

## 实验目的

安装并熟悉 CARLA 使用方法，

实现路径规划算法

实现路径跟踪控制

## 实验方法

### 1.CARLA 的下载与安装

首先通过浏览器转到 CARLA 官网（<http://carla.org>）如图 1。

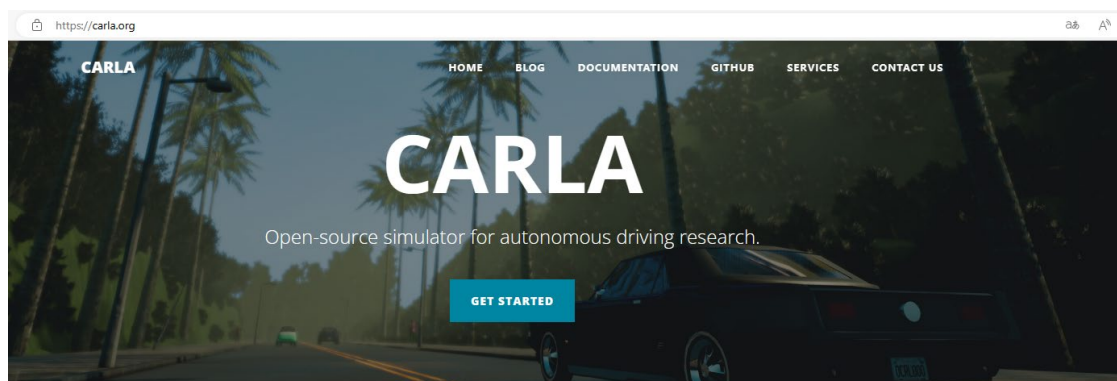


图 1

读者可在官网中可以查询到 CARLA 软件的相关功能、使用介绍和版本更新内容。此时需点击图中的“DOCUMENTATION”按键即可跳转到 CARLA 客户端相关界面，如图 2 所示。

