毕业设计（论文）中文摘要

基于 Unreal 的无人车交通场景搭建与行人检测

摘要：

近年来，无人驾驶技术一直是汽车领域的研究热点。在无人车的研发过程中，需要大量的试验来测试无人车算法的稳定性和准确性，由于实车试验价格昂贵，同时还需考虑车辆多种运行情况，所以需要一种虚拟测试来代替实车试验，例如 CARLA、LGSVL、AirSim 和 AADS 等模拟器。本文通过使用基于 Unreal 的模拟器 CARLA 来完成一般交通场景的搭建，并在其中验证无人车相关的控制算法及行人检测算法。

本文首先完成了道路、树木及楼房的建模。其次搭建了汽车及行人模型，规划了行人运动轨迹和交通信号灯交替时间。然后对无人车的感知、决策和控制三个方面进行了程序编写，完成了紧急避让、轨迹跟踪以及汽车和行人识别等功能。最后实现了无人车在虚拟环境中的安全运行，并验证了无人车控制算法的正确性。

关键词： CARLA　行人检测　无人车仿真　轨迹跟踪　避让控制

毕业设计（论文）外文摘要

Title Unreal-based driverless vehicle traffic scene construction and pedestrian detection

Abstract

In recent years, unmanned driving technology has always been a research hotspot in the automotive field. In the development process of unmanned vehicles, a large number of tests are required to test the stability and accuracy of the algorithm of unmanned vehicles. Because the actual vehicle test is expensive, and the various operating conditions of the vehicle need to be considered, a virtual test is required. Instead of real vehicle tests, such as CARLA, LGSVL, AirSim and AADS and other simulators. This article uses the Unreal-based simulator CARLA to complete the construction of general traffic scenes, and verifies the control algorithms and pedestrian detection algorithms related to unmanned vehicles.

This article first completed the modeling of roads, trees and buildings. Secondly, models of cars and pedestrians were built, and pedestrians' trajectories and traffic lights alternate time were planned. Then, the three aspects of perception, decision-making and control of the unmanned vehicle were programmed, and the functions of emergency avoidance, trajectory tracking, and vehicle and pedestrian recognition were completed. Finally, the safe operation of the unmanned vehicle in the virtual environment is realized, and the correctness of the unmanned vehicle control algorithm is verified.

Keywords： CARLA pedestrian detection unmanned vehicle simulation trajectory tracking avoidance control

目　　录

1 绪论

1.1 研究背景

近几年，无人车仍是重点研究对象，具有很大的市场前景和发展潜力。无人车是一个多技术集合体，需要各个领域的知识相结合才能实现。无人车的安全舒适是人们所需求的，所以在无人车投入市场使用前，要进行大量的模拟实验来确保无人车的安全性。

由于制造配备所需传感器的真实车辆非常重要且价格昂贵，同时还需要考虑车辆可能运行的所有可能情况，测试每个组件（包括硬件和软件），实车试验无法满足，所以需要一种虚拟测试来代替实车试验，并待技术成熟后再进行实车模拟。

机器人模拟器是一种有效的工具，可以让开发者在受控环境中快速而廉价地设计和测试机器人，而无需物理硬件。诸如 Gazebo [1]、V-REP [2]和 Webots [3]之类的流行模拟器已被用来模拟各种系统，包括工业机器人，无人驾驶飞机和自动驾驶汽车。对于机器人系统的验证和确认而言，仿真有很大的发展前景，它能提供一种自动化、经济高效且可扩展的替代方案，以替代手动且昂贵的现场测试过程[4]-[7]。并且仿真可以有效地发现各种机器人应用程序域中的错误[8]-[11]。例如就像 Uber [12]、NVIDIA [13]和 Waymo [14]一样，大规模地使用仿真来开发、训练和测试他们的算法。该领域对仿真的高需求催生了新一代专门模拟器的开发，例如 CARLA[15]、LGSVL [16]、AirSim [17]和 AADS [18]。

尽管模拟器具备这些优点，但同时存在一个称为现实差距的问题，该问题描述了模拟世界与现实世界之间的差异。这包括诸如缺乏传感器数据的丰富性和噪声性，或者缺乏各种环境，这可能导致概括世界的能力不足，无法对以前看不见的数据做出反应，所以模拟器的进一步发展对自动驾驶将会有重大影响，由此可见模拟器在自动驾驶研发中的地位。

同时感知技术日趋成熟，若是能够让无人车更精确、快速的感知周围的车辆、行人和道路信息，那就能保证无人车乘坐者的安全，减少和避免危险的发生，加快无人车进入市场的步伐，逐步取代现有的汽车。因此，行人和汽车识别作为无人车感知中

的重点，就变得尤其重要，它是无人车的眼睛，为无人车提供了视觉，能够识别并区分周围的人和物，使无人车更好的服务于人类[19]。行人检测[20]是无人车研究的一个难点，十几年来研究人员提出了众多算法模型，检测的速度和准确率不断提高，相信随着计算机技术的进一步发展，检测技术将更加成熟。

1.2 研究现状

1.2.1 模拟器研究现状

目前主流的汽车模拟器主要有三种：CARLA、LGSVL、AirSim。在介绍三种模拟器之前，首先介绍一下汽车模拟器的运行引擎，主要为虚幻引擎（Unreal Engine , UE）和 Unity 两种，本文重点介绍 UE。

UE 是 EpicGames 公司打造的游戏引擎，1998 年该公司发行了第一代 UE 产品。2002 年，Epic 公司发布了第二代引擎 UE2，UE2 相对于第一代来说，增添了实时性，即随时更改物体属性。2006 年，Epic 公司发布了第三代引擎，即 UE3，受到了广大使用者的欢迎。UE3 带来的创新可谓是巨大的，当时大多数的游戏引擎编程都需要代码编写，而 UE3 开发了可视化脚本 Kismet，即图形语言，省去了代码的堆叠，对于没有学过编程语言但想开发的人来说，无疑是一种福音，因此 UE 的知名度得到大大的提高。2014 年，Epic 公司发布了第四代引擎，即 UE4。就在不久前，UE5 已经发布，UE5 相对于 UE4 来说增加了光线追踪功能，使得物体更加的逼真，真实的模拟现实环境中的光线折射和反射效果，使得物体更加真实，但是还没有被广泛使用。所以到目前为止，UE4 仍然是是当前知名度最广、性能效果最好的引擎。从 1998 年到现在，经过 20 多年的不断改进和完善，虚幻引擎不光在游戏界名声大震，在建筑、影视、汽车等行业中，也受到众多人的喜爱，虚幻引擎相较于其他引擎来说，受众范围广，运用程度高，是次世代画面的代表游戏引擎。

UE 之所以有如此之大的影响力，必然有着其他引擎不可比拟的优势，针对当前版本的 UE4 引擎来说，UE4 引擎的优点有：

（1）UE4 具有优秀的显示效果，能够将一个模型更加真实的显示在模拟器当中。

（2）UE4 有着卓越的光照模拟和物理仿真，对于引擎来说，评价其性能好坏的重要一点，就是模拟现实世界的能力，模拟的越真实，说明其性能越好。

（3）UE4 具有可视化编程功能，减少了代码编程的时间，逻辑清晰可见。并且插件齐全，供用户自由选择。还有就是对 VR 和手柄等外设支持良好，且提供各种模板，

既有免费使用的，也有收费的。

（4）UE4 渲染效果强大，其采用 pbr 物理材质系统，能够达到类似 Vray 静帧的效果。

（5）UE4 是开源的，并且免费，可以为众多学者提供开发环境，促进引擎的进一步完善，也能促进人们的开发热情。引擎的源码可以从 Github 上下载，也就是说，任何人都可以自由控制引擎，包括修改界面和更改任何东西的属性，这一方面能够让开发者在程序出错后，快速、准确的判断出错的位置和原因，节省翻来覆去找错误的时间。另一方面，开源可以深入学习引擎的原理，更好的发挥引擎的强大功能。

所以由此可以看出，经过 20 多年的发展，UE4 不仅可以应用在游戏行业，还能用于其他方面，为其他行业的人提供开发环境，创造出更多的可能。对于游戏行业工作者来说，UE4 可以用来制作游戏和动画；对于影视行业工作者来说，UE4 可以进行电影场景设计和人物动作设计；对于建筑行业的人来说，UE4 可以进行建筑物设计；对于汽车行业的人来说，UE4 可以进行汽车模型搭建和运行仿真。总而言之，凡是涉及到三维建模、运动仿真和虚拟环境模拟的地方，UE4 都可以使用，可见其应用范围之广，UE4 界面如图 1.1 所示：

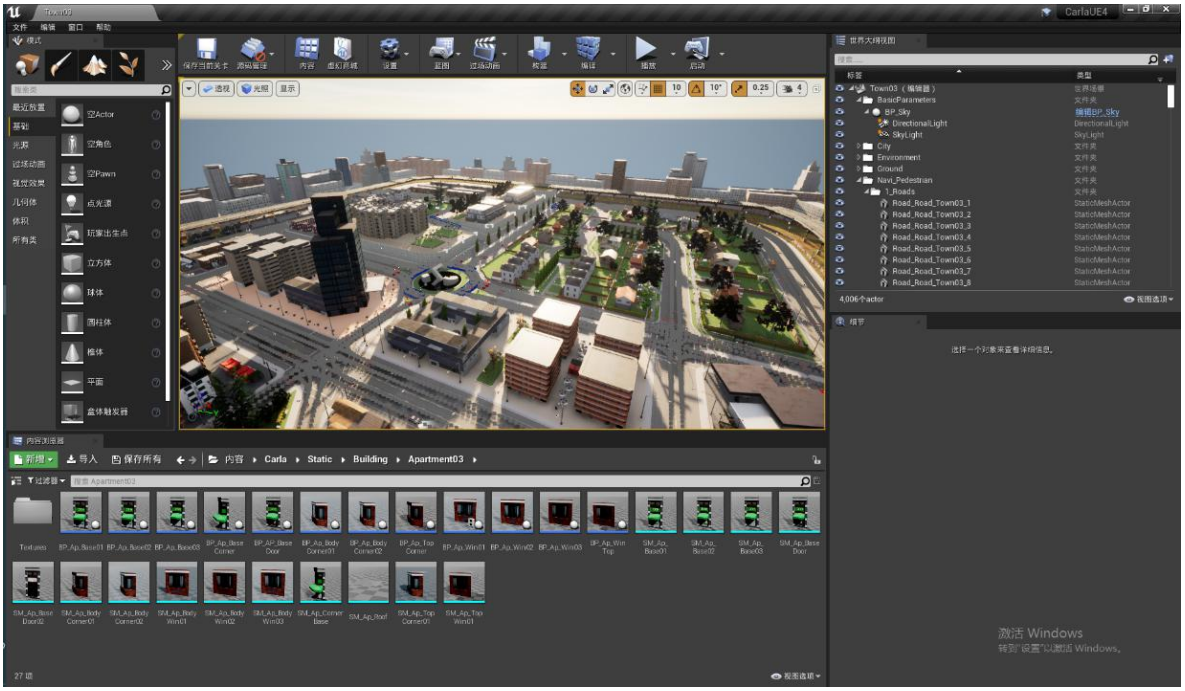


图 1.1 UE4 使用界面

Fig. 1.1 UE4 User Interface

接下来重点介绍基于 UE4 和 Unity 引擎的无人驾驶汽车模拟器：

（1）CARLA

CARLA [21]是由巴塞罗那大学计算机视觉中心开发的一款模拟器，如图 1.2 所示。该模拟器与 UE4 一起运行，它是免费使用且完全开源的。尽管它提供了各种不同的即用型地图，包括环境资产和各种道路使用者，例如不同的汽车，骑自行车的人和行人。但是它没有提供方便的方法来实现域随机化。创建和修改地图的最简单方法是使用 VectorZero 的工具 RoadRunner [22]。同样，可以导入带有 OpenDRIVE 文件的道路网络。即使它不支持 Open AI Gym，但也有几个包装程序将模拟器与库连接起来，同时 CARLA 缺乏轻松修改模拟的能力，开发的文件记录不充分，这使得实施定制系统以自动随机化任何东西变得非常困难。同样，驾驶物理学对汽车模拟来说是基本的，但 CARLA 提供很少的参数来调整系统以适应现实世界。

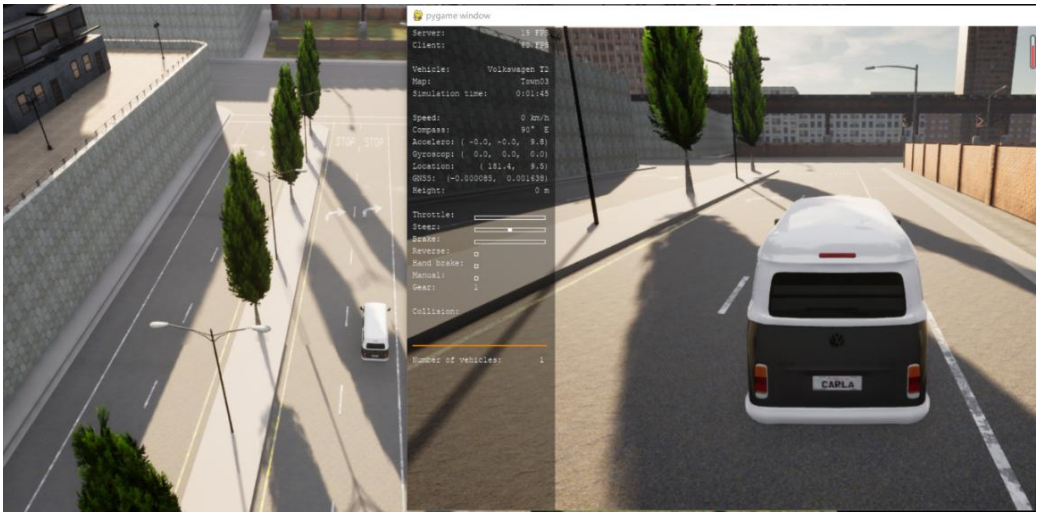


图 1.2 CARLA 模拟器

Fig. 1.2 CARLA Simulator

（2）LGSVL

第二个是图 1.3 中所示的 LG Electronics [23]的 LGSVL 模拟器（LG 硅谷实验室）。该模拟使用 Unity 引擎运行。它是免费的，并且开放源代码，可以根据《模拟器软件许可协议》使用。与 CARLA 一样，它支持使用 RoadRunner 创建地图，但是已经有几张地图可供使用。它还提供了各种汽车和行人，同样可以集成特定的车辆。 此外，它还提供本机 Open AI Gym 支持。不幸的是，它也缺乏与 CARLA 一样的灵活性。没有现成的系统可以随机化任何东西。同样，在从仿真系统外部连续加载新内容的同时，也没有方便的方法来运行仿真。

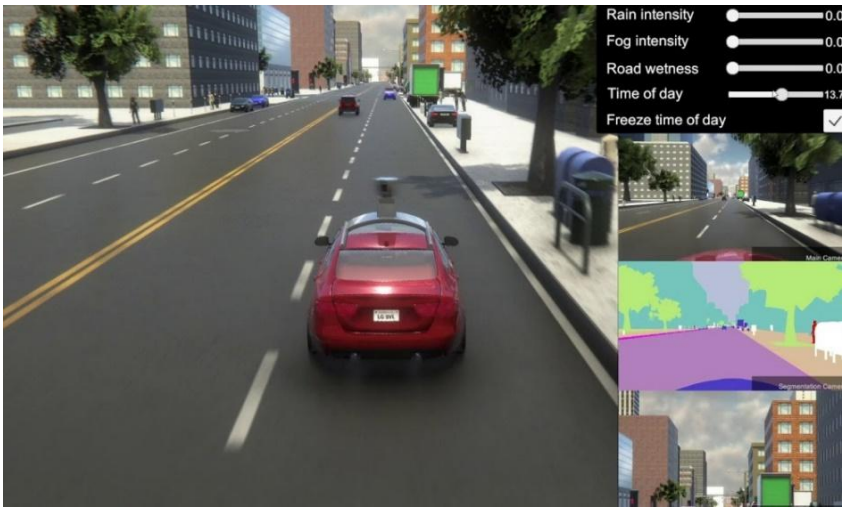


图 1.3 LGSVL 模拟器

Fig. 1.3 LGSVL Simulator

（3）AirSim

第三个是 Microsoft 的 AirSim[24]，如图 1.4 所示。AirSim 在 UE4 以及 Unity Engine 上运行。只要两个引擎在功能上保持同步，就能够在两个引擎上同时开发。像前两个系统一样，它是开源的，可以根据 MIT 许可证免费使用。它最初是作为无人机模拟器开发的，但自 2017 年以来，它还支持地面车辆。与 CARLA 和 LGSVL 不同，它已经提供了某种纹理交换形式的域随机化。J. Tobin, R Fong 等人[25]表明，即使使用简单的纯色，也可能获得令人满意的结果。由于其起源于无人机开发，因此几乎可以将任何类型的车辆集成到仿真中。并且它提供了多种地图，包括一个名为 Windridge City 的大区域，该区域比大多数其他模拟提供更大的区域。它还具有各种现成的车辆。

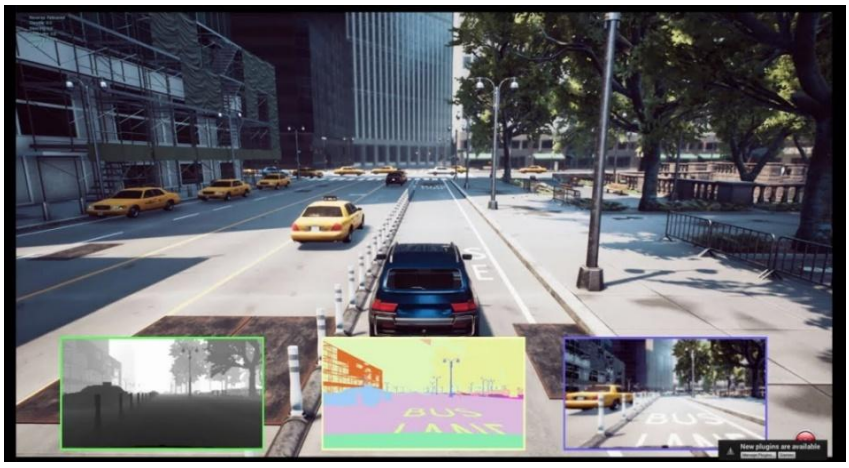


图 1.4 AirSim 模拟器

Fig. 1.4 AirSim Simulator

1.2.2 行人检测研究现状

行人检测技术开始于上世纪九十年代，直到现在，仍然被广泛研究和探讨。行人检测技术对于汽车的无人驾驶来说至关重要，不但可以应用于无人车系统，还可以应用于智能机器人等方面。虽然被研究了很长时间，但目前仍然存在许多难点，需要去解决，主要有以下几个方面：

