaw >= 0:

if target\_yaw - ego\_yaw <= 180:

yaw\_difference = target\_yaw - ego\_yaw

else:

yaw\_difference = (target\_yaw - 360) - ego\_yaw

else:

if target\_yaw - ego\_yaw >= -180:

yaw\_difference = target\_yaw - ego\_yaw

else:

yaw\_difference = (target\_yaw + 360) - ego\_yaw

target\_lane\_location = carla.Location(x=target\_location.x + 10 \* math.cos(math.radians(target\_yaw)), y=target\_location.y + 10 \* math.sin(math.radians(target\_yaw)), z=target\_location.z)

ego\_lane\_location = carla.Location(x=ego\_location.x + 10 \* math.cos(math.radians(ego\_yaw)), y=ego\_location.y + 10 \* math.sin(math.radians(ego\_yaw)), z=ego\_location.z)

middle\_lane\_location = carla.Location(x=(target\_lane\_location.x + ego\_lane\_location.x) / 2, y=(target\_lane\_location.y + ego\_lane\_location.y) / 2, z=(target\_lane\_location.z + ego\_lane\_location.z) / 2)

ego\_control = ego\_vehicle.get\_control()

ego\_control.steer = yaw\_difference / 180.0

ego\_control.throttle = 1.0

ego\_vehicle.apply\_control(ego\_control)

time.sleep(1.0)

# Wait for the target vehicle to be behind the ego vehicle

while target\_vehicle.get\_location().y < ego\_vehicle.get\_location().y:

world.tick()

# Come back to the same lane with a smooth turn

ego\_control = ego\_vehicle.get\_control()

ego\_control.steer = 0.0

ego\_vehicle.apply\_control(ego\_control)

time.sleep(1.0)

ego\_location

4.

import carla

import random

import time

def overtake\_vehicle(world, ego\_vehicle, target\_vehicle):

# Get the current location of the ego vehicle

ego\_location = ego\_vehicle.get\_location()

# Get the current location of the target vehicle

target\_location = target\_vehicle.get\_location()

# Calculate the distance between the ego and target vehicles

distance = ego\_location.distance(target\_location)

# Get the velocity of the ego vehicle

ego\_velocity = ego\_vehicle.get\_velocity()

# If the ego vehicle is already in front of the target vehicle, return

if ego\_velocity.x > target\_vehicle.get\_velocity().x:

return

# Otherwise, calculate the time it would take to overtake the target vehicle

time\_to\_overtake = distance / (ego\_velocity.x - target\_vehicle.get\_velocity().x)

# Calculate the new location of the ego vehicle at the end of the overtake maneuver

new\_x = ego\_location.x + (ego\_velocity.x \* time\_to\_overtake) + 5.0

new\_y = ego\_location.y

new\_z = ego\_location.z

new\_location = carla.Location(x=new\_x, y=new\_y, z=new\_z)

# Set the new location of the ego vehicle and wait for the maneuver to complete

ego\_vehicle.set\_location(new\_location)

time.sleep(time\_to\_overtake)

# Change lanes

ego\_control = ego\_vehicle.get\_control()

ego\_control.steer = -0.2

ego\_control.throttle = 1.0

ego\_vehicle.apply\_control(ego\_control)

# Set a new location for the ego vehicle to smoothly transition back to the original lane

return\_location = carla.Location(x=ego\_location.x, y=ego\_location.y + 10.0, z=ego\_location.z)

new\_location = carla.Location(x=ego\_location.x + 20.0, y=return\_location.y, z=ego\_location.z)

# Move the ego vehicle smoothly to the new lane

i = 0

while i < 100:

# Calculate the current progress of the transition back to the original lane

progress = float(i) / 100.0

# Calculate the new location of the ego vehicle

lerp\_location = carla.Location(

x=(1.0 - progress) \* new\_location.x + progress \* return\_location.x,

y=(1.0 - progress) \* new\_location.y + progress \* return\_location.y,

z=(1.0 - progress) \* new\_location.z + progress \* return\_location.z,

)

# Set the new location of the ego vehicle and tick the simulation

ego\_vehicle.set\_location(lerp\_location)

world.tick()

# Increment the counter

i += 1

# Set the ego vehicle back to driving straight

ego\_control.steer = 0.0

ego\_vehicle.apply\_control(ego\_control)

# Wait for the target vehicle to be behind the ego vehicle

while target\_vehicle.get\_location().y < ego\_vehicle.get\_location().y:

world.tick()

# Wait for a short time to let the target vehicle get far enough behind

time.sleep(1.0)

def main():

# Connect to the CARLA simulator

client = carla.Client('localhost', 2000)

client.set\_timeout(2.0)

# Load the town map

world = client.load\_world('Town01