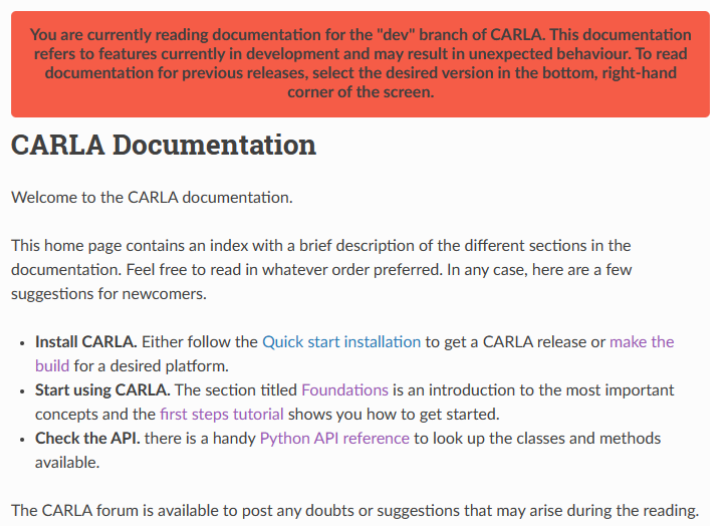


图 2

点击图 2 中的 **Quick start installation** 链接可以跳转到下载界面，如图三所示：

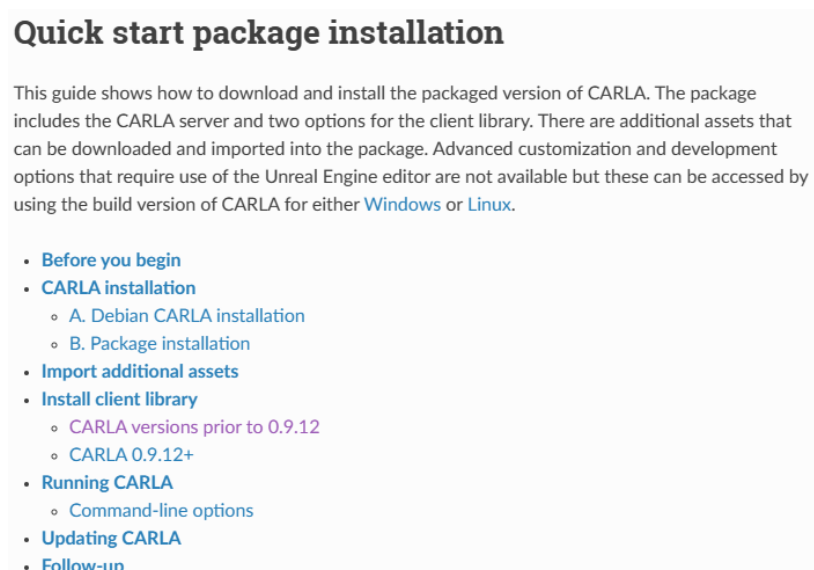


图 3

此时可以选择不同的 CARLA 版本下载，需注意，不同版本的 CARLA 对于系统配置有不同的要求，请在下载时请先查看官方文档。在此安装的 CARLA 版本为 0.9.14。

同时也可以通过进入如下 Github 网址(<https://github.com/carla-simulator/carla/blob/master/Docs/download.md>) 对 CARLA 进行下

载，如图 4 所示：

## Download

### Latest Release

- [CARLA 0.9.15 - Documentation](#)

### Nightly build

This is an automated build with the latest changes pushed to our `dev` branch. It contains the very latest fixes and features that will be part of the next release, but also some experimental changes. Use at your own risk!

- [CARLA Nightly Build \(Linux\)](#)
- [AdditionalMaps Nightly Build \(Linux\)](#)
- [CARLA Nightly Build \(Windows\)](#)
- [AdditionalMaps Nightly Build \(Windows\)](#)

### Versions 0.9.x

Here are the previous versions of CARLA with links to the specific documentation for each version:

- [CARLA 0.9.14 - Documentation](#)
- [CARLA 0.9.13 - Documentation](#)
- [CARLA 0.9.12 - Documentation](#)
- [CARLA 0.9.11 - Documentation](#)
- [CARLA 0.9.10 - Documentation](#)
- [CARLA 0.9.9 - Documentation](#)
- [CARLA 0.9.8 - Documentation](#)
- [CARLA 0.9.7 - Documentation](#)
- [CARLA 0.9.6 - Documentation](#)
- [CARLA 0.9.5 - Documentation](#)
- [CARLA 0.9.4 - Documentation](#)
- [CARLA 0.9.3 - Documentation](#)

图 4

点击上图框选进行下载，下载后解压到本地即可，解压完成后并不能立刻运行，需要下载其他环境，这里用的是 **python** 进行通讯交互，因此需要下载 **python** 相关依赖(这里采用的 **python** 版本为 3.7)，如图 5 所示；

```
1  #更新pip
2  python.exe -m pip install --upgrade pip
3
4  #安装依赖库
5  pip3 install carla
6  pip3 install pygame
7  pip3 install numpy
8  pip3 install networkx
9  pip3 install distro
10 pip3 install Shapely
```

图 5

在安装好依赖项后可以并运行 **CarlaUE4.exe**，如图 6 所示：

名称	修改日期	类型	大小
.vscode	2023/3/30 21:58	文件夹	
CarlaUE4	2022/12/24 8:12	文件夹	
Co-Simulation	2022/12/24 8:12	文件夹	
Engine	2022/12/24 8:12	文件夹	
HDMaps	2022/12/24 8:12	文件夹	
Plugins	2022/12/24 8:12	文件夹	
PythonAPI	2023/4/5 15:32	文件夹	
CarlaUE4.exe	2022/12/24 8:04	应用程序	189 KB
CHANGELOG	2022/12/24 6:28	文件	67 KB
Dockerfile	2022/12/24 6:28	文件	1 KB
LICENSE	2022/12/24 6:28	文件	2 KB