第一是复杂多变的环境因素，环境因素直接影响了图像的质量，只有越清晰、分辨率越高的图像才更容易识别，如何去除这些影响是人们需要解决的问题。

第二是行人的步态和形态识别，步态识别就是判断行人是慢行还是跑步，通过对人的动作来进行识别，形态识别就是对人的外表识别，如身高、胖瘦等。如何获取更多、更细致的行人的信息需要进一步研究。

最后是遮挡问题，现实世界是复杂的，存在人被遮挡的情况，如果不能及时正确的识别，将会带来安全隐患，这也是一个难点，无人车要想进入实际道路使用，就必须解决这一问题。

行人检测按照原理可以分为三大类：

（1）基于背景建模的算法

常用的背景建模算法有：高斯混合模型算法，ViBe 算法[26]，帧差分算法。所谓背景建模，就是指摄像机静止不动，这样所拍摄的画面中，除了行人、汽车等物体以外，其他事物都是静止的，进而能够只提取出运动的物体，直接节省了检测其他部分的时间，然后对目标进行判断、识别。背景建模算法的优点有：原理简单，运行速度快。但正是由于这些优点也导致了一些问题，例如只能检测运动的物体，无法识别静止的事物；其次受光照、阴影的影响很大；还有就是对于图中的密集物体，如树木、树叶等，无法正常检测。

（2）基于机器学习的算法

第二种算法是目前最常用的检测方法，提取的目标特征主要有颜色、边缘、纹理等信息，采用的分类器有神经网络，AdaBoost，支持向量机(Support Vector Machines，SVM)，以及深度学习。一般采用滑动窗口的技术。基于机器学习的算法基本都是基于法国研究人员 Dalal 在 2005 年发表的方向梯度直方图（Histogram of Oriented Gradient, HOG) + SVM 的行人检测算法[27]。几年以前，Felenszwalb 等人[28]提出了形变部件模型（Deformable Part Mode，DPM)算法，该算法能够解决行人姿态不同的识别问题，并且能很好的区分行人和其他事物。

（3）基于深度学习的算法

基于背景建模和机器学习的方法具有局限性，即行人检测速度和准确率不稳定，受限于环境条件，所以需要新的算法进行替代。十年以前，人们开始将深度学习技术应用到图像分类中[29]，通过几年的不断探索和实验，人们发现基于深度学习的算法在物体识别方面具有许多优势，能够弥补前两种方法的弊端。

经过不断的发展，目前有许多基于深度学习的算法，例如 Faster-RCNN[30]、SSD[31]、FPN[32]、YOLO[33]等，相比之前的算法，精度有了显著的提升，不但具有很高的准确率，并且检测速度也得到了大大的提升。

1.3 研究内容

对于模拟来说，所搭建的环境越真实，物理性能越好，那仿真结果的准确性就越高，对于汽车实验来说具有重要帮助，能够大大减少实验成本，有助于汽车的开发。行人检测对于自动驾驶来说是关键，检测的准确性和快速性将直接影响汽车的安全性能，采用深度学习算法将提高检测的速度。

本文的主要目的是搭建一个虚拟环境，并在其中验证无人驾驶汽车相关算法的正确性，主要采用的软件为 CARLA，通过 Python 编程实现算法功能，并在 CARLA 中显示实际运行结果，本文各章的主要研究内容如下：

第一章主要是相关模拟器的介绍和使用方法以及行人检测算法简介。

第二章主要研究内容是搭建模拟环境，即一般的交通场景，包括地图的搭建、行人和汽车模型的搭建和天气设置，主要在 UE4 中进行模型设计，然后在 CARLA 中调用所设计的模型和地图，完成交通场景的搭建，为之后无人车算法测试搭建场所。

第三、四、五章开始构建无人驾驶汽车，主要针对无人车三个技术方面进行研究。第三章研究无人车的感知识别，用来实现汽车和行人检测，首先要完成传感器添加和配置，实现数据采集功能，然后对比现有模型，实现汽车和行人的识别，并在图中标出。第四章研究无人车的决策规划，用来实现避让车辆和行人，通过搭建简易危险场景来测试算法的准确性，算法的原理为坐标变换，通过变换测得视线中汽车和行人距自己的距离，然后根据距离进行避让。第五章研究无人车的控制执行，即无人车按照规定路线行驶，不发生跑偏，且转向平顺，满足一般舒适性要求，主要研究汽车的纵向和横向控制，纵向控制采用 PID 控制，横向控制方面通过对比斯坦利控制（Stanley Control）[34]、纯追踪控制（Pure Pursuit Control）[35]和模型预测控制（Model Predictive Control，MPC）[36]三种算法的优缺点及难易程度，最后选择斯坦利控制。

第六章主要是结果分析，通过将三、四、五章的程序进行整合，实现汽车在虚拟环境中的自动驾驶，最后分析结果，评价各种算法的优缺点并进行总结，论文框架如图 1.5 所示：

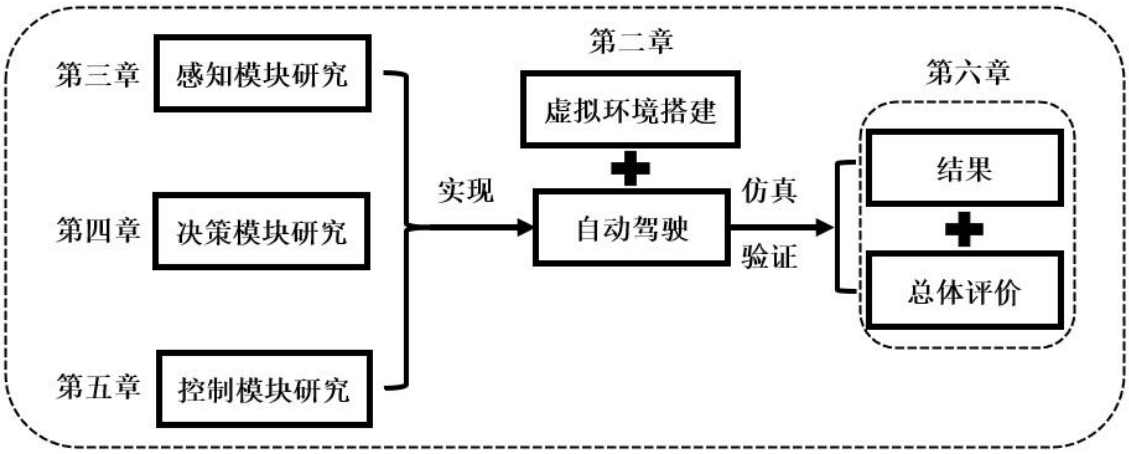


图 1.5 论文框架

Fig. 1.5 Paper Framework

2 无人车交通场景搭建

2.1 CARLA 模拟器

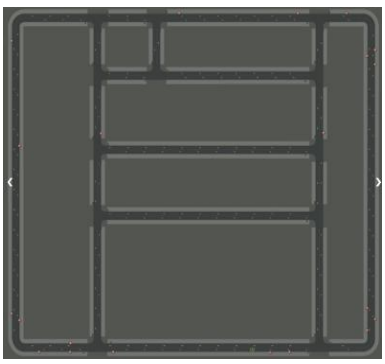
本文使用开源的城市驾驶模拟器 CARLA。CARLA 从一开始就是为了供使用者试验自动驾驶模型和算法验证而开发的。CARLA 是开源的，并且 CARLA 提供了免费的常见交通场景事物蓝图，包括汽车模型、行人模型、建筑房屋、交通信号灯和路灯等。CARLA 仿真平台提供了多种传感器供开发者使用，开发者可以自定义传感器参数，同时 CARLA 可以自由更改试验环境，包括天气状况和时间。CARLA 基于 UE4，用 UE4 来将 CARLA 可视化。CARLA 使用 Python 来进行程序编写，并且官方提供了许多现有程序供使用，能够实现对交通场景的更改和传感器配置，本文的程序部分参考了官方的例子。

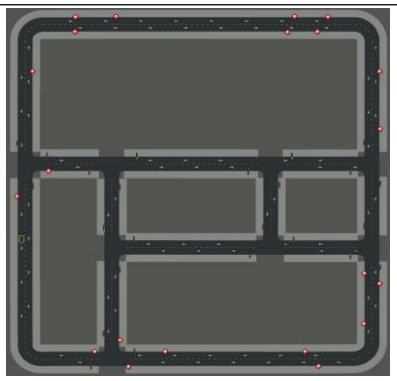
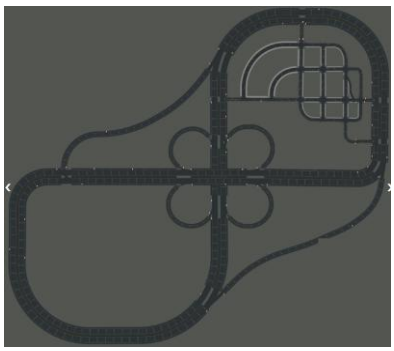
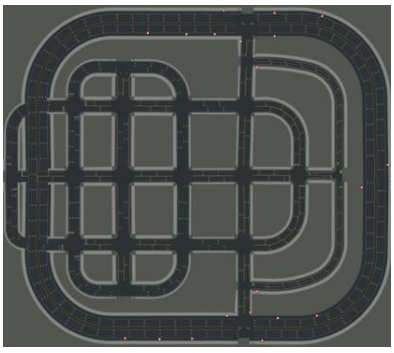
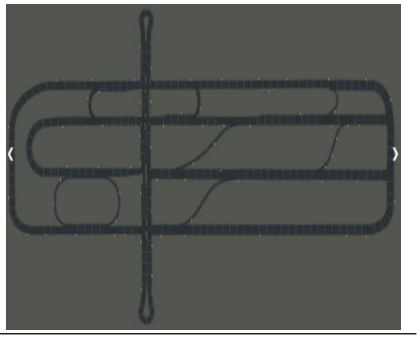
CARLA 主要分为服务端（Server）与客户端（Client）两个模块，Server 用来建立这个仿真世界，而 Client 则是由用户控制，用来调整、变化这个仿真世界。UE4 主要负责对仿真世界进行修改，更改模型及相关配置参数。整个实验仿真过程为：先在 UE4 中搭建所需的模型，并对模型的参数进行配置，然后在 CARLA 中调用并显示已搭建的模型，最后通过 Python 编写相关算法程序，实现无人车在所搭建环境中的安全运行。

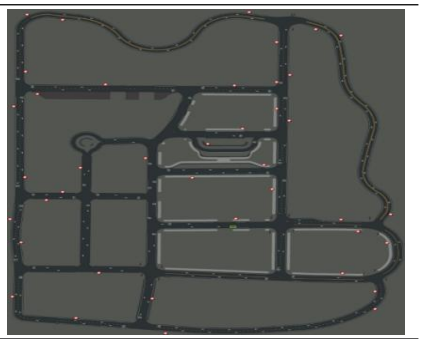
2.2 世界道路模型搭建

表 2.1 地图简介

Table 2.1 Introduction to Map

城镇	描述	道路图
Town01	具有“T 型交叉路口”的基本城镇布局。	

Town02	与 Town01 类似，但较小。	
Town03	最复杂的城镇，有 5 车道交界处，回旋处，不平坦处，隧道。	
Town04	一个无限的环路，有一条高速公路和一个小镇。	
Town05	方格网镇，带有交叉路口和一座桥。	
Town06	高速公路地图，有许多高速公路出入口。	

Town07	乡村环境，道路狭窄，几乎没有交通信号灯和谷仓。	

在 CARLA 的官方库中总共有七张地图，分别对应了不同的实验环境和路况，可以模拟多种情景。如表 2.1 所示，CARLA 现有的地图有 7 种。

本文采用 Town02 作为实验地图，首先第一步选取实验道路，本文选取 Town02 的道路，并在 UE4 中导入了相关道路的模型，搭建了一个基础，然后在此基础上搭建其他模型。

第二步道路选定好后，可以搭建一些房屋建筑，CARLA 所给的建筑蓝图种类多样，足够本文使用，如图 2.1 所示，本文基于所给定的道路，进行沿途建筑搭建，可以根据一般道路场景进行搭建，本文此次参考乡镇的一般交通场景，所以以低层建筑物为主。

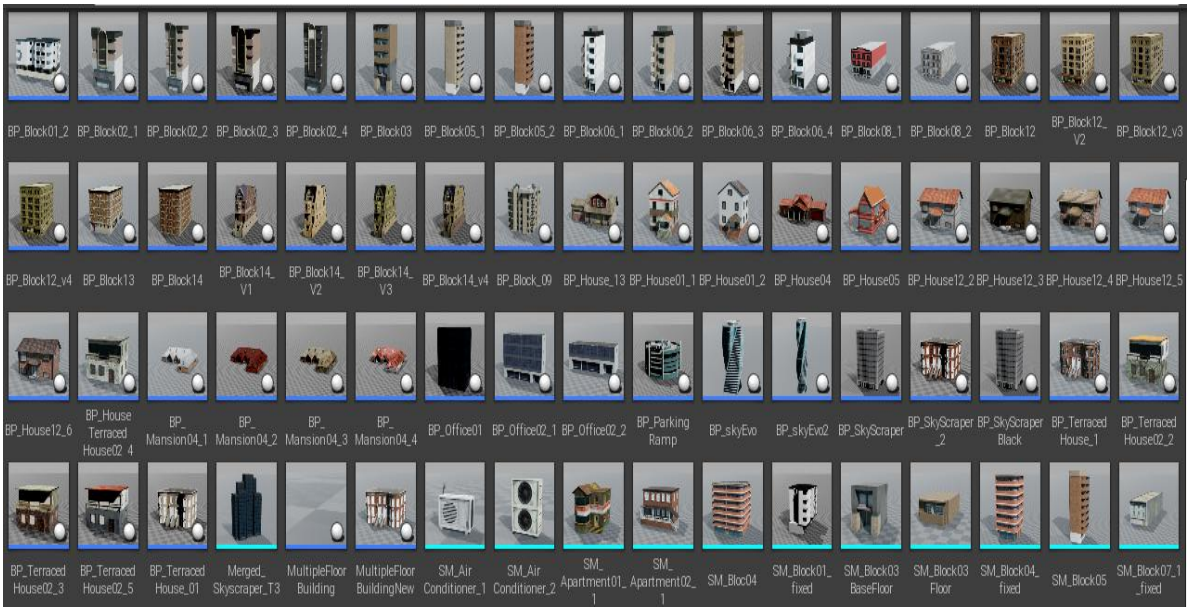


图 2.1 建筑蓝图

Fig. 2.1 Architectural Blueprint

第三步进行交通信号灯和路灯的搭建，交通信号灯的位置和变换时间比较重要，决定了交通场景的安全性和拥挤情况，保证所有车安全行驶。本文对交通信号灯的位置的安放比较重视，首先在路口处优先考虑安放交通信号灯，但是对于短距离多路口

的情况，可以视情况而定，不一定要安放，还有就是环岛，可以选择视情况而定。UE4 中的交通信号灯如图 2.2 所示：

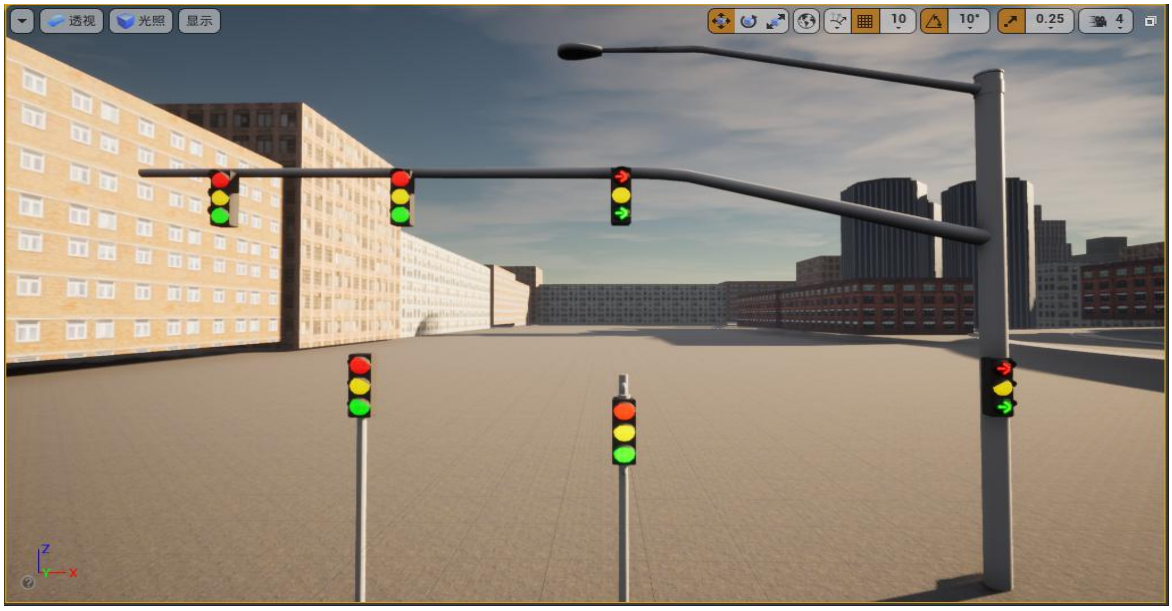


图 2.2 交通信号灯

Fig. 2.2 Traffic Lights

最后一步创建 Client，并且设置一个超时时间防止连接时间过久，本文根据不断测试，发现时间设置为 5 秒效果最好，所以本次超时时间设置为 5 秒。然后连接到世界，并获取世界的信息。最后 Town02 世界如图 2.3 所示：

图 2.3 Town02 世界

Fig. 2.3 Town02 World

2.3 汽车模型设计及搭建

CARLA 的库中有许多汽车和行人模型，本文则选用特斯拉的 model3 作为参考车型。首先在 UE4 的界面中，可以修改车辆的基本参数，包括引擎设置、差速器设置、变速箱设置、整车质量、阻力系数、底盘高度等等。