图 6

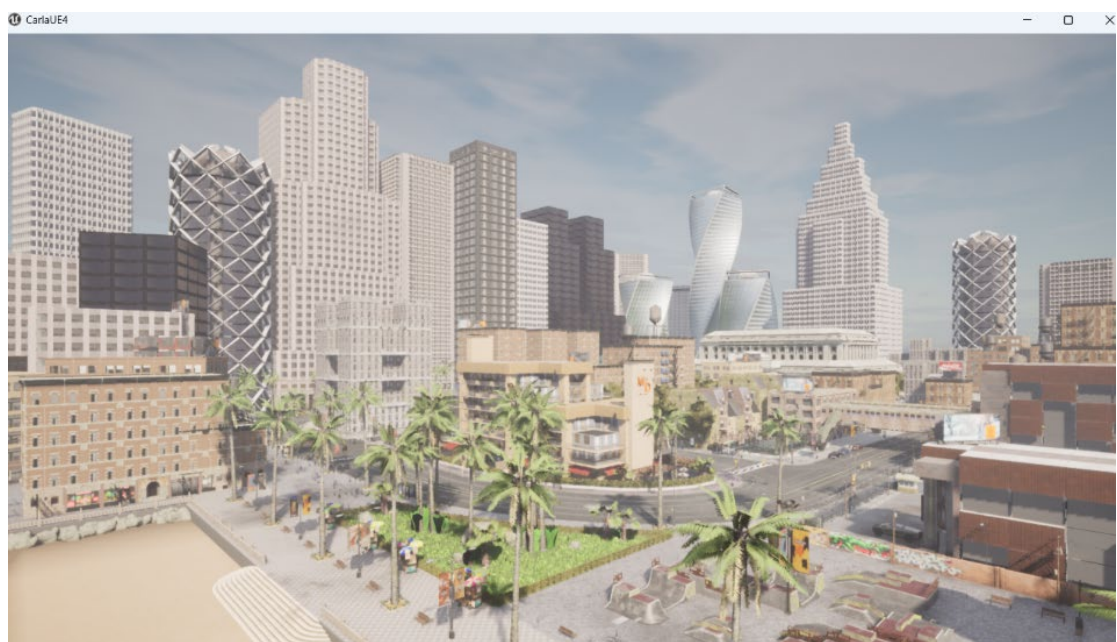


图 7

若运行例程的时候出现“No module name Carla”，则需要 pip 安装 carla-0.9.14-cp37-cp37m-win\_amd64.whl 到当前的 py 解释器中。注意这里是 py 3.7。如图 7 所示：

新加坡 (D:) > Carla > CARLA_0.9.14 > WindowsNoEditor > PythonAPI > carla > dist				
<div> <div> <div></div> <div>排序</div> </div> <div> <div></div> <div>查看</div> </div> <div>...</div> </div>				
名称	修改日期	类型	大小	
carla-0.9.14-cp37-cp37m-win_amd64.whl	2022/12/24 7:32	WHL 文件	4,633 KB	
carla-0.9.14-py3.7-win-amd64.egg	2022/12/24 7:32	EGG 文件	4,634 KB	

图 7

到此，CARLA 的安装与配置完成。

## 2.熟悉 CARLA 基本操作

在 CARLA 的官方文档中较为详细的介绍了 CARLA 的使用步骤，再此仅对 CARLA 的基础操作进行讲解。

打开 CarlaUE4.exe 之后，可以通过使用鼠标控制观众视图的俯仰和偏航以及 QWE-ASD 键移动视角来观察地图，图 8 即为移动后的客户端视角：



图 8

此时已经可以使用命令行、Windows 中的可执行文件或 Linux 中的 shell 脚本启动 CARLA。

要通过 Python API 操作 CARLA，我们需要通过开放端口将 Python 客户端连接到服务器。需将如下代码 Python main 文件的开头：

---

```
import carla
import random

# Connect to the client and retrieve the world object
client = carla.Client('localhost', 2000)
world = client.get_world()
```