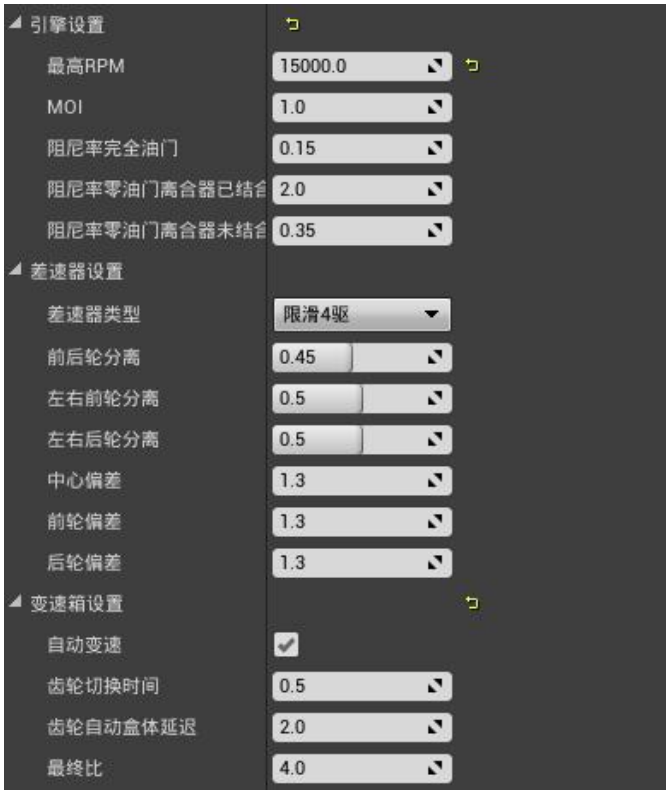


图 2.4 汽车参数配置

Fig. 2.4 Car Parameter Configuration

如图 2.4 所示，本次实验车的最高 RPM 设置为 15000.0，差速器类型为限滑 4 驱，变速箱采用自动变速，其他参数选择默认设置。

整车质量设置为 1850kg，车轮阻力系数设置为 0.3，底盘宽度和高度分别设置为 180.0cm 和 140.0cm，如图 2.5 所示：

图 2.5 质量和车轮设置

Fig. 2.5 Quality and Wheel Settings

其他汽车参数采用默认设置，最后 UE4 中的汽车模型如图 2.6 所示：

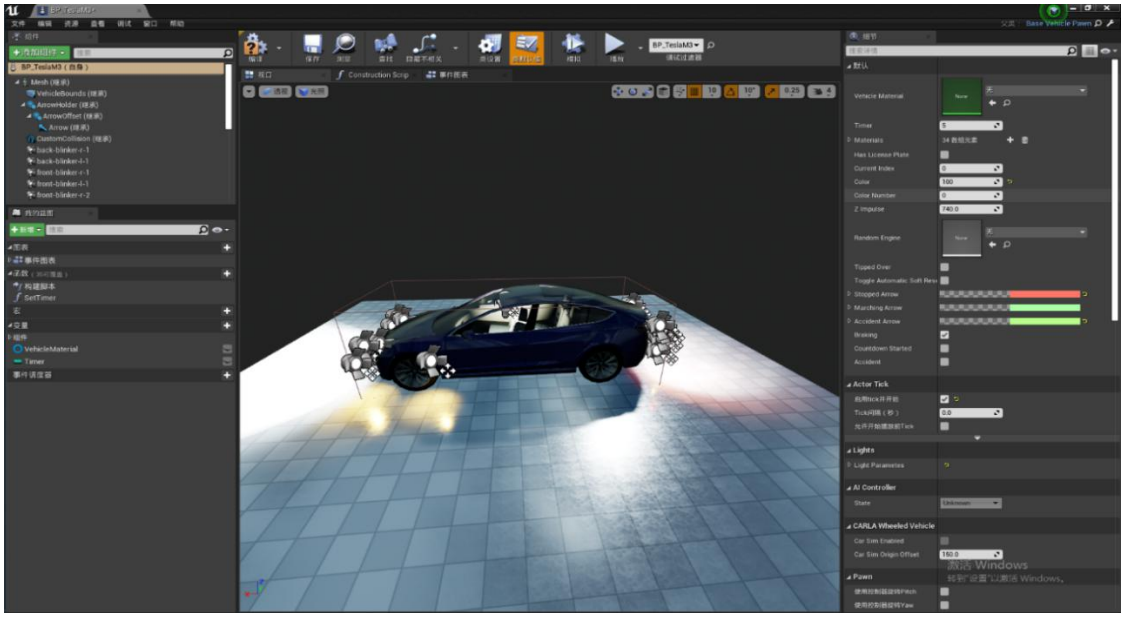


图 2.6 汽车模型

Fig. 2.6 Car Model

在 UE4 中配置好汽车的基本参数后，还需要在 CARLA 中利用如下命令生成上述 actor，首先定义它的蓝图，包括颜色、种类等，如下语句：

```
blueprint_library = world.get_blueprint_library()
vehicle_bp = blueprint_library.filter("model3")[0]
```

构建好蓝图以后，下一步选定生成点。可配置具体的坐标，或采用 random 函数随机生成，随机生成可能带来的后果就是两辆汽车在同一位置生成，导致发生错误，所以在一个位置上只能有一个模型生成，否则会产生错误。

随机位置生成采用 random 函数：

```
spawn_point = random.choice(world.get_map().get_spawn_points())
```

或给定具体的 x、y、z 坐标和转向：

```
location = carla.Location(230, 195, 40)
rotation = carla.ratation(0, 0, 0)
```

汽车生成以后，可以定义它的动态参数，包括油门、转向、刹车、手刹和档位，还可以通过 CARLA 提供的自动驾驶函数设置为自动驾驶：

```
vehicle.set_autopilot(True)
```

CARLA 中的汽车具有许多物理特性，包括弹力、惯性、摩擦力等参数，可以在中途将车“冻住”，通过抹杀它的物理仿真：

```
vehicle.set_simulate_physics(False)
```

还可以具体设置它的动态参数：

```
vehicle.apply_control(carla.VehicleControl(throttle=1.0,
steer=0.0, brake=0.0, hand_brake=(False), gear=1))
```

最终在 CARLA 中生成的汽车模型如图 2.7 所示：

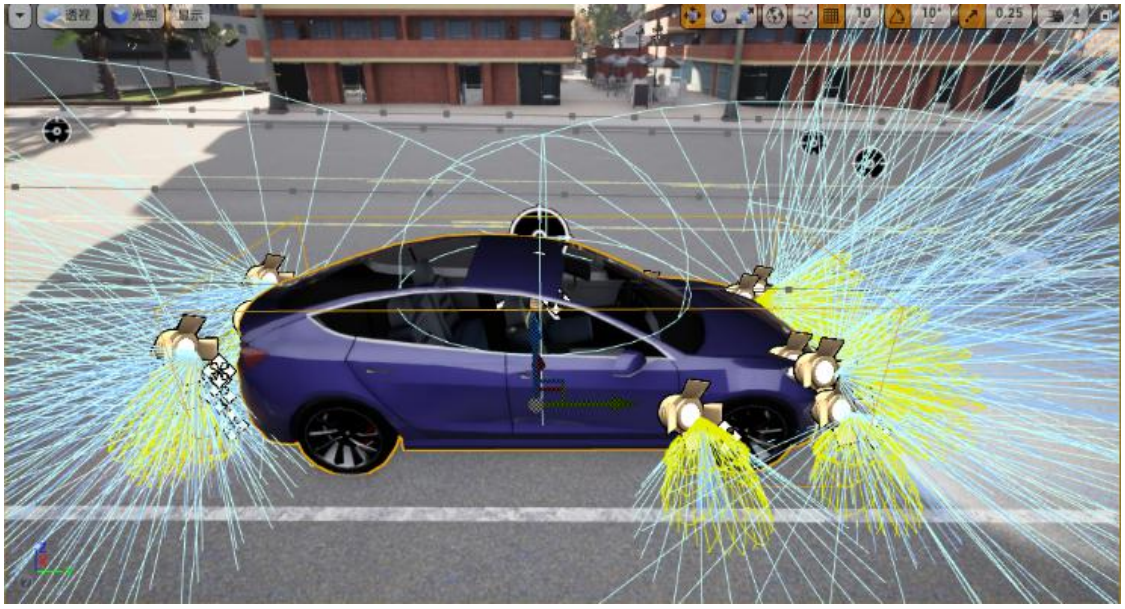


图 2.7 CARLA 中的汽车模型

Fig. 2.7 Car Model in CARLA

2.4 行人模型设计及搭建

行人模型搭建与汽车类似，首先在 UE4 中设置基础参数，如表 2.2 所示：

表 2.2 行人模型参数

Table 2.2 Pedestrian Model Parameters

参数名称	数值	单位
最大步高	45	cm
可行走地面角度	45	°
地面摩擦力	8.0	N
最大行走速度	30	km/h

最大蹲伏行走速度	10	km/h

同时还可以选择行人类型，如男性、女性，还可以选择成人或孩童，UE4 中提供的模型如图 2.8 所示，由于本文涉及到了行人检测，所以为了确保仿真结果的准确性和可靠性，采用随机模型选取。

图 2.8 行人模型

Fig. 2.8 Pedestrian Model

设置好基本参数后，在 CARLA 中定义它的蓝图：

```
blueprint = world.get_blueprint_library().filter("walker.*")
```

然后选择种类，本次实验行人种类采用随机选择，有大约 28 种行人模型供随计选择，从小孩到成年人：

```
random.choice(blueprint)
```

出生点位置设置与汽车类似，也采用随机生成，从地图的任意 spawn point 点生成：

```
spawn_point = random.choice(world.get_map().get_spawn_points())
```

与汽车不同的是，行人生成后可以设置其运动状态，例如步行和跑步两种状态，通过设置行人的速度来实现：

```
player_control = carla.WalkerControl()
player_control.speed = 3
```

CARLA 中行人速度有个分界线，当设置的行人运动速度大于 1.5m/s 时，CARLA 设定行人为跑步，当设置的行人运动速度小于 1.5m/s 时，设定行人为慢走，行人慢走和跑步两种状态如图 2.9 和图 2.10 所示：

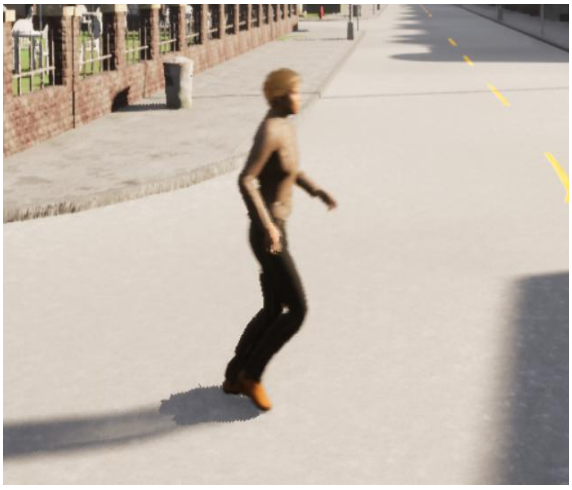


图 2.9 跑步状态

Fig. 2.9 Running State

图 2.10 慢走状态

Fig. 2.10 Slow Walking State

2.5 天气系统模型设计及参数配置

虚拟环境的搭建除了 actor 以外，必不可少的就是环境状况，CARLA 提供了天气（weather）函数来改变天气和风力，CARLA 已经编写好函数，直接调用即可。Weather 函数通过改变 cloudiness、precipitation 和 sun altitude angle 三个参数的值，可以实现在雨天、雪天、晴天，以及无风、微风、大风等状况的模拟，也能模拟白天和晚上的情况，并且物理参数会随着环境状况的改变而改变，例如摩擦因数、能见度等，可以模拟真实的环境，多种环境条件如图 2.11 和图 2.12 所示：

图 2.11 晴天条件模拟

Fig. 2.11 Simulation of Sunny Conditions

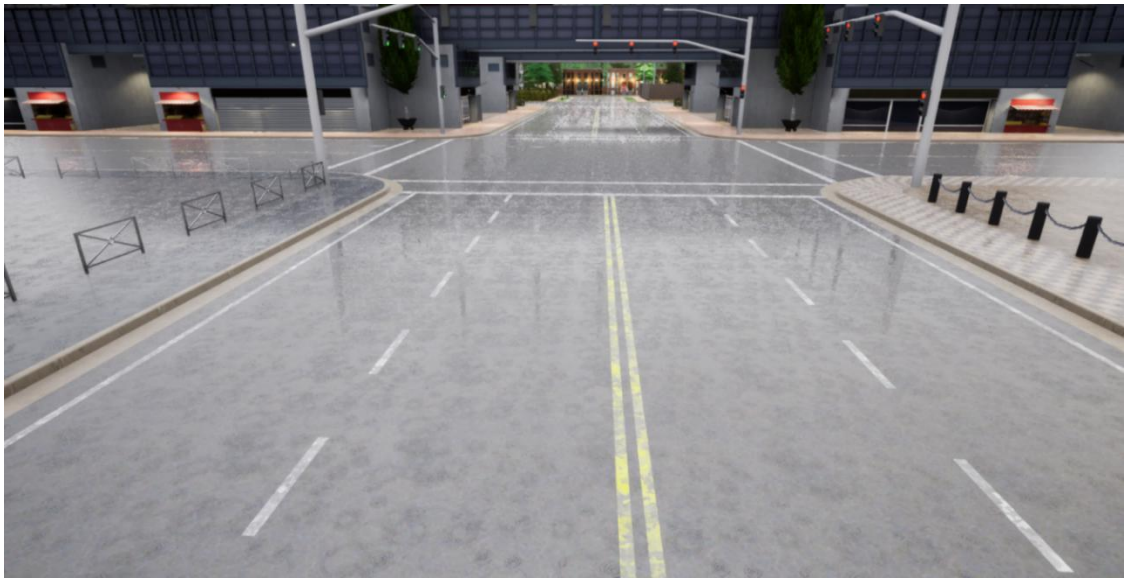


图 2.12 雨天条件模拟

Fig. 2.12 Simulation of Rainy Conditions

本文此次所选择的天气情况为晴天无云，去除了光照等环境因素对试验的影响。选择的时间为中午 12 点，即太阳位于正上方，sun altitude angle 为 90°，cloudiness 和 precipitation 参数值设置为 0。

3 无人车感知模块研究

3.1 传感器参数配置及程序开发

CARLA 的传感器种类多样，官方提供的传感器种类多样，如表 3.1 所示，本文主要用到的传感器为 RGB 相机、深度相机、分割相机、collision 和 LIDAR。接下来将重点介绍以上传感器的配置。

表 3.1 传感器种类

Table 3.1 Sensor Types

Sensor	Sensor 输出数据形式	功能	类别
RGB Camera	carla.Image	普通的 RGB 相机	Cameras
深度相机	carla.Image	以灰度图形式储存	Cameras
分割相机	carla.Image	直接输出景物分割图，不同颜色代表不同的种类	Cameras
Collision	carla.CollisionEvent	汽车发生碰撞时启动，会将事故的信息记录下来	Detector
Lane invasion	carla.LaneInvasionEvent	汽车变道时启动，记录 Lane ID 与汽车 ID	Detector
Obstacle	carla.ObstacleDetectionEvent	将可能挡在前方行驶道路上的物体记下	Detector
GNSS	carla.GNSSMeasurement	记录车子的地理位置	Other
IMU	carla.IMUMeasurement	记录汽车的轴加速度与角加速度	Other
LIDAR	carla.LidarMeasurement	激光雷达	Other
Radar	carla.RadarMeasurement	声波雷达	Other

3.1.1 摄像头模块开发及应用

摄像头的添加与汽车类似，先创建蓝图，定义一些基础属性，再定义位置，然后再选择想要的汽车安装上去。不过，这里的位置都是相对汽车中心点的位置（以米计量）。

对于相机传感器，在创建之前，需要先定义几个常量，以方便之后的调用，常量主要为图片大小，控制输出图片的尺寸及分辨率，本次选用常见尺寸：640x480。

对于 RGB 相机，首先创建蓝图并配置一些属性， 主要包括相机位置、输出图片格式和视场角，其中相机的位置相对于车中心，x 轴方向往前移动 2m，z 轴方向向上移动 2.5m。视场角设置为 110°，决定相机的视野范围。

```
camera_bp = blueprint_library.find('sensor.camera.rgb')
camera_transform = carla.Transform(carla.Location(x=2.0, z=2.5))
camera_bp.set_attribute('fov', 110)
```

其次还要对相机定义它的 callback function，定义每次仿真世界里传感器数据传回来后，要对它进行什么样的数据处理，例如将照片保存到固定文件里或者直接显示，本文采用直接显示，实时更新画面：

```
camera.listen(lambda data: process_img(data))
```

在定义 callback function 之前，需要定义函数 process_img()，该函数主要对摄像头所采集的一维数组数据进行矩阵化，转化为二维图片格式，实现了数据的可视化，转换为图像信息，输出结果如图 3.1 所示：

图 3.1 RGB 相机照片

Fig. 3.1 RGB Camera Photo

对于深度相机和语义分割相机来说，创建步骤与 RGB 相机一样，首先定义蓝图和基础数据，然后定义 callback function，将原始一维数组数据转换成图像数据后实现图像输出，深度相机和语义分割相机图片如图 3.2 和图 3.3 所示：

图 3.2 深度相机照片

Fig. 3.2 Depth Camera Photo

图 3.3 语义分割相机照片

Fig. 3.3 Semantic Segmentation of Camera Photos

3.1.2 雷达模块开发及应用

Lidar 可以设置的参数比较多且复杂，对于本文研究来说，不需要过于复杂的 Lidar 模型，只需要其能实现简单数据的采集，所以定义一些常用参数，如表 3.2 所示：

表 3.2 Lidar 参数配置

Table 3.2 Lidar Parameter Configuration

参数	数值	单位
channels	32	-
points per second	90000	pts/s
rotation frequency	40	Hz
range	20	m

然后定义 Lidar 的位置，并将其装到车上。在输出图像之前，需要调用 open3d 功能，因为 Lidar 所采集的数据是点云数据，无法显示成图像形式，需要用 open3d 进行可视化，主要设置的参数有窗口大小、背景颜色和点的尺寸，如表 3.3 所示：