---

CARLA 服务器通常加载默认地图（通常为 Town10）。如果要其他地图启动 CARLA，需要在 main 文件中添加如下代码：

---

```
world = client.load_world('Town01')
```

---

与此同时你也可以通过输入如下代码来查看可用地图：

---

```
print(client.get_available_maps())
```

---

此时地图已经选择好，可以通过如下代码查询 CARLA 中所带的所有车辆模型并生成：

---

```
# Get the blueprint library and filter for the vehicle blueprints
vehicle_blueprints=world.get_blueprint_library().filter('*vehicle
*')
```

---

每张 CARLA 地图都提供了预定义的生成点，这些生成点均匀分布在地图上的道路上。因此，我们需要在地图中预定义的生产点上生成车辆文件，具体代码如下：

---

```
# Get the map's spawn points

spawn_points = world.get_map().get_spawn_points()

# Spawn 50 vehicles randomly distributed throughout the map

# for each spawn point, we choose a random vehicle from the
blueprint library

for i in range(0,50):

    world.try_spawn_actor(random.choice(vehicle_blueprints),
random.choice(spawn_points))
```

---

此时需要添加一辆中心车辆，以作为仿真的中心点：

---

```
ego_vehicle=world.spawn_actor(random.choice(vehicle_bluepri
nts),random.choice(spawn_points))
```

---

下一步我们可以向中心车辆中添加传感器，在这里，我们将一个标准的摄像头传感器连接到自我车辆上，以记录一些视频数据具体代码如下：

---

```
# Create a transform to place the camera on top of the vehicle

camera_init_trans = carla.Transform(carla.Location(z=1.5))
```



# We create the camera through a blueprint that defines its properties

```
camera_bp=world.get_blueprint_library().find('sensor.camera.rgb')
```

# We spawn the camera and attach it to our ego vehicle

```
camera = world.spawn_actor(camera_bp, camera_init_trans, attach_to=ego_vehicle)
```

---

熟悉完 CARLA 的部分操作后，此时可以运行 CARLA 安装目录（CARLA\_0.9.14\WindowsNoEditor\PythonAPI\examples）下的 manual\_control.py 文件，用户可以通过 WASD 按键来控制中心车辆，具体效果如图 9 所示：



用户也可以尝试在中心车辆上添加传感器以观察效果。



### 3.路径规划算法

首先 Carla 里的 **behavior planning** 大致分为全局路线规划、行为规划、轨迹规划与底层控制四大部分。用户可以直接通过修改位于 **CARLA\_0.9.14\WindowsNoEditor\PythonAPI\examples** 中自带的 **automatic\_control.py** 文件进行路径规划。但是 **automatic\_control.py** 原本的功能为随机在地图上生成一辆小汽车，随机设置目的地，让汽车在限速内行驶至目的地。

CARLA 软件定义了 **Behavior Agent** 类，**BehaviorAgent** 在构建时需要两个输入，一个是属于 **Actor class** 的 **vehicle**，另外一个就是车辆驾驶风格（**string type**）。用户可以通过查看位于 **C:\Users\12489\Downloads\CARLA\_0.9.14\WindowsNoEditor\PythonAPI\carla\agents\navigation** 目录下的 **behavior\_agent.py** 文件，如图 10 所示：

```
class BehaviorAgent(BasicAgent):  
    """  
    BehaviorAgent implements an agent that navigates scenes to reach a given  
    target destination, by computing the shortest possible path to it.  
    This agent can correctly follow traffic signs, speed limitations,  
    traffic lights, while also taking into account nearby vehicles. Lane changing  
    decisions can be taken by analyzing the surrounding environment such as tailgating avoidance.  
    Adding to these are possible behaviors, the agent can also keep safety distance  
    from a car in front of it by tracking the instantaneous time to collision  
    and keeping it in a certain range. Finally, different sets of behaviors  
    are encoded in the agent, from cautious to a more aggressive ones.  
    """
```

图 10

根据图 10 可以查看该文件所能实现的功能，继续往下可以查看到部分代码：

---

```
# Parameters for agent behavior
```

```
if behavior == 'cautious':
```

```
self._behavior = Cautious()

elif behavior == 'normal':

    self._behavior = Normal()

elif behavior == 'aggressive':

    self._behavior = Aggressive()
```

---

如代码所展示，输入不同的驾驶风格字符串，**agent** 的成员行为会选择不同的 **class** 进行初始化。而 **Cautious()**, **Normal()**和 **Aggressive()** 这三个成员可以在同目录下的 **behavior\_types.py** 文件中查看到。

在这里以 **Normal()**和 **Aggressive()**举例,如图 11 所示:

```
class Normal(object):
    """Class for Normal agent."""
    max_speed = 50
    speed_lim_dist = 3
    speed_decrease = 10
    safety_time = 3
    min_proximity_threshold = 10
    braking_distance = 5
    tailgate_counter = 0

class Aggressive(object):
    """Class for Aggressive agent."""
    max_speed = 70
    speed_lim_dist = 1
    speed_decrease = 8
    safety_time = 3
    min_proximity_threshold = 8
    braking_distance = 4
    tailgate_counter = -1
```

图 11

**types\_behavior** 里主要是定义汽车的限速 (**max\_speed**)、与前车保持的安全时间 (**safety\_time**)，与前车的最小安全距离 (**braking\_distance**) 等安全相关的参数，**agressive** 的车相对 **normal**

的车来说速度更快，跟车更紧。用户可以修改部分参数达到不同的控制效果。

回到 `automatic_control.py` 文件下，查看如下代码：

---

```
spawn_points = self.map.get_spawn_points()

spawn_point = random.choice(spawn_points) if spawn_points
else carla.Transform()

self.player = self.world.try_spawn_actor(blueprint, spawn_point)

self.modify_vehicle_physics(self.player)
```

---

该代码从地图中获取所有可用的车辆生成点。这些生成点是地图中提前已经定义好的,用于在模拟场景中放置车辆。从前一步获取的 `spawn_points` 列表中随机选择一个生成点。如果 `spawn_points` 列表为空,则使用默认的 `carla.Transform()` 作为生成点。使用选择的 `spawn_point` 在模拟世界中生成一个新的车辆演员。`blueprint` 是之前定义好的车辆的蓝图,用于指定车辆的类型和属性。`self.world.try_spawn_actor()` 方法尝试在指定的生成点生成车辆演员,并将其赋值给 `self.player` 变量。

因此用户可以修改车辆生产点来更改车辆起点位置。

同时用户还能看到如下代码，如图 12 所示：

```
749         # Set the agent destination
750         spawn_points = world.map.get_spawn_points()
751         destination = random.choice(spawn_points).location
752         agent.set_destination(destination)
```

图 12

该代码从当前模拟地图中获取所有可用的车辆生成点并从这些生成点中随机选择一个位置作为目标点。最后将这个随机选择的目标位置设置为车辆智能体的目的地。

用户可以更改部分代码来更改车辆的目的地。

现在用户可以根据之前的学习来更改车辆的起始位置和目的位置，并通过调用之前安装依赖项 `networkx` 中自带的 A\*算法实现车辆路径规划。具体代码部分用户可以查看 `CARLA` 安装目录下 `WindowsNoEditor\PythonAPI\carla\agents\navigation` 中的 `global_route_planner.py` 文件，代码部分如图 13 所示：

```
287 def _path_search(self, origin, destination):
288     """
289     This function finds the shortest path connecting origin and destination
290     using A* search with distance heuristic.
291     origin      : carla.Location object of start position
292     destination : carla.Location object of end position
293     return      : path as list of node ids (as int) of the graph self._graph
294                   connecting origin and destination
295     """
296     start, end = self._localize(origin), self._localize(destination)
297
298     route = nx.astar_path(
299         self._graph, source=start[0], target=end[0],
300         heuristic=self._distance_heuristic, weight='length')
301     route.append(end[1])
302     return route
```