表 3.3 open3d 参数配置

Table 3.3 Open3d Parameter Configuration

参数	数值

width	960
height	540
background color	[0.05, 0.05, 0.05]
point size	1

最后定义它的 callback function，将采集到的点云数据传输给 open3d 函数，经过可视化后输出图像，如图 3.4 所示：

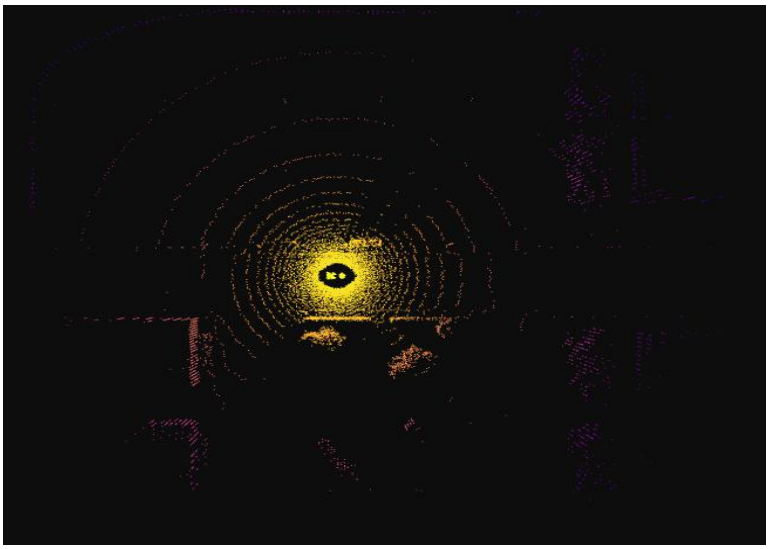


图 3.4 Lidar 图片

Fig. 3.4 Lidar Picture

3.1.3 碰撞传感器模块开发及应用

碰撞传感器（collision）的作用主要是采集碰撞信息，当汽车撞到其他物体，例如行人、栏杆、树木等等，collision 会反馈一个信号，提示发生碰撞，对于自动驾驶来说，碰撞信号尤为重要，关系着整车安全，collision 的配置较为简单，主要是位置的设置：

```
transform_collision = carla.Transform()
```

然后使能 callback function，并在发生碰撞后返回一个 warning 警告：

```
collision.listen(lambda collision_event: print("warning"))
```

3.2 基于 Haar Cascades 算法的汽车识别研究

对于自动驾驶汽车来说，图像识别对于汽车安全来说至关重要，如何识别摄像头所采集图片中的信息则是一个难点问题，故本文针对以上内容，设计了一个基于 CARLA 的汽车识别实验：将摄像头安装在汽车上，采集数据，然后对图像数据进行

处理，利用识别算法检测出目标，然后用矩形边框标识出来，实现在无人车运动过程中，实时检测视野中的汽车。本文采用的汽车识别方法为 Haar Cascades 算法[37]，主要设计思路如下：

（1）创建 serve

使用 CARLA 官方的 egg 格式文件连接客户端与服务器，然后加载道路地图，为了提高演示效果，将时间设置为中午 12 点，太阳角度为 90°，云量和降雨量参数值设置为 0。然后选择特斯拉的 Model 3 作为试验车，搭载一个 RGB 摄像头模块采集图像，因采用图像识别，并把摄像头采集到的图像尺寸设置为 1028x720。

（2）生成并配置待测目标

为了提高识别显示效果，故为试验场地添加 50 辆汽车，汽车模型采用随机生成，出生点位置也采用随机生成，设置各个待测车的状态为自由行驶状态，最大车速设置为 30km/h，车与车之间的安全距离设置为 5m，即小于安全距离时及时刹车，待大于安全距离后正常行驶，并且遵循交通规则。

（3）生成实验车

实验车生成位置采用随机选取，汽车的颜色也采用随机选取，生成的实验车如图 3.5 所示，摄像头 camera 位于试验车上，相对于车的位置坐标为 x=5.0,z=2.5，在车子的前上方，实验车开启自动驾驶功能，且遵守交通规则。

图 3.5 实验车模型

Fig. 3.5 Experimental Car Model

（4）车辆识别

要进行汽车识别，首先要获取图片， 图片实际是由像素构成，是一个巨大的像素矩阵，本文通过查阅 CARLA 官方文档，得知摄像头所采集的数据为一个一维的数据组，不是二维的图片数据格式，需要编写一个函数，实现数据格式的转换。其次通过阅读官方文档可知采集的数据类型为 bytes，即每个像素点包含四个信息 BGRA，BGR 为蓝色、绿色和红色的值，A 表示透明度，由于摄像头所采集的数据都是不透明，所以这一数据影响不大，将在处理中删除并忽略这一组值。最后本文编写了一个名为 parse_image 的函数，用来对采集的数据进行转换和处理。

parse_image 的函数的原理是，先将数据转换为一个 numpy array，然后将一维数组转换为一般图像的矩阵表示，由于设定的尺寸为 1028x720，所以该矩阵的形状为：1028x720x3，即 1028x720 个像素点，每个像素点有 RGB 三种颜色的不同强度。

有了图像数据以后，通过使用 OpenCV 中自带的 Haar Cascades 算法对图片进行识别，由于 Haar Cascades 算法需要通过大量的带有标签的数据来训练模型，所以本文采用一个已经训练好的模型（文件为 xml 格式），同时为了提高运算效率，本文采用灰度的图像数据作为模型的输入，即把 RBG 三维的数据转换为了一维的黑白数据，可以大量的降低复杂度，提高识别速度，但准确度会有所下降。

算法整体思路如图 3.6 所示：

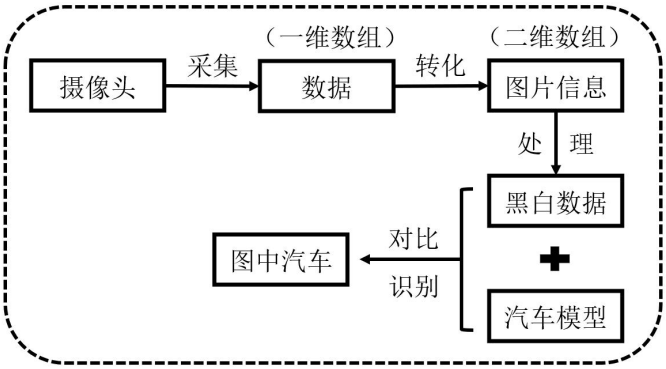


图 3.6 算法整体思路

Fig. 3.6 The Overall Idea of The Algorithm

（5）显示并标记车辆

本文通过直接显示摄像头采集回来的图像，来评价识别的准确度和效率。所以通过使用 OpenCV 提供的 imshow 方法可以将一个图像显示出来，但是在显示摄像头采集到的图像之前，如果图像中有汽车的话，需要通过 rectangle 方法在该图像上有车的位置画上蓝色的框，运行结果如图 3.7 所示。

本节通过对汽车识别算法进行研究，发现在识别方面总存在一个矛盾，即准确率和速率的矛盾，当提高了准确率后，运行速率会下降，导致视频帧数急剧下降；当提高了速率后，识别又会不准。经过多次实验后，本文找到了一个平衡点，获得了较满意的效果，本文认为可能是算法的原理导致了目前的问题，需要对算法进行进一步完善和优化。

图 3.7 汽车识别

Fig. 3.7 Car Identification

3.3 基于 HOG+SVM 和 Haar Cascades 算法的行人识别研究

本次毕业设计采用了两种行人检测方法，一种是传统的 HOG+SVM 行人识别，另一种是基于 OpenCV 内置的 Haar Cascades 算法进行行人检测。通过对比两种方法来选择合适的算法。测试地图仍选用 Town02，行人模型采用随机选取，既有成人，也有孩童，确保实验的真实性，生成点位置也采用随机选取，行人状态为慢步行走，由于地图过大，所以生成 75 名行人，设置行人可以沿着街道走，也可以横穿马路，随机选择。以下分别是两种检测方法的简介和分析。

（1）HOG+SVM 行人识别

HOG 是一种边缘特征，它利用了边缘的朝向和强度信息。HOG 特征提取原理大致可以分为 5 步：

第一步对图像进行标准化处理，降低局部阴影及背景因素的影响。

第二步计算图像梯度，即计算每一个像素点水平和垂直方向的梯度值，然后求得

梯度的方向和幅值。

第三步划分细胞单元（cell），并计算每个点处的梯度朝向和强度。

第四步组合成 block，统计 block 直方图 。把各个 cell 单元组合成相连通的区间（blocks）。

第五步梯度直方图归一化。

最后一步收集并汇总 HOG 特征。形成整个直方图特征，该特征描述了行人的基本信息，如外观、样式和动作等信息，且该特征受光照变化影响较小。HOG 特征提取原理如图 3.8 所示：

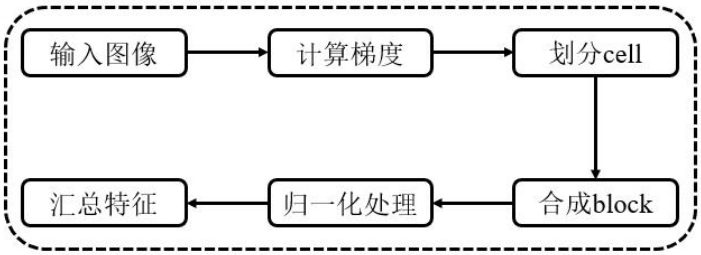


图 3.8 HOG 特征提取原理

Fig. 3.8 HOG Feature Extraction Principle

SVM 专门处理线性二分类问题。本文所使用的插件为 OpenCV，其中已经提供了计算 HOG 特征的方法,只需要进行相关参数的配置，即可采集相关的 HOG 特征，然后供 SVM 分类器使用，其中 SVM 分类器采用基于网格的方法，所以需要提供正负样本，正样本为检测目标即行人，负样本为非目标，即汽车、楼房建筑等，本文采用官方提供的已经训练好的模型，不需要对 SVM 分类器进行训练。

本文所选取窗口大小为 128x64，block 大小为 16x16，block 步长为 8 像素，cell 为 8x8 像素，每个 cell 分 9 个 bin，其他参数选择默认。本文的行人检测主要流程为：首先定义 HOG 对象，采用上述参数配置。其次配置 SVM 分类器，由于 OpenCV 自带行人分类器，所以采用默认的分类器。然后对输入的图片进行灰度化处理，并对处理后的图片，进行识别，检测出行人。最后绘制边框，圈出行人目标。

本文运用已经训练好的行人模型进行识别，并试验了两种情况下的行人识别，一种是摄像头固定，另一种是摄像头安装在汽车上，随汽车一起运动，进行检测。

从图 3.9 和 3.10 中可以看出，当摄像头位置固定时，行人识别的准确率较高，当摄像头开始运动时，能够看出整个视频的帧数急剧下降，识别准确率也较低。所以两

者对比可以看出，HOG+SVM 的优点与缺点。其中优点有：静态识别的准确率较高；受光照等因素影响小。

但同时由于 HOG+SVM 的原理问题，也导致了一系列缺点，例如：实时性差，运算速度慢，当镜头移动时，识别速度和准确率快速下降；无法处理遮挡问题；对噪点较为敏感；较小物体识别准确率低。

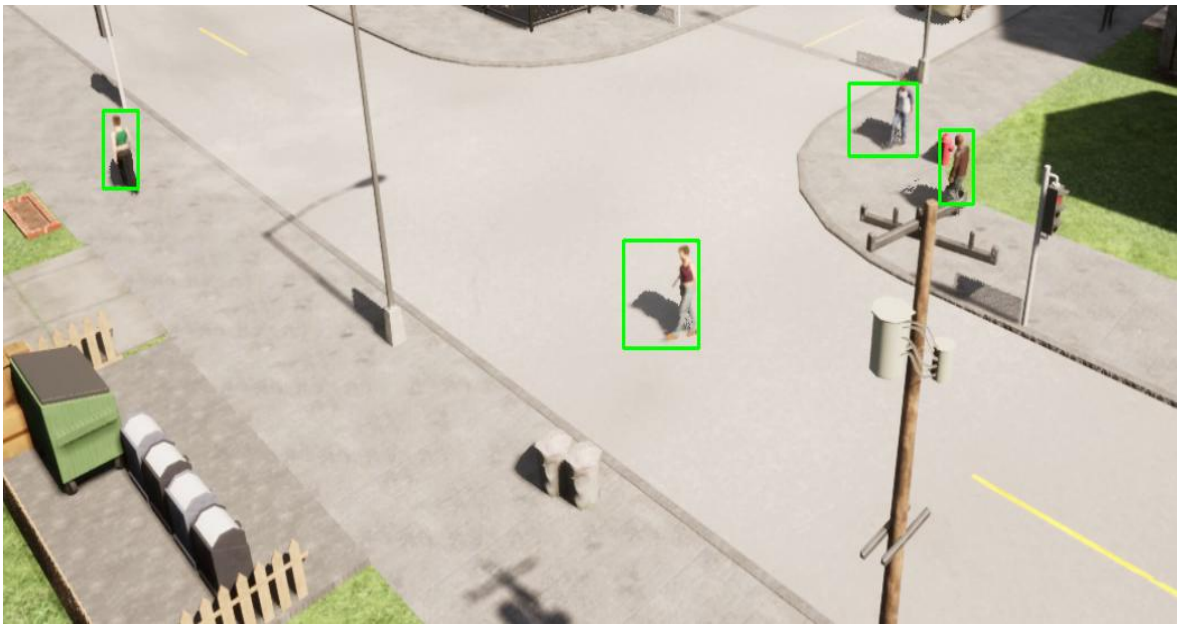


图 3.9 摄像头固定情况

Fig. 3.9 The Fixed Situation of The Camera

图 3.10 摄像头运动情况

Fig. 3.10 Camera Movement

（2）Haar Cascades 行人识别

该算法需要用大量的行人正样本图片和非行人负样本图片来进行训练，通过不断训练级联函数，然后再用于识别。该算法利用了一种名为“Boosted Cascade”的级联方式，仅用一些简单的特征参考，就能实现快速的物体检测。这是一种机器学习的方法。其中的级联函数是通过利用许多包含人体的正样本图片，与不包含人体的负样本图片来进行训练得到的。经过训练后，可以用来在其它图片中进行物体的识别。

本文采用了一个官方提供的模型，只需要导入该分类器，即可对目标进行检测，导入分类器的方法是使用 CascaClassifter 函数，即：

cv2.CascadeClassifier("haarcascade_fullbody.xml")

detectMultiScale 函数用来控制识别速度和精度，其中有几个输入量： minSize，用来配置目标最小尺寸；maxSize 即为目标最大尺寸；image 是输入的待检测图像；scaleFactor 决定了迭代的次数，及每次搜索窗口的放大比例，影响运行速度，该值一般大于 1，该值越小，迭代次数越多，检测精度也越高，但运行速度下降，该值越大，则相反，运行速度提高，但精度会有所下降；minNeighbors 是构成检测目标的相邻矩形的最小个数，该值决定了误检率，该值越大，则一幅图片中出现的矩形框越少，精度越高，该值越小则反之。通过不断调节 scaleFactor 和 minNeighbors，发现在其值为 1.03 和 4 时的效果最好。对于 Haar Cascades 行人识别算法，本文同样在两种情况下进行了模拟，结果如图 3.11 和 3.12 所示：

图 3.11 摄像头固定情况

Fig. 3.11 Camera Fixation

当摄像头静止时，从图中可以看出行人识别的准确率也较高，但是没有 HOG +SVM 算法准确率高，同时由于使用的是已经训练好的样本，对行人侧面的识别率不高，可能与训练的样本有关。

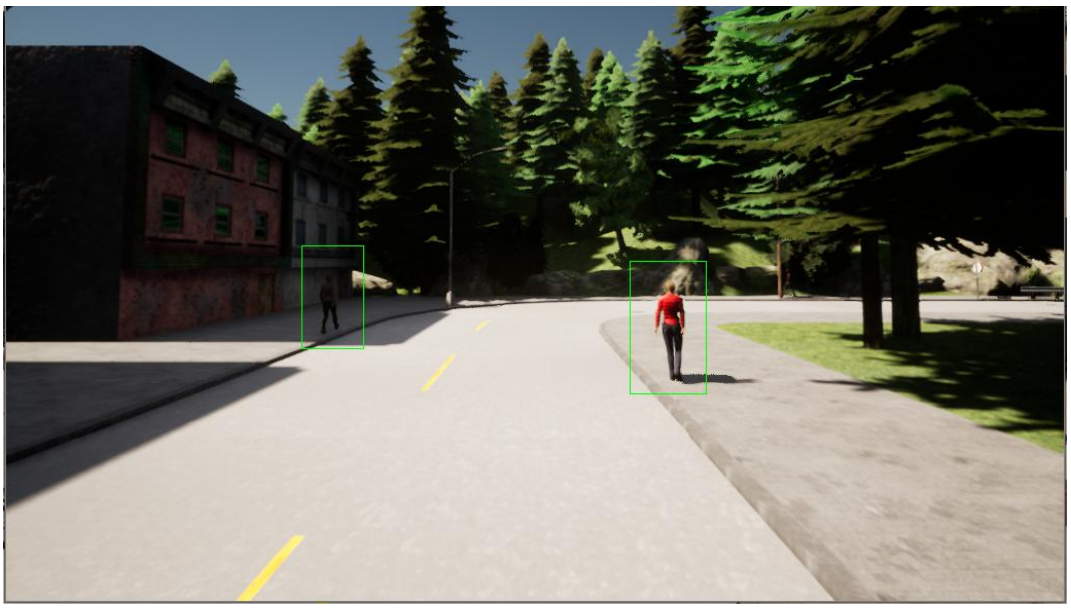


图 3.12 摄像头运动情况

Fig. 3.12 Camera Movement

当摄像头随汽车一起运动时，从图中可以看出视频帧数降低较少，整体流畅，计算速度快，检测准确率虽然没有静止时高，但也较为满意，虽然对图片中的侧向运动的行人识别率不高，但是整体运行效果比 HOG+SVM 好，可以通过自己训练样本，来改善行人的侧向识别率。

通过对结果分析，可以看出 Haar Cascades 算法的优点就是运行速度快，视频流畅，但存在识别率不理想的缺点。

通过对比两种方法，本文最后采用 Haar Cascades 算法，HOG+SVM 算法运算速度较慢，无法满足要求，所以平衡了快速性和准确性后，确定最终选用 Haar Cascades 行人识别算法。本节主要对比了两种检测的方法，本文所采用的检测方法都是传统的检测方法，传统的方法已经赶不上新技术的发展脚步了，所以要解决一些原理性的问题还需要用新的算法来替代。