图 13

到此，即可在 `CARLA` 中实现车辆路径规划。

#### 4. 轨迹跟踪控制

由上文所示，在 `CARLA` 中道路已经预定义过生成点，因此 `CARLA` 的轨迹跟踪可以通过预设轨迹点（`x,y,speed`）来控制车辆跟

随道路。车辆需要以一定的所需速度到达这些航点，因此需要纵向和横向控制。在此根据（<https://github.com/ActuallySam/Carla-Trajectory-Tracking-Project?tab=readme-ov-file>）中提供的模型进行实现。

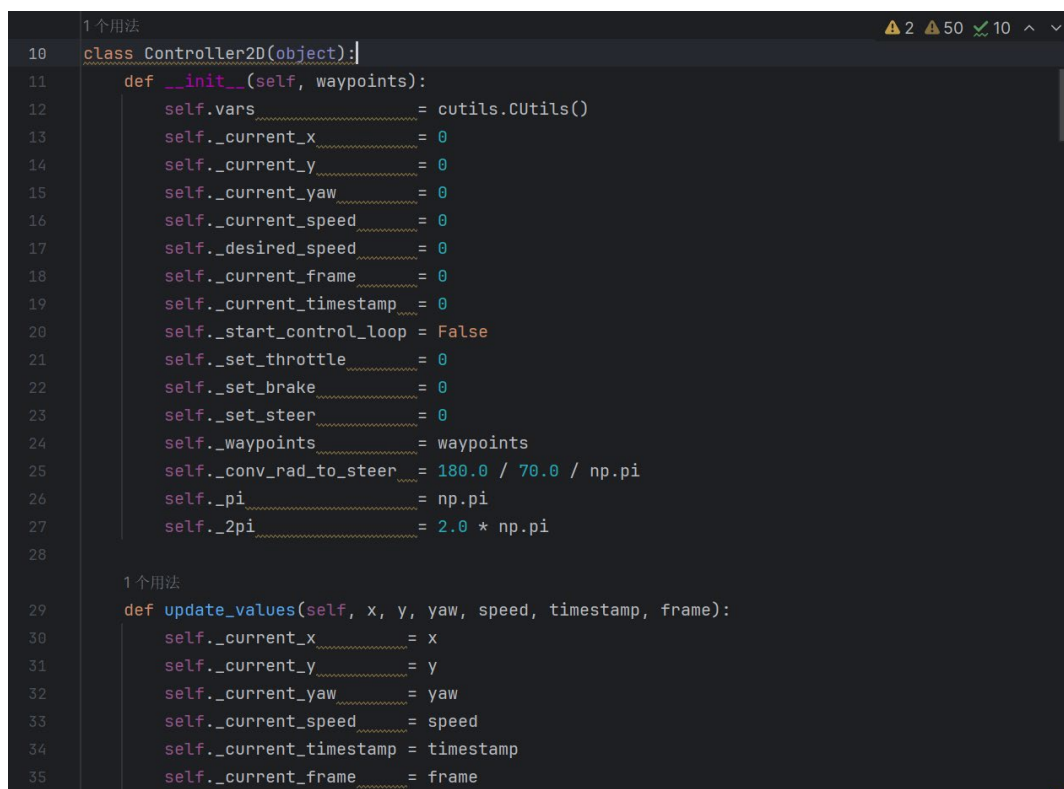
下载完该仿真后，在之前环境配置上的基础上，需在环境中配置：

---

```
pip install matplotlib==3.0.0
```

---

该算法设计了一个非线性模型预测器，用户可以打开目录下的 `controller2d.py` 文件进行查看，该代码首先通过类定义一些内部变量来存储车辆的当前状态和控制输出。根据车辆的当前状态和目标路径点，计算出适当的油门、刹车和转向控制输出，以使车辆沿着目标路径进行跟踪。它会通过内部的一些算法和计算，不断更新控制输出，以实现平滑和精准的路径跟踪，具体代码如图 14 所示下：



```
1 个用法
10 class Controller2D(object):
11     def __init__(self, waypoints):
12         self.vars = cutils.CUtils()
13         self._current_x = 0
14         self._current_y = 0
15         self._current_yaw = 0
16         self._current_speed = 0
17         self._desired_speed = 0
18         self._current_frame = 0
19         self._current_timestamp = 0
20         self._start_control_loop = False
21         self._set_throttle = 0
22         self._set_brake = 0
23         self._set_steering = 0
24         self._waypoints = waypoints
25         self._conv_rad_to_steering = 180.0 / 70.0 / np.pi
26         self._pi = np.pi
27         self._2pi = 2.0 * np.pi
28
29     1 个用法
29     def update_values(self, x, y, yaw, speed, timestamp, frame):
30         self._current_x = x
31         self._current_y = y
32         self._current_yaw = yaw
33         self._current_speed = speed
34         self._current_timestamp = timestamp
35         self._current_frame = frame
```

图 14

再此基础上分别设计横向控制器和纵向控制器，如图 15、16 所示：

```
159 # MODULE 7: IMPLEMENTATION OF LONGITUDINAL CONTROLLER HERE
160 #####
161 #####
162 """
163     Implement a longitudinal controller here. Remember that you can
164     access the persistent variables declared above here. For
165     example, can treat self.vars.v_previous like a "global variable".
166 """
167
168 # Change these outputs with the longitudinal controller. Note that
169 # brake_output is optional and is not required to pass the
170 # assignment, as the car will naturally slow down over time.
171
172 kp = 1.0
173 ki = 0.2
174 kd = 0.01