4 无人车决策模块研究

所谓决策，就是根据感知模块所采集的信息，判断并确定下一步要执行的动作，如左转、右转、停车等，以及生成相应行驶的轨迹。对于本文来说，不考虑复杂情况以及轨迹生成，本文主要针对车辆行驶过程中，遇到一些突发状况时的判断和反应，如前车突然停车和行人横穿马路，用来测试无人车在行驶过程中遇到危险情况后的决策反应。

4.1 危险场景设计及避让算法编写

对于交通场景来说，危险场景就是在车辆与车辆或者车辆与行人即将相撞的场景。通过查阅相关资料，了解了简单的几种危险情况，所以本文为了验证无人车决策模块的避让算法，搭建了一个简易的危险场景模拟环境，即与停止的汽车相撞和行人横穿马路，并针对危险场景，给出了一个防撞方法，图 4.1 为模拟场景简易图，整体设计思路如下：

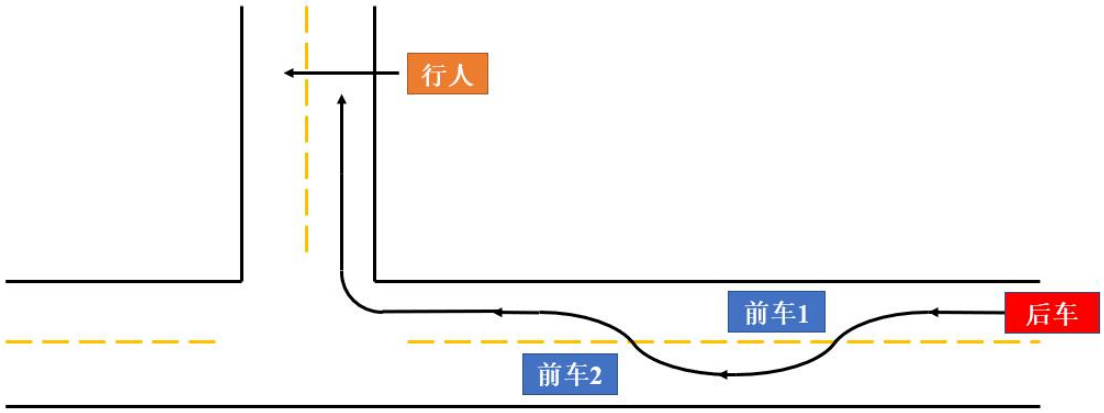


图 4.1 模拟场景

Fig. 4.1 Simulation Scene

（1）生成停止的前车与行人

本次场景总计生成两辆停止的前车和一个行人，两辆车分别位于左侧和右侧车道，两辆前车的位置分别设置为距离后车 40m 和 80m 处，两辆前车为设置为静止状态。行人位于路口处，步行通过路口。

（2）设置安全距离

这里车辆间的安全车距被设置为 10m，即只要两车纵向坐标的差值要不小于 10m，后车正常行驶。当两车小于安全距离后，开始进行决策判断。安全距离由车速和交通规则所设定，不同的车速下，安全距离也不相同。

车辆与行人间的安全距离设置为 5m，也是当小于安全距离后，进行决策判断。

（3）设定车辆避让策略

当与前车小于安全距离后，开始进行变道避让，本文只给出避让位于右侧车道的车辆的决策判断，避让位于左侧车道的车辆同理可得。车辆动作设定为左转、向右回轮和直行。

1）左转判断

当两车的距离小于安全距离时，后车开始转向。本文默认的车速不高，所以左转的判断条件很简单，当小于安全距离就开始左转，左转时的油门和转向角很重要，决定了转向的幅度和时间，本文基于舒适性考虑，采用小转角、长时间的左转方案，保证车辆行驶的舒适性，通过不断实验可得油门设置为最大的 20%，转角设置为最大转角的一半。实际运行情况如图 4.2 所示：

图 4.2 汽车左转时刻

Fig. 4.2 When The Car Turns Left

2）右转回轮判断

如果没有右转回轮，后车就会原地绕圈，不符合实际情况。所以当两车“横向距离”超过一定范围之后，后车开始回轮。何时开始右转回轮是一个难点，有两种设置方法，第一种是时间设置，第二种是距离设置，采用时间设置，受限于车速，且控制不稳定，计算较为复杂，采用距离设置，只需要考虑距离即可，故采用距离设置，根据一般的道路宽度，这里设定为 2.5 米，转向和油门的大小通过不断实验可得，油门

设置为最大的 50%，转角设置为最大转角的 50%，在这种参数配置下，右转回正的效果最好。

3）直行判断

当后车车身方向与当前道路方向平行的时候，即开始进行直行，油门按照之前直行时的默认大小，转向角回正归零。实际运行情况如图 4.3 所示：

图 4.3 汽车右转后直行时刻

Fig. 4.3 Time when The Car Goes Straight after Turning Right

综上所述，车辆避让的思路大致为：当小于纵向安全距离时，开始左转。当横向距离达到设定时，开始右转回正。当后车与当前道路方向平行时，开始直行，避让第二辆车的思路与第一辆相同。

（4）设定行人避让策略

对于行人避让来说，主要是当与行人的纵向距离小于安全距离后，车辆刹车停车，待行人通过后在正常行驶，不需要变道超越。

1）刹车判断

鉴于本次试验的车速不高，设定汽车与行人的纵向安全距离为 5m，横向安全距离为车宽的一半，只有当横纵向距离均小于安全距离时，车辆才会刹车至停止，因为要排除行人位于汽车右侧的情况，这种情况下，行人也未与车相撞。刹车类型采用脚刹，制动参数设定为最大。

2）正常行驶判断

待与行人的横纵向距离大于安全距离后，恢复正常行驶，油门和转向恢复默认值。

（5）分析与总结

通过结果来看，整体效果较为满意，当汽车遇到停止的前车时，能够进行换道超

越，且在换到过程中始终与前车保持着安全距离，未发生剐蹭现象，同时在遇到行人时，也能够及时的刹车停下，与行人间的距离也始终维持在安全范围内，证明该避让策略可行。但毕竟场景不复杂，以及避让策略单一，还不能真正的在路上使用，仍然有大量的地方需要进一步完善和改进，如本文所设计的决策，是基于规则的决策，即根据交通规则、知识、经验等建立的决策行为，是固定的，但往往真实的交通情况复杂多变，有限的决策行为无法满足全部情况，所以需要增加基于学习算法的决策，即根据多种环境情况，进行自主学习，遇到不同的情况自动地输出相应的决策规划。由此可见，需要改进的地方还有很多，仍需不断提高。

4.2 避让原理总结及进一步应用

避让原理主要是采用距离判断，本文通过坐标变换获取的距离，如图 4.4 所示，CARLA 中所获取的目标坐标为世界坐标，需要经过坐标变换后转为相机坐标，然后在经过变换得到图像坐标，进而在图中得以标记和显示。

（1）坐标变换

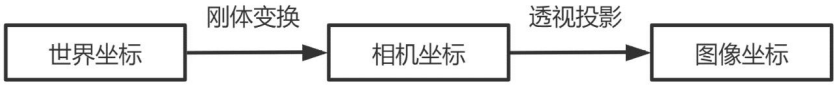


图 4.4 坐标变换

Fig. 4.4 Coordinate Transformation

1）世界坐标与相机坐标变换

坐标系都是相对的，空间中的任意一个坐标系都可以通过旋转和平移转换成新的坐标系。从世界坐标系变换到相机坐标系属于刚体变换，只需要进行旋转和平移操作即可，如图 4.5 所示：

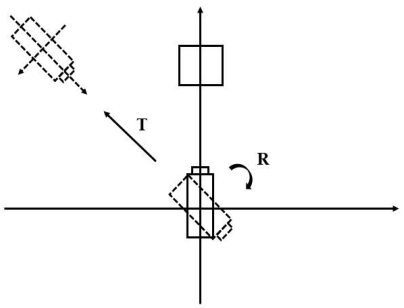


图 4.5 旋转平移变换

Fig. 4.5 Rotation and Translation Transformation

二者之间的数学表达式为：

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} \#(4.1)$$

$$X = RX_C + T \#(4.2)$$

其中，R 代表对坐标的旋转操作，T 代表对坐标的平移操作。R、T 这两个参数为摄像机的外参数。

旋转操作不单指绕一个轴，而是绕 X、Y、Z 三个坐标轴变换的效果相叠加。例如绕 X 轴旋转 RX：

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\,\theta & sin\,\theta \\ 0 & -sin\,\theta & cos\,\theta \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} \#(4.3)$$

R=RX*RY*RZ。其由三个方向的旋转角共同控制，故具有三个方向的自由度。

2）相机坐标与图像坐标转换

从相机坐标转换到图像坐标，是三维到二维的变换，属于光学中的投影，遵照小孔成像原理，摄像机依据此原理，进行投影变换，如图 4.6 所示。其中 f 为摄像机的焦距，其属于摄像机的内参数。

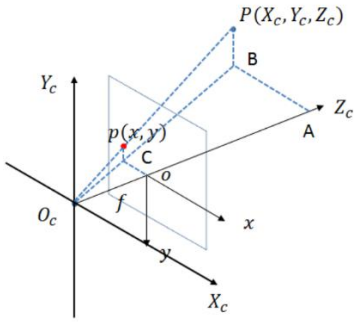


图 4.6 变换原理

Fig. 4.6 Transformation Principle

故可得变换关系为：

$$z_c \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_C \end{bmatrix} \#(4.4)$$

综上本文所采用的避让判断是距离判断，首先通过坐标变换获取汽车周围行人和汽车位置，然后再进行坐标变换判断汽车或行人是否在相机视野内，最后再根据距自己的距离来进行避让。

（2）坐标变换的进一步应用

本文的避让原理采用了坐标变换，主要用来获取两者之间的距离和判断是否在视野范围内，但是坐标变换还有进一步的应用，即绘制物体的边界框和汽车运动的轨迹路线，如图 4.7 和 4.8 所示。归根结底就是三个坐标系之间的相互转换，绘制轨迹路线能够判断汽车是否按照规定路线正常行驶。绘制物体的边界框能够帮助汽车判断碰撞的范围，合理设置安全距离。

图 4.7 绘制物体边界框

Fig. 4.7 Draw The Bounding Box of The Object

图 4.8 绘制汽车运动轨迹路线

Fig. 4.8 Draw The Route of The Car Movement

5 无人车控制模块研究

无人车辆控制核心主要是横向控制和纵向控制，纵向控制用来控制速度，使之稳定在期望速度，横向控制用来控制转向，使无人车始终按照预定轨迹行驶。本文采用的是车辆运动学模型，不考虑动力学模型，控制模块如图 5.1 所示：

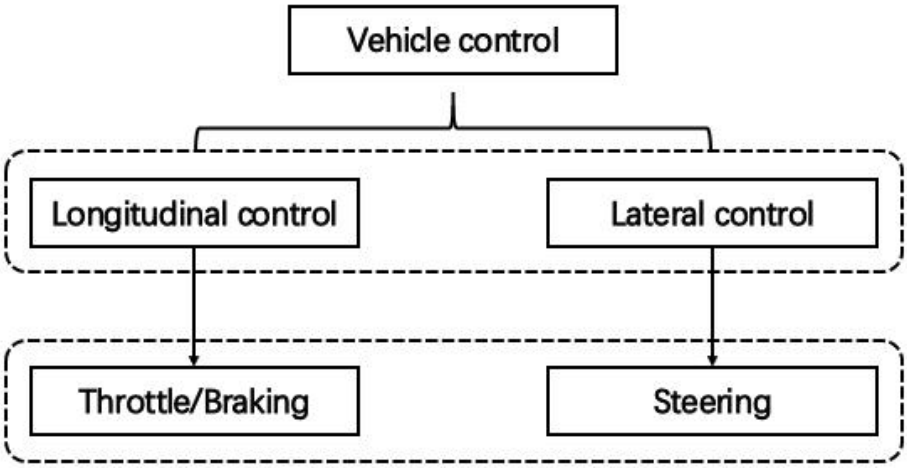


图 5.1 无人车控制模块结构

Fig. 5.1 Unmanned Vehicle Control Module Structure

5.1 基于 PID 控制的车辆纵向控制研究及程序开发

车辆的纵向控制主要是速度控制，车辆的速度通过油门和制动以保持在参考速度。本文通过 PID 控制器不断对比期望速度，使得车速稳定在理想值附近，原理如图 5.2 所示：

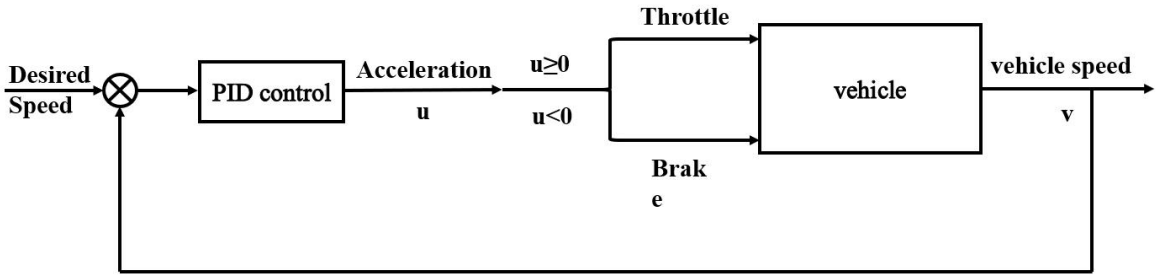


图 5.2 纵向控制原理图

Fig. 5.2 Schematic Diagram of Longitudinal Control

所以控制算法为：

$$u = k_P(v_d - v) + k_I \int_0^t (v_d - v)\mathrm{d}t + K_D \frac{\mathrm{d}(v_d - v)}{\mathrm{d}t} \#(5.1)$$

其中v_d为期望速度、v为车速、u为加速度输入。如果$u \geq 0$，则 Throttle = u，Brake = 0，如果$u < 0$,则 Throttle = 0，Brake = $-u$，通过控制 Throttle 和 Brake 来控制车速维持在期望速度。

本次实验的期望车速设定为 20km/h，k_P、k_I、K_D的值通过不断实验可得，分别为 0.8、0.5 和 0，在此配置下的车速波动小，总体效果较为满意。

5.2 三种车辆横向控制算法研究及程序开发

车辆横向控制的三种方法：Stanley、Pure Pursuit 和 MPC，本文就三种方法分别进行了试验，并得到了相应的结果，并对三种算法进行了对比和总结。在实施三种控制方法之前，要先搭建自行车模型，自行车模型如图 5.3 所示，自行车模型有动力学模型和运动学模型，本文采用运动学模型控制，自行车模型受以下方程组控制[38]：

$$\dot{x}_C = v\,cos\,(\theta + \beta)\,\#(5.2)$$

$$\dot{y}_c = v\,sin\,(\theta + \beta)\,\#(5.3)$$

$$\dot{\theta} = \frac{v\,cos\,\beta\,tan\,\delta}{L}\#(5.4)$$

$$\dot{\delta} = \omega\#(5.5)$$

$$\beta = tan^{-1}\left(\frac{l_r\,tan\,\delta}{L}\right)\#(5.6)$$

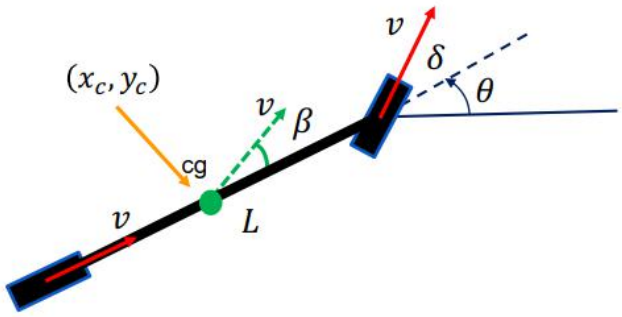


图 5.3 自行车模型

Fig. 5.3 Bicycle Model

输入为自行车速度v和转向角比率ω。输入也可以直接是转向角δ，而不是简化情况下的比率。

（1）纯追踪控制

Pure Pursuit 是几何路径跟踪控制器，仅使用车辆运动学的几何形状和参考路径来进行跟踪。该算法使用预视点，该点是车辆前方参考路径上的固定距离，如图 5.4 所示。 车辆需要使用之前计算的转向角前进到该点。

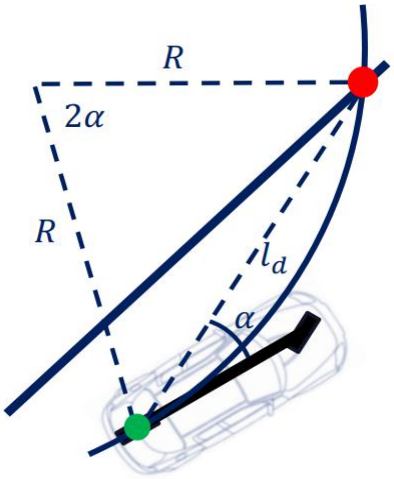


图 5.4 Pure Pursuit 原理

Fig. 5.4 Principle of Pure Pursuit

在这种方法中，后轴的中心用作车辆上的参考点。在上图中，目标点被选为红色点。后轴与目标点之间的距离表示为l_d，设定目标是使车辆以正确的角度转向，然后继续前进。因此，几何关系图如图 5.4 所示，车辆的前进方向与前视线之间的夹角称为α。因为车辆是刚体，并且绕圆行进。该圆的瞬时旋转中心（ICR）如图 5.4 所示，半径表示为R。

1）计算公式

$$\frac{l_d}{sin\,2\alpha} = \frac{R}{sin\left(\frac{\pi}{2} - \alpha\right)} \#(5.7)$$

$$\frac{L_d}{2\,sin\,\alpha\,cos\,\alpha} = \frac{R}{cos\,\alpha} \#(5.8)$$

$$\frac{ld}{sin\,\alpha} = 2R \#(5.9)$$

$$k = \frac{1}{R} = \frac{2\,sin\,\alpha}{l_d} \#(5.10)$$

k是曲率。 使用自行车模型：

$$R = L/tan\,\delta\,\#(5.11)$$

因此，转向角δ可计算为：

$$\delta = arctan\left(\frac{2L\,sin\,\alpha}{l_d}\right)\#(5.12)$$

Pure Pursuit 控制器是一种简单的控制。它忽略了车辆上的动力，并假定车轮处于防滑状态。此外，如果将其调整为低速运行，则控制器在高速运行时会非常危险。一种改进是基于车辆的速度来改变超前距离：

$$l_d = k_{\mathrm{dd}} v_f \#(5.13)$$

因此，转向角更改为：

$$\delta = tan^{-1}\left(\frac{2L\,sin\,\alpha}{k_{\mathrm{dd}} v_f}\right)\#(5.14)$$

本次横向控制算法验证采用 racetrack 地图，根据给定赛道进行仿真验证，运行结果如图 5.5 所示：

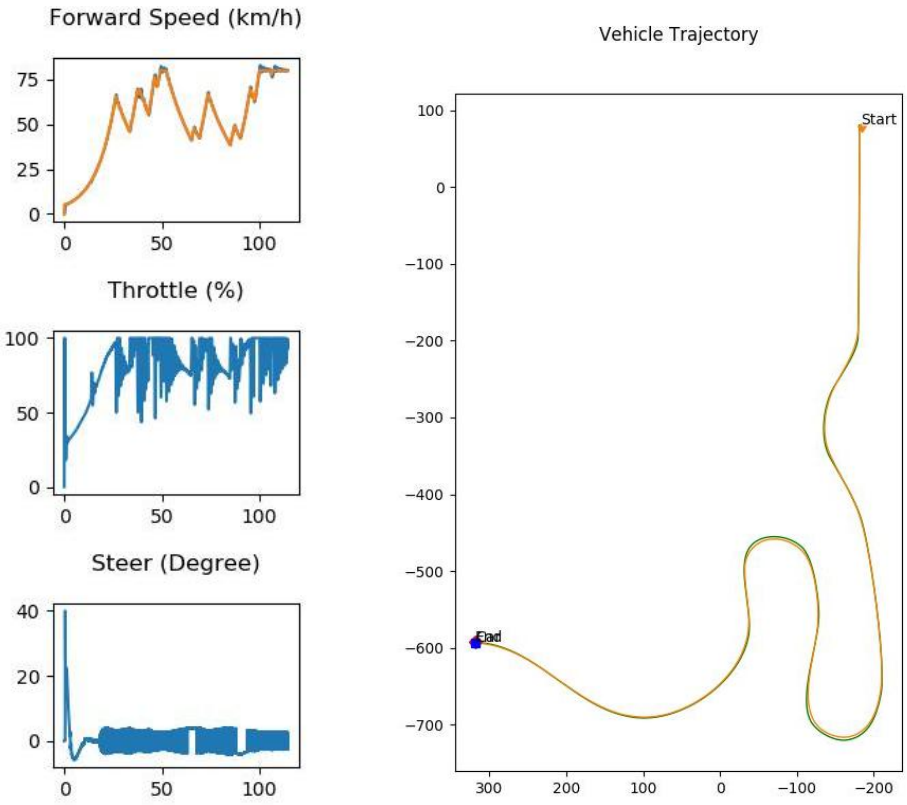


图 5.5 Pure Pursuit 运行结果

Fig. 5.5 Pure Pursuit Running Results

2）分析与总结

本文分析了该算法有效的原因，首先跨轨误差定义为航向矢量和目标点之间的横向距离，如图 5.6 所示。如果跨轨误差较小，则意味着车辆会更好地沿着路径行驶。

跨轨误差和曲率k之间的关系：

$$sin\,\alpha = \frac{\mathrm{e}}{l_d}\#(5.15)$$

$$k = \frac{2}{l_d^2}\mathrm{e}\#(5.16)$$

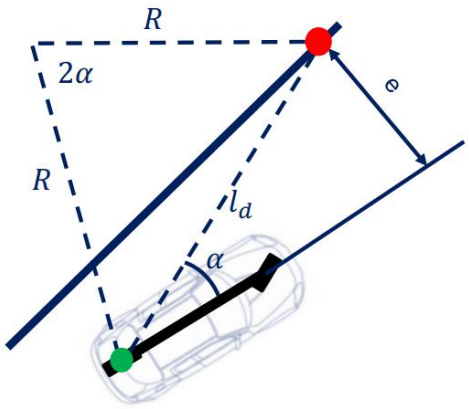


图 5.6 Pure Pursuit 中的跨轨误差

Fig. 5.6 Cross-track Error in Pure Pursuit

上式表明，曲率k与跨轨误差成正比。随着误差的增加，曲率也会增加，从而使车辆更积极地返回路径。比例增益 $2/l_d^2$ 可以自己调整。简而言之，Pure Pursuit 作为对交叉轨迹误差起作用的转向角的比例控制器，可以通过控制转向角来减小跨轨误差，因此该方法有效。

在该段代码中，设置了最小前视距离为 2m，前视距离越大，运行轨迹就会表现得更加平滑，但更加平滑的结果就是在某些急剧的转角处会存在转向不足的情况。所以该参数的调整，对于车辆稳定性来说至关重要，在本文最初的实验中，使用的l_d都是几何关系计算所得，但是发现车辆运行极其不稳定，虽然也能沿规定路径行驶，但转向变化幅度大，速度一快就会跑偏，后来改为了 $l_d = k_{\mathrm{dd}}v_f + 2$ 后，转向幅度明显改善，稳定性大大提高，也中和了一部分的转向不足。

纯追踪控制算法虽然达到了基本的跟踪要求，但是当存在显著的速度改变时会遇到问题，该算法仅适用于低速情况。并且选择l_d的值的方法也多种多样，本文选择的是一种与速度有关的表示，但其实查阅资料后发现，方法并不唯一，取决于多种因素，所以优化该算法的方法就是如何优化前视距离函数。

由于 Pure Pursuit 控制器原理的局限性，导致该算法无论如何优化，也仅适用于低速情况，通过结果图 5.5 也可以看出，其行驶路径与规定路径具有肉眼可见的误差，可以看出其行驶并不是很稳定，从 steer 图中也可以看出，steer 的值一直在波动，并

不稳定，所以还需要寻求新的方法来实现轨迹跟踪。

（2）斯坦利控制

其次，本文将讨论 Stanley Control。这是斯坦福大学的无人驾驶汽车团队使用的路径跟踪方法。与以后轴为参考点的纯追赶方法不同，斯坦利方法以前轴为参考点，它同时考虑了航向误差和跨轨误差。 在这种方法中，跨轨误差定义为路径上最近的点与车辆前轴之间的距离，如图 5.7 所示：

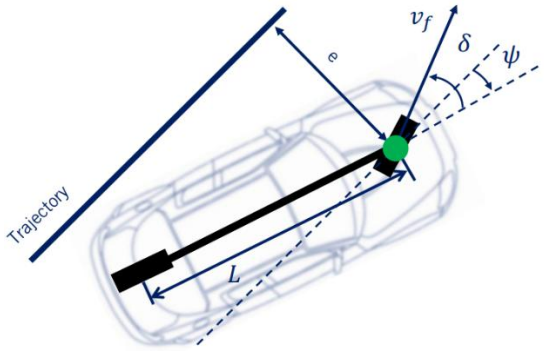


图 5.7 Stanley 原理

Fig. 5.7 Stanley Principle

1）计算公式

由原理图可得，$\psi(t)$是轨迹航向与车辆航向之间的角度。转向角标记为δ。斯坦利方法有三种直观的转向规律。

首先，消除航向误差，$\delta(t) = \psi(t)$。

其次，消除跨轨误差。该步骤是找到路径与车辆之间的最接近点，其被表示为 $\mathrm{e}(t)$。转向角可以按以下方式进行校正：

$$\delta(t) = tan^{-1}\left(\frac{k\mathrm{e}(t)}{v_f(t)}\right) \tag{5.17}$$

最后一步是遵守最大转向角范围。这意味着$\delta(t) \in [\delta_{min}, \delta_{max}]$，这样就可以得到：

$$\delta(t) \in [\delta_{min}, \delta_{max}]\#(5.18)$$

$$\delta(t) = \psi(t) + tan^{-1}\left(\frac{k\mathrm{e}(t)}{v_f(t)}\right)\#(5.19)$$

对比纯追踪控制算法，可以看出 Stanley 方法考虑的更全面，使用 Stanley 控制算法，也可以完成百分之百的航点跟踪。此外，通过观察发现车辆行驶的速度比纯追踪控制器稳定得多，尤其是转弯时，并且 Stanley 方法的实际行驶路径与预设路径相差

很小，且 steer 值波动小，行驶稳定。仿真结果如图 5.8 所示：

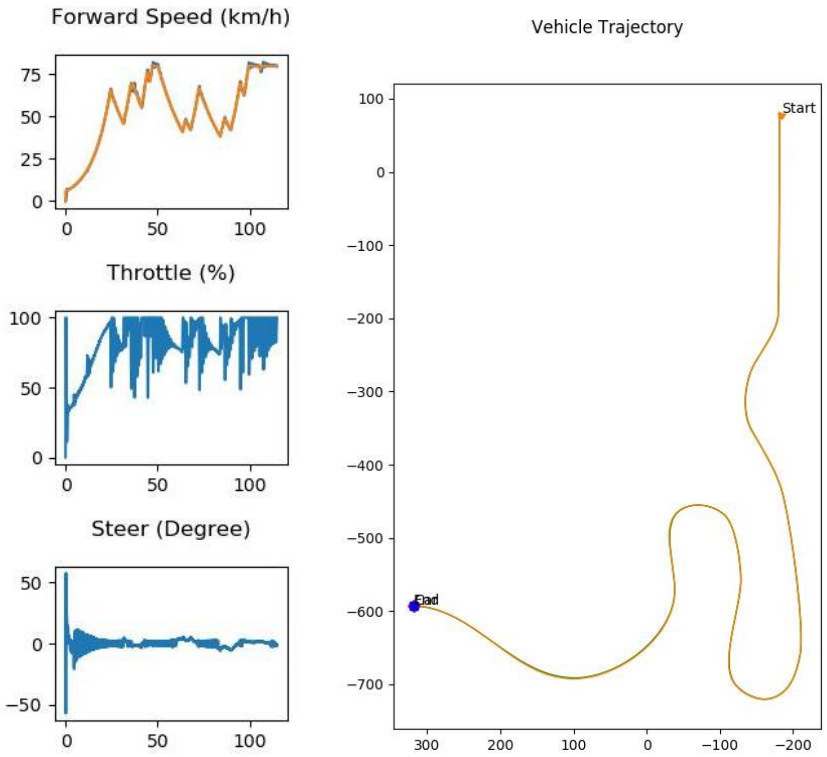


图 5.8 Stanley 运行结果

Fig. 5.8 Stanley Running Results

2）分析与总结

Stanley 控制器不仅考虑航向误差，而且还纠正了跨轨误差，正是这个原因使得其比 Pure Pursuit 控制器稳定且误差小。首先，如果航向误差大而跨轨误差小，则表示ψ较大，因此转向角也会变大，并朝相反方向转向以校正航向误差，这可以使车辆定向，并与轨迹相同，如图 5.9 所示：

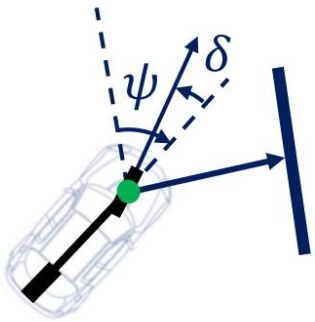


图 5.9 Stanley 中的航向误差和跨轨误差

Fig. 5.9 Heading Error and Cross-track Error in Stanley

对于航向误差大、跨轨误差小的情况，汽车运动过程可以绘制如图 5.10 所示：

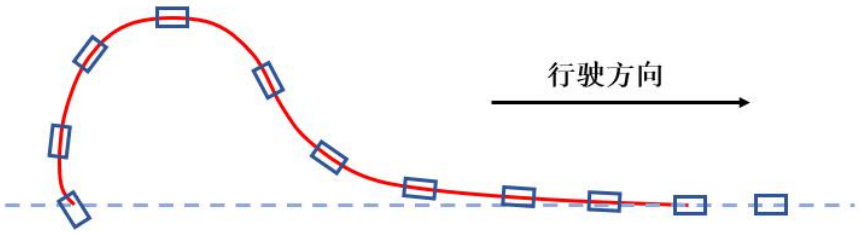


图 5.10 航向误差大、跨轨误差小的情况

Fig. 5.10 Large Heading Error and Small Cross-track Error

其次，如果跨轨误差较大且航向误差较小，则可以

$$tan^{-1}\left(\frac{ke(t)}{v_f(t)}\right) \approx \frac{\pi}{2} \#(5.20)$$

$$\delta(t) = \psi(t) + \frac{\pi}{2} \#(5.21)$$

假设航向误差$\psi(t) = 0$，$\delta(t)$将为π / 2。当航向因转向角而改变时，航向校正会抵消跨轨校正，并将转向角驱动回零。当车辆接近路径时，跨轨误差下降，转向角开始按以下方式校正航向对齐，此方案的运动过程可以绘制如图 5.11 所示：

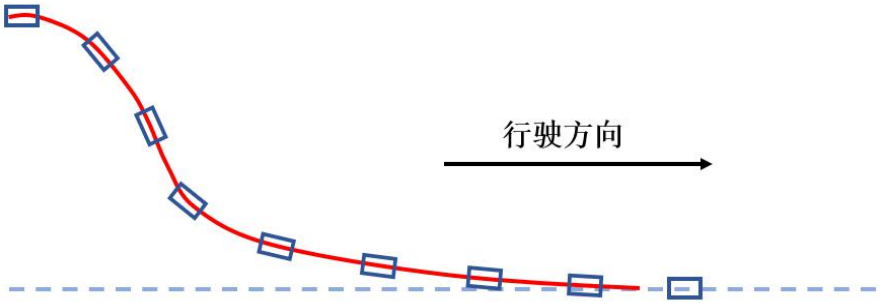


图 5.11 航向误差小、跨轨误差大的情况

Fig. 5.11 Small Heading Error and Large Cross-track Error

简而言之，Stanley 控制器是一种简单但有效且稳定的方法，可用于以后的控制。相同的是这两种方法都是几何控制器。

（3）模型预测控制

1）成本函数

在此项目中，要控制车辆遵循赛道，因此成本函数应包含与参考路径的偏差，偏差越小效果越好。同时，为了使车上的乘客在旅途中感到舒适，控制命令幅度的最小化，较小的转向效果更好。这类似于最佳控制理论的优化问题，并在控制性能和输入积极性之间进行权衡。在这两个目标之上，可以得出成本函数：

$$J(x(t),U)=\sum_{j=t}^{t+T-1}\delta x_{j|t}^{T}Q\delta x_{j|t}+u_{j|t}^{T}Ru_{j|t}\ \#(5.22)$$

在上式中，给定转向角，δx是预测点和参考点之间的距离，如下所示：

$$\delta x_{j|t}=x_{j|t,des}-x_{j|t}\#(5.23)$$

u是转向输入。要求得δx，首先要具有预测模型。

2）预测模型

MPC 的主要概念是使用工厂模型来预测系统的未来发展[39]。在这种情况下，本文通过使用简单的运动自行车模型，相关方程组由前文可知，则(t+1)时刻状态为:

$$x_{(t+1)}=x_{(t)}+\dot{x}\Delta t \quad (5.24$$

$$y_{(t+1)}=y_{(t)}+\dot{y}\Delta t \quad (5.25$$

$$\theta_{(t+1)}=\theta_{(t)}+\dot{\theta}\Delta t \quad (5.26$$

$$\delta_{(t+1)}=\delta_{(t)}+\dot{\delta}\Delta t \quad (5.27$$

状态 X 为[x，y，θ，δ]，θ为航向角，δ为转向角。输入U为[v，φ]，v为速度，φ为转向率。

3）MPC 结构

现在，有了成本函数和预测模型。下一步是寻求最佳输入，以优化成本函数。MPC 的具体计算过程如下：

首先，假设转向角范围为$\delta(t)\in[\delta_{min},\delta_{max}]$。使用一种简单的方法将模型的输入（即转向角$\delta$）离散为具有相同间隔的值。

然后，使用上述模型和输入 get，可以得到预测输出[[x，y，θ，δ]。

最后一步是选择成本函数的最小值及其相应的输入δ。（在这种情况下，将转向角定义为从$\delta_{min}=-1.2$ 到$\delta_{max}=1.2$，以 0.1 弧度间隔进行划分，然后将其放入成本函数和 for 循环中，以找到最小值及其对应的输入δ）并在每个时间步骤中重复上述过程。

整体过程如图 5.12 所示：

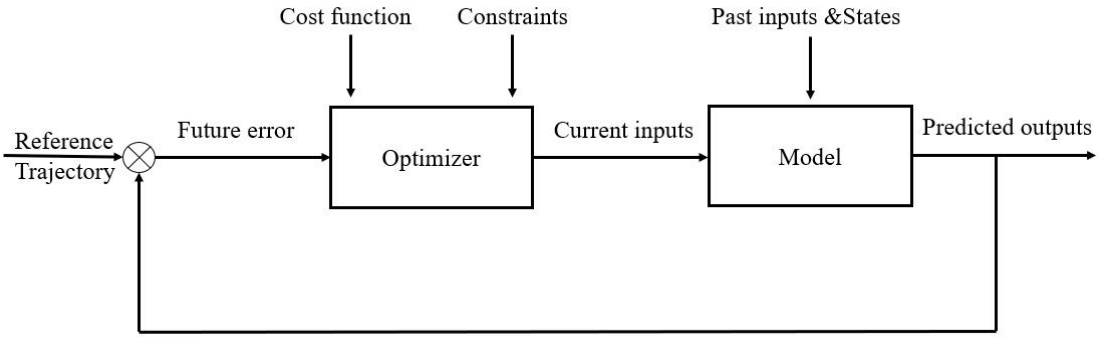


图 5.12 MPC 结构

Fig. 5.12 MPC Structure

4）分析与总结

如图 5.13 所示，通过使用 MPC 控制器也完成了百分之百的航点跟踪。但是车辆的行驶并不像使用 Stanley Control 那样稳定。

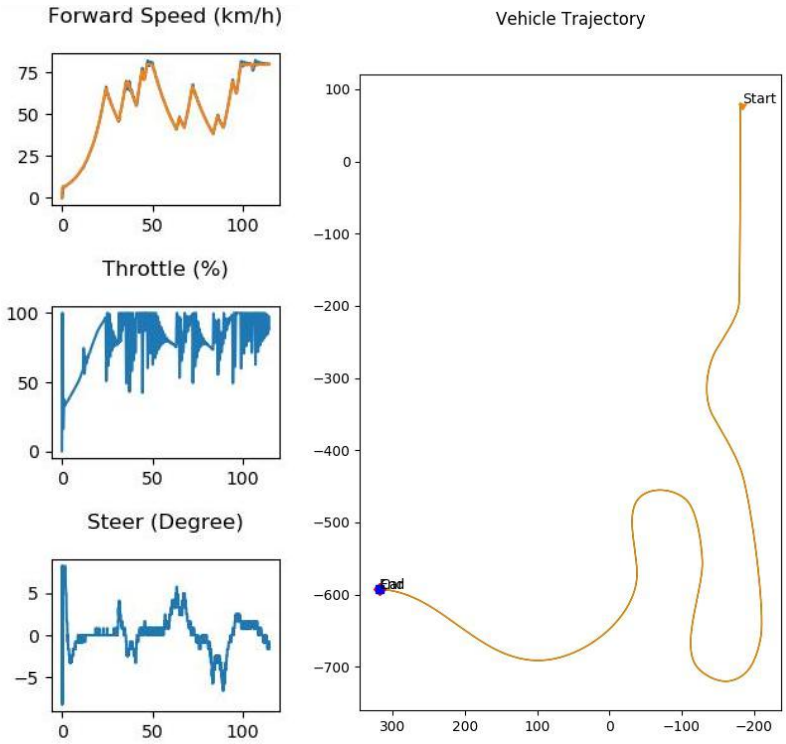


图 5.13 MPC 运行结果

Fig. 5.13 MPC Running Result

通过分析成本函数，发现在其中设置了输入δ，因为不想采取太大的行动，否则可能会导致乘客感到不适。但是引入δ所带来的不利因素就是不稳定，所以可以通过改进δ来实现稳定运行，由于本文只是基于 MPC 的概念来编写程序，所以没有将其优化至完美，需进一步改进。