175
176 throttle_output = 0
177 brake_output = 0
178
179 # pid control
180 st = t - self.vars.t_previous
181
182 # error term
183 e_v = v_desired - v
184
```

图 15

```
202 #####
203 #####
204 # MODULE 7: IMPLEMENTATION OF LATERAL CONTROLLER HERE
205 #####
206 #####
207 """
208     Implement a lateral controller here. Remember that you can
209     access the persistent variables declared above here. For
210     example, can treat self.vars.v_previous like a "global variable".
211 """
212
213 # Change the steer output with the lateral controller.
214 steer_output = 0
215
216 # Use stanley controller for lateral control
217 k_e = 0.3
218 slope = (waypoints[-1][1]-waypoints[0][1]) / (waypoints[-1][0]-waypoints[0][0])
219 a = -slope
220 b = 1.0
221 c = (slope*waypoints[0][0]) - waypoints[0][1]
222
223 # heading error
224 yaw_path = np.arctan2(waypoints[-1][1]-waypoints[0][1], waypoints[-1][0]-waypoints[0][0])
225 # yaw_path = np.arctan2(slope, 1.0) # This was turning the vehicle only to the right
226 yaw_diff_heading = yaw_path - yaw
227 if yaw_diff_heading > np.pi:
228     yaw_diff_heading -= 2 * np.pi
229 if yaw_diff_heading < - np.pi:
```

图 16

用户可以查看同目录下的 `module_7.py` 文件，查看图 17 代码：



```

6 用法
def send_control_command(client, throttle, steer, brake,
                        hand_brake=False, reverse=False):
    """Send control command to CARLA client.

    Send control command to CARLA client.

    Args:
        client: The CARLA client object
        throttle: Throttle command for the sim car [0, 1]
        steer: Steer command for the sim car [-1, 1]
        brake: Brake command for the sim car [0, 1]
        hand_brake: Whether the hand brake is engaged
        reverse: Whether the sim car is in the reverse gear
    """
    control = VehicleControl()
    # Clamp all values within their limits
    steer = np.fmax(np.fmin(steer, 1.0), -1.0)
    throttle = np.fmax(np.fmin(throttle, 1.0), 0)
    brake = np.fmax(np.fmin(brake, 1.0), 0)

    control.steer = steer
    control.throttle = throttle
    control.brake = brake
    control.hand_brake = hand_brake
    control.reverse = reverse
    client.send_control(control)