MPC 具有很多优点。它可以应用于线性或非线性模型。并且具有简单明了的公式，可以处理多个约束。同时成本函数可以针对不同的目标进行设计。但是它也有计算量大的缺点。特别是对于非常普遍的非线性模型，MPC 必须通过数值求解，而不能提供封闭形式的解决方案。

6 仿真结果分析

（1）感知方面

如图 6.1 所示，所构建的无人车，能够实现汽车与行人的检测，能够在摄像头所拍摄的图片中用蓝色和绿色边框圈出汽车与行人，当遇到汽车或行人时，能够及时避让，待其通过后再继续前进。还有无人车能够实现交通信号灯的识别，红灯亮起时能够停下，待绿灯亮起时，再继续前进，总体效果还算较为理想。

图 6.1 感知识别结果

Fig. 6.1 Perceptual Recognition Results Perception

但同时也存在一些问题，最重要的就是识别的准确率问题，可以从图片中看出，汽车的识别准确率还较为满意，但是当图片中出现一些类似长方形的物体或者是地面上长方形的影子时，容易与汽车混淆，识别成汽车，本文分析最直接的原因是识别算法的原因，因为对于汽车识别来说，采用了一维的黑白数据来进行识别，虽然提高了运算速度，但是却牺牲了准确率，还有可能是模型训练的不够充分，数据库不够大，也可能是原因之一。对于行人检测来说，在图中可以看出，对于行人的侧面识别不够理想，本文分析可能与数据库中大部分图片都是人的正面或者背面的原因，对于侧面的训练不够多，所以导致侧面识别率低。总而言之，解决识别率的根本途径还是得从

算法入手，应用深度学习新型的识别算法也许能从根本上解决问题，达到快速和准确同时兼顾。

（2）决策方面

决策方面，总体来说，结果较为满意，所搭建的无人车，能够计算与汽车和行人之间的距离，做到小于安全距离内停车，当大于安全距离再正常行驶，以及当前车停止不动一段时间后，能够从左侧变道超车，继续前进，汽车与行人的避让策略如图 6.2 和 6.3 所示：

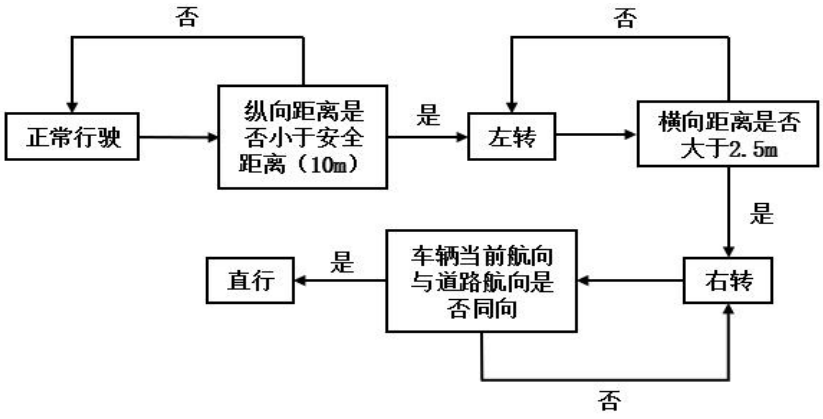


图 6.2 汽车避让策略

Fig. 6.2 Car avoidance strategy

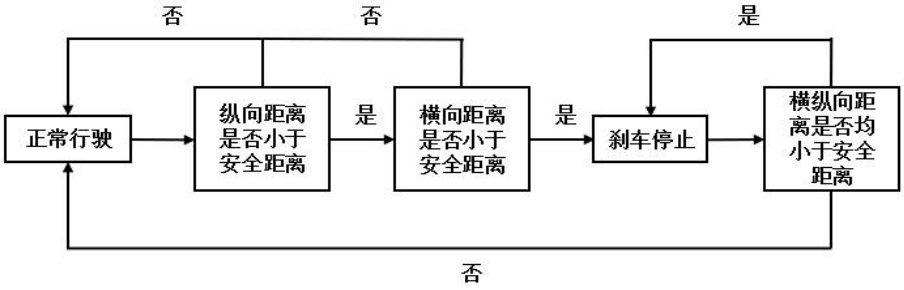


图 6.3 行人避让策略

Fig. 6.3 Pedestrian avoidance strategy

本文所采用的是距离控制，距离控制的原理就是坐标变换，对物体在 CARLA 中的世界坐标进行坐标变换后，可以得到物体相对相机的坐标和图像内的二维坐标。本文的决策方面只用到了相对于相机的坐标，但其实该原理还有一个重要的作用，就是可以进行绘制物体的边界框。物体边界框的绘制十分重要，与感知不同的是，该边界框的绘制是三维的，汽车、行人识别绘制的边界框是二维的，三维的优势就是可以对物体的碰撞进行分析，是立体的，虽然在图中所看到的还是二维投影图，但其实物体

的三维边框已经绘制出，可以与感知识别相结合，进一步完善无人车的感知模块。

决策方面也存在一些问题，例如决策行为单一，不灵活，不能做到行为预测以及处理突发情况，当突然有危险情况发生时，只能做到直线刹车这一简单行为。由于本文没有考虑过多的危险场景，所以导致了决策方面的单一化，这一点有待进一步完善。

（3）控制方面

本文试验了三种横向控制的方法，其中纯追踪控制原理简单，实现起来容易，但是缺点就是转向角波动大，汽车总是左右摇晃，虽然幅度不是很大，但是很影响乘坐的舒适性。斯坦利算法整体较为稳定，相对于纯追踪，该算法引入了航向误差，使得对于一些情况的处理更加平顺，变化不剧烈。MPC 算法相对于前两种，是原理上的改变，可以用于高速的情况，本文只是基于概念而编写的，没有考虑更多复杂的因素，所以最后的效果可能没有斯坦利的好。最终本文采用的车辆横向控制算法是斯坦利算法，是几何跟踪算法，转向波动小，驾驶舒适性好。车辆纵向控制算法采用的是 PID 控制算法，整体车速较为稳定，车速波动小。从结果来看汽车能够按照预定轨迹进行跟踪行驶，始终行驶在车道中央，且转弯时转向平顺。

控制方面存在问题主要是算法问题，由于采用的是几何跟踪算法，所以适用于低速情况，当车辆高速行驶时，就会发生转向剧烈变化，最终撞向其他物体，导致跟踪失败，所以针对高速不适用的情况，还需要采用新型的控制算法来解决此问题。

总体来说，可以看出整体运行效果较为满意，基本功能要求均已实现，但通过分析发现仍存在一些问题需要进一步完善和优化。

总　结

本文主要针对无人车虚拟环境搭建和无人车控制算法，首先搭建了虚拟测试环境，然后对无人车自动驾驶技术的三个方面进行了研究，即感知、决策和控制。探究了如何实现一辆汽车的自动驾驶，并围绕三个技术方向进行了 Python 程序编写。

本文的主要研究成果如下：

（1）实现了虚拟环境搭建，尽可能逼真的还原真实的交通场景，包括道路、树木、建筑、汽车和行人的建模。

（2）完成了 RGB 相机、深度相机、语义分割相机和激光雷达多种传感器的配置，实现了相机图像的采集和显示，以及雷达点云图的显示。

（3）实现了基本识别，包括对汽车、行人和交通信号灯的识别，能够从摄像头所拍摄的视频中找出汽车和行人，并用不同颜色的边框圈出。

（4）完成了危险场景的搭建，并通过 Python 编程实现了汽车遇到简单危险情况时的决策反应，使汽车安全行驶。

（5）完成了无人车的轨迹跟踪， 纵向采用 PID 算法控制速度，横向采用斯坦利算法控制转向，实现了无人车以规定速度按照预定路线行驶，并且尽可能的减小了转向的波动，保证了舒适性的要求。

（6）实现了无车人在所搭建的虚拟交通环境中的安全运行，实现了基础的自动驾驶功能。

本文的整体结果较为满意，实现了基础的自动驾驶功能，但通过对仿真结果的仔细分析，也存在以下一些问题：

（1）识别的准确率。当遇到光照、天气等因素影响，以及一些相似的物体存在时，会错当成行人或汽车，还有当汽车或行人与自身相距较远，在图片上显示较小时，也会识别不出。

（2）决策算法。本文所采用的决策算法较为简单，难以应对一些复杂的场景，同时当多种情况同时发生时，是优先避让行人还是汽车，也会无法应对，无法判断事

件的优先级。

（3）控制算法。本文所采用的车辆横向控制为斯坦利控制，属于几何路径跟踪算法，适用于低速情况，高速时无法使用，会发生跑偏，需要进一步优化。

以上的问题，也就是本文未来的研究方向，主要研究如何优化识别算法，兼顾准确率与快速性；如何应对道路复杂多变情况，并针对每一种情况，计算出最好的应对策略；解决跟踪算法高速不适用的问题，进一步优化跟踪算法。

参 考 文 献

[1] Koenig N, Howard A. Design and use paradigms for gazebo, an open-source multi-robot simulator[C]. IEEE, 2004, 3: 2149-2154.

[2] Rohmer E, Singh S P N, Freese M. V-REP: A versatile and scalable robot simulation framework[C]. IEEE, 2013: 1321-1326.

[3] Michel O. Cyberbotics Ltd. Webots™: professional mobile robot simulation[J]. International Journal of Advanced Robotic Systems, 2004, 1(1): 5.

[4] Sotiropoulos T, Waeselynck H, Guiochet J, et al. Can robot navigation bugs be found in simulation? an exploratory study[C]. IEEE, 2017: 150-159.

[5] Timperley C S, Afzal A, Katz D S, et al. Crashing simulated planes is cheap: Can simulation detect robotics bugs early?[C]. IEEE, 2018: 331-342.

[6] Robert C, Sotiropoulos T, Waeselynck H, et al. The virtual lands of Oz: testing an agribot in simulation[J]. Empirical Software Engineering, 2020: 1-30.

[7] Gladisch C, Heinz T, Heinzemann C, et al. Experience paper: Search-based testing in automated driving control applications[C]. IEEE, 2019: 26-37.

[8] Gambi A, Mueller M, Fraser G. Automatically testing self-driving cars with search-based procedural content generation[C]. International Symposium on Software Testing and Analysis, 2019: 318-328.

[9] Mullins G E, Stankiewicz P G, Gupta S K. Automated generation of diverse and challenging scenarios for test and evaluation of autonomous vehicles[C]. IEEE, 2017: 1443-1450.

[10] Tuncali C E, Pavlic T P, Fainekos G. Utilizing S-TaLiRo as an automatic test generation framework for autonomous vehicles[C]. IEEE, 2016: 1470-1475.

[11] Rocklage E, Kraft H, Karatas A, et al. Automated scenario generation for regression testing of autonomous vehicles[C]. IEEE, 2017: 476-483.

[12] Uber. Self-Driving Simulation[EB/OL]. https://www.uber.com/us/en/atg/research-and-development/simulation, 2018.

[13] NVIDIA. NVIDIA DRIVE Constellation: Virtual reality autonomous vehicle simulator[EB/OL]. https://www.nvidia.com/en-us/self-driving-cars/drive-constellation, 2019.

[14] Waymo. Waymo safety report: On the road to fully self-driving[EB/OL]. https://waymo.com/safety, 2018.

[15] Dosovitskiy A, Ros G, Codevilla F, et al. CARLA: An open urban driving

simulator[C]. PMLR, 2017: 1-16.
[16] LG. LGSVL Simulator[EB/OL]. https://www.lgsvlsimulator.com, 2019.
[17] Shah S, Dey D, Lovett C, et al. Airsim: High-fidelity visual and physical simulation for autonomous vehicles[C]. Field and service robotics, 2018: 621-635.
[18] Li W, Pan C W, Zhang R, et al. AADS: Augmented autonomous driving simulation using data-driven algorithms[J]. Science robotics, 2019, 4(28).
[19] 乔维高, 徐学进. 无人驾驶汽车的发展现状及方向[J]. 上海汽车, 2007, (07): 40-43.
[20] 徐渊, 许晓亮, 李才年, 姜梅, 张建国. 结合 SVM 分类器与 HOG 特征提取的行人检测[J]. 计算机工程, 2016, 42(01): 56-60.
[21] Dosovitskiy A, Ros G, Codevilla F, et al. CARLA: An open urban driving simulator[C]. PMLR, 2017: 1-16.
[22] CARLA Simulator. How to make a new map with RoadRunner[EB/OL]. https://carla.org/, 2020.
[23] Rong G, Shin B H, Tabatabaee H, et al. Lgsvl simulator: A high fidelity simulator for autonomous driving[C]. IEEE, 2020: 1-6.
[24] Shah S, Dey D, Lovett C, et al. Airsim: High-fidelity visual and physical simulation for autonomous vehicles[C]. Field and service robotics, 2018: 621-635.
[25] Tobin J, Fong R, Ray A, et al. Domain randomization for transferring deep neural networks from simulation to the real world[C]. IEEE, 2017: 23-30.
[26] Barnich O, Van Droogenbroeck M. ViBe: A universal background subtraction algorithm for video sequences[J]. IEEE Transactions on Image processing, 2010, 20(6): 1709-1724.
[27] Dalal N, Triggs B. Histograms of oriented gradients for human detection[C]. IEEE, 2005, 1: 886-893.
[28] Felzenszwalb P F, Girshick R B, McAllester D, et al. Object detection with discriminatively trained part-based models[J]. IEEE transactions on pattern analysis and machine intelligence, 2009, 32(9): 1627-1645.
[29] Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks[J]. Advances in neural information processing systems, 2012, 25: 1097-1105.
[30] Ren S, He K, Girshick R, et al. Faster R-CNN: towards real-time object detection with region proposal networks[J]. IEEE transactions on pattern analysis and machine intelligence, 2016, 39(6): 1137-1149.
[31] Liu W, Anguelov D, Erhan D, et al. Ssd: Single shot multibox detector[C]. European conference on computer vision, 2016: 21-37.
[32] Lin T Y, Dollár P, Girshick R, et al. Feature pyramid networks for object detection[C]. Proceedings of the IEEE conference on computer vision and pattern recognition, 2017:

2117-2125.
[33] Redmon J, Farhadi A. Yolov3: An incremental improvement[J]. arXiv preprint arXiv:1804.02767, 2018.
[34] Hoffmann G M, Tomlin C J, Montemerlo M, et al. Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing[C]. IEEE, 2007: 2296-2301.
[35] Coulter R C. Implementation of the pure pursuit path tracking algorithm[R]. Carnegie-Mellon UNIV Pittsburgh PA Robotics INST, 1992.
[36] Falcone P, Borrelli F, Asgari J, et al. Predictive active steering control for autonomous vehicle systems[J]. IEEE Transactions on control systems technology, 2007, 15(3): 566-580.
[37] Viola P, Jones M. Rapid object detection using a boosted cascade of simple features[C]. CVPR 2001. IEEE, 2001, 1: I-I.
[38] Steven Waslander, Jonathan Kelly. Introduction to Self-Driving Cars[EB/OL]. https://www.coursera.org/learn/intro-self-driving-cars, 2020.
[39] 陈威, 廖文浩, 刘明春. 基于MPC的自动驾驶车辆横向路径跟踪控制[J]. 南昌大学学报(工科版), 2020, 42(03): 279-288.