```

图 17

函数创建了一个 **VehicleControl** 对象,用于存储车辆控制命令。

函数使用 **np.fmax()** 和 **np.fmin()** 函数将输入的 **steer**、**throttle** 和 **brake** 值限制在它们各自的合法范围内。这可以确保不会发送超出允许范围的控制命令。函数将经过限制的 **steer**、**throttle** 和 **brake** 值赋给 **VehicleControl** 对象的相应属性。函数设置 **control.hand\_brake** 和 **control.reverse** 属性,以指示是否启用手刹和倒车模式。最后,函数调用 **client.send\_control(control)** 将构建好的 **VehicleControl** 对象发送到 **CARLA** 客户端,以控制模拟车辆。

同时通过图 18 的部分代码将车辆轨迹数据写入文件:



```

224  def write_trajectory_file(x_list, y_list, v_list, t_list):
225      create_controller_output_dir(CONTROLLER_OUTPUT_FOLDER)
226      file_name = os.path.join(CONTROLLER_OUTPUT_FOLDER, 'trajectory.txt')
227
228      with open(file_name, 'w') as trajectory_file:
229          for i in range(len(x_list)):
230              trajectory_file.write('%3.3f, %3.3f, %2.3f, %6.3f\n' % \
231                                  (x_list[i], y_list[i], v_list[i], t_list[i]))

```

图 18

用户可以查看全部控制代码以实现轨迹跟踪，仿真结束后可以查看每个预设轨道点的速度要求如图 19 所示：

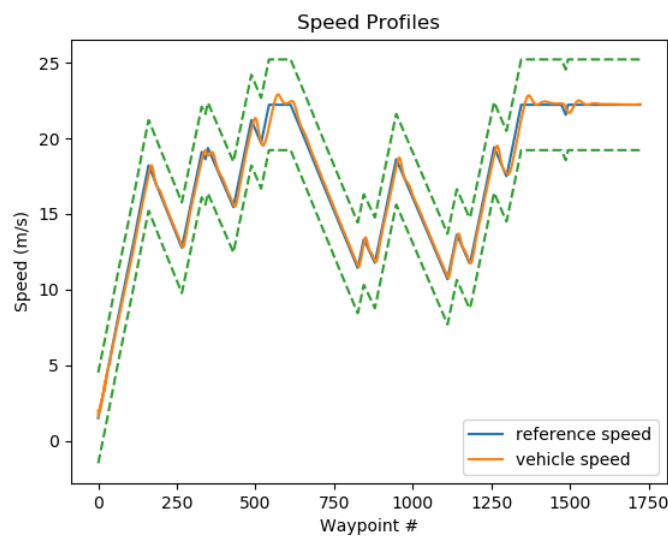


图 19

同时也可以查看导轨道形状如图 20 所示：

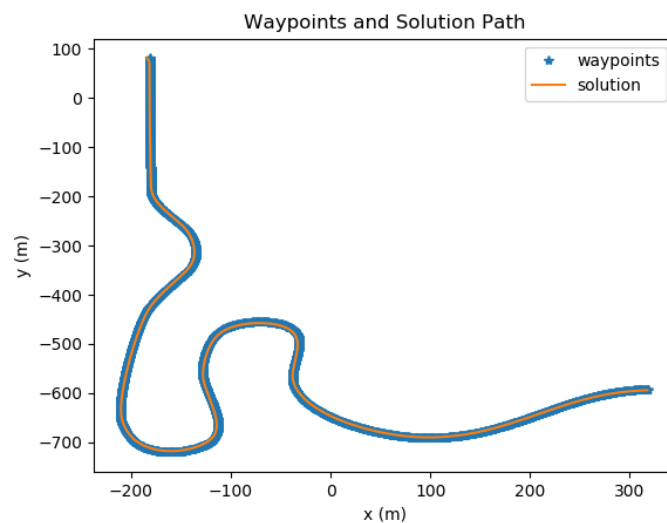


图 20