致　谢

通过三个月的学习，我的毕业设计圆满完成，虽然经历了许多困难，但是在各位同学和老师的帮助下，都顺利的解决了。所以我要对曾经帮助过我的人表示感谢。

首先我要感谢我的企业指导老师许晟杰工程师，在本次毕业设计中，不但给予了我许多学习资料，而且还为我提供了舒适的工作环境和电脑。由于本次题目对电脑的性能要求较高，所以感谢老师能给我提供一台性能卓越的电脑。许老师认真负责，不定时的会向我询问进度情况，并为我的问题提供解决方案，最重要的是教会了我遇到问题时如何查找资料，这一点对我的学习成长有很大影响，我能够遇到问题时独立解决，最后再次对许老师表示衷心的感谢！

其次我要感谢我的校内指导老师武春龙老师，在本次毕业设计中，武老师对我的论文撰写提供了很大帮助，许老师帮助我解决技术问题，武老师帮助我解决论文格式和论文逻辑等问题，武老师帮助我理清了论文逻辑，并对我的论文提出了许多宝贵意见，非常感谢武老师的帮助！

最后我要感谢我的同学们，每当我想放弃时，总有你们在鼓励我继续前进，没有你们的帮助与支持，我也不会成功走到现在，我要对曾经帮助过我的同学表示衷心的感谢！

附录 A　避让控制程序

```
import glob
import os
import sys
import cv2
import copy
try:
    sys.path.append(glob.glob('../carla/dist/carla-*%d.%d-%s.egg' %
(
        sys.version_info.major,
        sys.version_info.minor,
        'win-amd64' if os.name == 'nt' else 'linux-x86_64'))[0])
except IndexError:
    pass

import carla
from carla import Transform, Location, Rotation
from carla import Map
from carla import Vector3D
```

```
import numpy as np
import random
import time
actor_list = []
def process_img(image):
        global Image_Array
        i = np.array(image.raw_data)  # convert to an array
        i2 = i.reshape((image.height, image.width, 4))  # was flatt
    ened, so we're going to shape it.
        i3 = i2[:, :, :3]  # remove the alpha (basically, remove the
    4th index  of every pixel. Converting RGBA to RGB)
        cv2.imshow('Carla Tutorial', i3 )
        cv2.waitKey(1)
        return i3/255.0  # normalize
try:
    client = carla.Client('localhost', 2000)
    client.set_timeout(10.0)
    world = client.load_world('Town02')
    blueprint_library = world.get_blueprint_library()
    vehicle_bp1 = blueprint_library.filter('tt')[0]
    vehicle_bp1.set_attribute('color', '255,255,255')
    spawn_point_v1 = Transform(Location(x=-3.7, y=300, z=1),
```

```
                                Rotation(pitch=0, yaw=-90, roll=0))
    vehicle1 = world.spawn_actor(vehicle_bp1, spawn_point_v1)
    actor_list.append(vehicle1)
    vehicle_bps = world.get_blueprint_library().filter('vehicle.*.*')
    vehicle_bps = [x for x in vehicle_bps if int(x.get_attribute('num
ber_of_wheels')) == 4]
    vehicle_bp = np.random.choice(vehicle_bps)
    spawn_point_v6 = Transform(Location(x=-3.7, y=270, z=1),
                               Rotation(pitch=0, yaw=-90, roll=0))
    vehicle2 = world.spawn_actor(vehicle_bp, spawn_point_v6)
    actor_list.append(vehicle2)
    vehicle_bp = np.random.choice(vehicle_bps)
    spawn_point_v3 = Transform(Location(x=-7.5, y=230, z=1),
                               Rotation(pitch=0, yaw=-90, roll=0))
    vehicle3 = world.spawn_actor(vehicle_bp, spawn_point_v3)
    actor_list.append(vehicle3)
    ped_blueprints = world.get_blueprint_library().filter("walker.*")

    spawn_point_v2 = Transform(Location(x=20, y=195, z=2),
                               Rotation(pitch=0, yaw=-90, roll=0))
    player = world.spawn_actor(random.choice(ped_blueprints), sp
awn_point_v2)
```

```
    player_control = carla.WalkerControl()
    player_control.speed = 0
    pedestrian_heading = -90
    player_rotation = carla.Rotation(0, pedestrian_heading, 0)
    player_control.direction = player_rotation.get_forward_vector()


    player.apply_control(player_control)
    actor_list.append(player)
    spectator = world.get_spectator()
    spawn_point_v8 = Transform(Location(x=-3.7, y=195, z=5),
                               Rotation(pitch=0, yaw=45, roll=0))
    spectator.set_transform(spawn_point_v1)
    time.sleep(3)
    while True:
        time2 = time.time()
        x_v1 = vehicle1.get_location().x
        y_v1 = vehicle1.get_location().y
        if (y_v1 - 270) > 12:
            vehicle1.apply_control(carla.VehicleControl(throttle=0.7,
steer=0.0))
        if 0 < (y_v1 - 270) <= 12:
            time1 = time.time()
```

```
77.             while True:
78.                 x_v1 = vehicle1.get_location().x
79.                 vehicle1.apply_control(carla.VehicleControl(throttle=
    0.2, steer=-0.6)) # 左转
80.                 if abs(-3.7 - x_v1) > 0.9:
81.                     vehicle1.apply_control(carla.VehicleControl(throttl
    e=0.5, steer=0.26))
82.                     break
83.             while True:
84.                 y_v1 = vehicle1.get_location().y
85.                 if abs(270 - y_v1) < 1.0:
86.                     vehicle1.apply_control(carla.VehicleControl(throttl
    e=0.3, steer=0.0))
87.                 if (270 - y_v1) > 2:
88.                     y_v1 = vehicle1.get_location().y
89.                     x_v1 = vehicle1.get_location().x
90.                     if (x_v1 + 7.5) > 0:
91.                         vehicle1.apply_control(carla.VehicleControl(thr
    ottle=0.4, steer=-0.1))
92.                     if (x_v1 + 7.5) == 0:
93.                         vehicle1.apply_control(carla.VehicleControl(thr
    ottle=0.5, steer=0.0))
```

```
94.                     if (x_v1 + 7.5) < 0:
95.                         vehicle1.apply_control(carla.VehicleControl(thr
    ottle=0.4, steer=0.1))
96.                     if 0 < (y_v1 - 230) < 12:
97.                         vehicle1.apply_control(carla.VehicleControl(thr
    ottle=0.4, steer=0.0))
98.                         break
99.             time2 = time.time()
100.                print((time2-time1)*1000,"ms")
101.            if 0 < (y_v1 - 230) < 10:
102.                while True:
103.                    x_v1 = vehicle1.get_location().x
104.                    vehicle1.apply_control(carla.VehicleControl(thrott
    le=0.4, steer=0.7))
105.                    if abs(7.5 + x_v1) > 1.2:
106.                        vehicle1.apply_control(carla.VehicleControl(thr
    ottle=0.7, steer=-0.24))
107.                        break
108.                while True:
109.                    y_v1 = vehicle1.get_location().y
110.                    if abs(230 - y_v1) < 2:
```

```
111.                    vehicle1.apply_control(carla.VehicleControl(thr
    ottle=0.2, steer=0.0))
112.                if (230 - y_v1) > 5:
113.                    y_v1 = vehicle1.get_location().y
114.                    x_v1 = vehicle1.get_location().x
115.                    if (x_v1 + 3.7) > 0:
116.                        vehicle1.apply_control(carla.VehicleControl(
    throttle=0.3, steer=-0.1))
117.                    if (x_v1 + 3.7) == 0:
118.                        vehicle1.apply_control(carla.VehicleControl(
    throttle=0.5, steer=0.0))
119.                    if (x_v1 + 3.7) < 0:
120.                        vehicle1.apply_control(carla.VehicleControl(
    throttle=0.3, steer=0.1))
121.                    if 0 < (y_v1 - 192.5) < 20:
122.                        vehicle1.apply_control(carla.VehicleControl(
    throttle=0.5, steer=0.0))
123.                        break
124.            if 0 < (y_v1 - 193) < 2:
125.                while True:
126.                    x_v1 = vehicle1.get_location().x
127.                    y_v1 = vehicle1.get_location().y
```

```
                vehicle1.apply_control(carla.VehicleControl(throttle=0.2, steer=0.7))
                time.sleep(1.86)
                vehicle1.apply_control(carla.VehicleControl(throttle=0.5, steer=0))
                player_control = carla.WalkerControl()
                player_control.speed = 1
                player_control.direction = player_rotation.get_forward_vector()
                player.apply_control(player_control)
                break

            if 0 < (20 - x_v1) < 7:
                if player_control.speed:
                    y_v1 = vehicle1.get_location().y
                    vehicle1.apply_control(carla.VehicleControl(throttle=0, steer=0, brake=0.3))
                    time.sleep(3)
                    vehicle1.apply_control(carla.VehicleControl(throttle=0.5, steer=0.0, brake=0))
                    player_control = carla.WalkerControl()
                    player_control.speed = 0
```

```
145.                 player_control.direction = player_rotation.get_for
    ward_vector()
146.                 player.apply_control(player_control)
147.         if cv2.waitKey(25) & 0xFF == ord('q'):
148.         # destroy cv2 window when 'q' is pressed
149.             cv2.destroyAllWindows()
150.             break
151.     finally:
152.         for actor in actor_list:
153.             actor.destroy()
154.         print("All cleaned up!")
```

附录 B　汽车和行人识别程序

```
import copy
import random
import glob
import os
import sys
import logging
try:
    sys.path.append(glob.glob('C:/wangpengchao/WindowsNoEditor
/PythonAPI/carla/dist/carla-*%d.%d-%s.egg' % (
        sys.version_info.major,
        sys.version_info.minor,
        'win-amd64' if os.name == 'nt' else 'linux-x86_64'))[0])
except IndexError:
    pass
import carla
import cv2
import numpy as np
from carla import ColorConverter as cc
```

```
import argparse
import time
argparser = argparse.ArgumentParser(
        description=__doc__)
argparser.add_argument(
        '--filterw',
        metavar='PATTERN',
        default='walker.pedestrian.*',
        help='pedestrians filter (default: "walker.pedestrian.*")')
args = argparser.parse_args()
client = carla.Client("localhost", 2000)
client.set_timeout(10.0)
world = client.load_world('Town02_Opt', carla.MapLayer.Buildings | carla.MapLayer.All)
world.unload_map_layer(carla.MapLayer.All)
debug = world.debug
world_snapshot = world.get_snapshot()

for actor_snapshot in world_snapshot:
    actual_actor = world.get_actor(actor_snapshot.id)
    if actual_actor.type_id == 'traffic.traffic_light':
        tran = actor_snapshot.get_transform().location
```

```
            tran.z += 1.0
            debug.draw_box(carla.BoundingBox(tran,carla.Vector3D(0.5,0.5,2)),actor_snapshot.get_transform().rotation, 0.05, carla.Color(255,0,0,0),0)

weather = carla.WeatherParameters(
    cloudiness=0.0,
    precipitation=0.0,
    sun_altitude_angle=90.0)

world.set_weather(weather)

picked_spawn_points = []
blueprintsWalkers = world.get_blueprint_library().filter(args.filterw)
# get vehicle blueprint
model3_bp = world.get_blueprint_library().find('vehicle.tesla.model3')
model3_bp.set_attribute('color', '255,255,255')
camera_bp = world.get_blueprint_library().find('sensor.camera.rgb')
camera_bp.set_attribute('image_size_x', str(1028))
```

```
camera_bp.set_attribute('image_size_y', str(720))

# get avaliable spawn points
spawn_points = world.get_map().get_spawn_points()
random.shuffle(spawn_points)

# spawn vehicle
model3_spawn_point = spawn_points[-1]
model3 = world.spawn_actor(model3_bp, model3_spawn_point)
picked_spawn_points.append(model3_spawn_point)

NUMBER_OF_VEHICLES = 20
number_of_walkers = 75
SpawnActor = carla.command.SpawnActor
walkers_list = []
all_id = []

vehicle_bps = world.get_blueprint_library().filter('vehicle.*.*')
vehicle_bps = [x for x in vehicle_bps if int(x.get_attribute('number_
    of_wheels')) == 4]
vehicle_list = []
```

```
for i in range(NUMBER_OF_VEHICLES):
    point = spawn_points[i]
    vehicle_bp = np.random.choice(vehicle_bps)
    try:
        vehicle = world.spawn_actor(vehicle_bp, point)
        picked_spawn_points.append(point)
        vehicle_list.append(vehicle)
    except:
        print('failed')
        pass

tm = client.get_trafficmanager()
tm.global_percentage_speed_difference(20.0)
tm_port = tm.get_port()

for v in vehicle_list:
    v.set_autopilot(True, tm_port)
    tm.ignore_lights_percentage(v,100)
    tm.distance_to_leading_vehicle(v,5)
    tm.vehicle_percentage_speed_difference(v,-20)

model3.set_autopilot(True)
```

```
percentagePedestriansRunning = 0  # how many pedestrians wil
    l run
    percentagePedestriansCrossing = 75  # how many pedestrian
    s will walk through the road
    # 1. take all the random locations to spawn
    point = []
    for i in range(number_of_walkers):
        point.append(spawn_points[i+25])
    # 2. we spawn the walker object
    batch = []
    walker_speed = []
    for i in range(number_of_walkers):
        walker_bp = random.choice(blueprintsWalkers)
        # set as not invincible
        if walker_bp.has_attribute('is_invincible'):
            walker_bp.set_attribute('is_invincible', 'false')
        # set the max speed
        if walker_bp.has_attribute('speed'):
            if (random.random() > percentagePedestriansRunning):

                # walking
```

```
117.                walker_speed.append(walker_bp.get_attribute('speed')
    .recommended_values[1])
118.            else:
119.                # running
120.                walker_speed.append(walker_bp.get_attribute('speed')
    .recommended_values[2])
121.        else:
122.            print("Walker has no speed")
123.            walker_speed.append(0.0)
124.        batch.append(SpawnActor(walker_bp, point[i]))
125.    results = client.apply_batch_sync(batch, True)
126.    walker_speed2 = []
127.    for i in range(len(results)):
128.        if results[i].error:
129.            logging.error(results[i].error)
130.        else:
131.            walkers_list.append({"id": results[i].actor_id})
132.            walker_speed2.append(walker_speed[i])
133.    walker_speed = walker_speed2
134.    # 3. we spawn the walker controller
135.    batch = []
```

```
    walker_controller_bp = world.get_blueprint_library().find('controller.ai.walker')
    for i in range(len(walkers_list)):
        batch.append(SpawnActor(walker_controller_bp, carla.Transform(), walkers_list[i]["id"]))
    results = client.apply_batch_sync(batch, True)
    for i in range(len(results)):
        if results[i].error:
            logging.error(results[i].error)
        else:
            walkers_list[i]["con"] = results[i].actor_id
    # 4. we put altogether the walkers and controllers id to get the objects from their id
    for i in range(len(walkers_list)):
        all_id.append(walkers_list[i]["con"])
        all_id.append(walkers_list[i]["id"])
    all_actors = world.get_actors(all_id)
    # wait for a tick to ensure client receives the last transform of the walkers we have just created
    world.wait_for_tick()
    # 5. initialize each controller and set target to walk to (list is [controler, actor, controller, actor ...])
```

```
153.    # set how many pedestrians can cross the road
154.    world.set_pedestrians_cross_factor(percentagePedestriansCross
        ing)
155.    for i in range(0, len(all_id), 2):
156.        # start walker
157.        all_actors[i].start()
158.        # set walk to random point
159.        all_actors[i].go_to_location(world.get_random_location_from
        _navigation())
160.        # max speed
161.        all_actors[i].set_max_speed(float(walker_speed[int(i / 2)]))
162.
163.    def parse_image(image):
164.        global Image_Array
165.        global Car_Cascade
166.        global Cars
167.        global classfier
168.        global faceRects
169.        array = np.frombuffer(image.raw_data, dtype=np.dtype("u
        int8"))
170.        array = np.reshape(array, (image.height, image.width, 4))
```

```
171.        array = array[:, :, :3]
172.        array_gray = cv2.cvtColor(array, cv2.COLOR_BGR2GRAY)

173.        Car_Cascade = cv2.CascadeClassifier('cars.xml')
174.        Cars = Car_Cascade.detectMultiScale(array_gray, 1.3, 5,
    minSize=(72, 72))
175.        classfier = cv2.CascadeClassifier("haarcascade_fullbody.xml")

176.        faceRects = classfier.detectMultiScale(array_gray, scaleFact
    or=1.05, minNeighbors=3, minSize=(12, 12), maxSize=(200, 20
    0))
177.        Image_Array = array
178.
179.    ints = 0
180.    # spawn camera
181.    camera = world.spawn_actor(camera_bp,
182.                               carla.Transform(carla.Location(x=1, z=2.
    0), carla.Rotation(yaw=180,pitch=-60)),
183.                               model3,
184.                               carla.AttachmentType.SpringArm
185.                               )
186.    camera.listen(lambda image:parse_image(image))
```

```
187.    Image_Array = None
188.    Cars = None
189.
190.    while True:
191.
192.        if Image_Array is not None:
193.            image = copy.copy(Image_Array)
194.            if Cars is not None:
195.                for (x, y, w, h) in Cars:
196.                    cv2.rectangle(image,(int(x),int(y)),(int(x+w),int(y
    +h)),(255,0,0), 2)
197.            if len(faceRects) > 0:
198.                ints += 1
199.                for faceRect in faceRects:
200.                    x, y, w, h = faceRect
201.                    cv2.rectangle(image, (x - 10, y - 10), (x + w +
    10, y + h + 10), (0, 255, 0), 1)
202.                if ints >= 3:
203.                    ints = 0
204.            cv2.imshow('Carla Tutorial', image)
205.        if model3.is_at_traffic_light():
206.            traffic_light = model3.get_traffic_light()
```

```
207.            if traffic_light.get_state() == carla.TrafficLightState.Red:
208.                traffic_light.set_state(carla.TrafficLightState.Green)
209.        if cv2.waitKey(25) & 0xFF == ord('q'):
210.            # destroy cv2 window when 'q' is pressed
211.            cv2.destroyAllWindows()
212.            vehicle_list.destroy()
213.            walkers_list.destroy()
214.            break
```

附录 C　坐标变换程序

```
1. import math
2. import glob
3. import os
4. import sys
5. import logging
6. import argparse
```

```
7. try:
8.     sys.path.append(glob.glob('../carla/dist/carla-*%d.%d-%s.egg' %
   (
9.         sys.version_info.major,
10.         sys.version_info.minor,
11.         'win-amd64' if os.name == 'nt' else 'linux-x86_64'))[0])
12. except IndexError:
13.     pass
14.
15. import carla
16. from carla import Transform, Location, Rotation
17. import weakref
18. import random
19. import cv2
20. from imutils.object_detection import non_max_suppression
21. try:
22.     import pygame
23.     from pygame.locals import K_ESCAPE
24.     from pygame.locals import K_SPACE
25.     from pygame.locals import K_a
26.     from pygame.locals import K_d
27.     from pygame.locals import K_s
```

```
28.     from pygame.locals import K_w
29. except ImportError:
30.     raise RuntimeError('cannot import pygame, make sure pyga
    me package is installed')
31.
32. try:
33.     import numpy as np
34. except ImportError:
35.     raise RuntimeError('cannot import numpy, make sure nump
    y package is installed')
36.
37. VIEW_WIDTH = 1920//2
38. VIEW_HEIGHT = 1080//2
39. VIEW_FOV = 90
40. NUMBER_OF_VEHICLES = 20
41. BB_COLOR = (248, 64, 24)
42. BB_COLOR1 = (0, 0, 255)
43. red = carla.Color(255, 0, 0)
44. green = carla.Color(0, 255, 0)
45. blue = carla.Color(47, 210, 231)
46. cyan = carla.Color(0, 255, 255)
47. yellow = carla.Color(255, 255, 0)
```

```
48. orange = carla.Color(255, 162, 0)
49. white = carla.Color(255, 255, 255)
50. percentagePedestriansRunning = 0  # how many pedestrians wil
    l run
51. percentagePedestriansCrossing = 20  # how many pedestrians w
    ill walk through the road
52. SpawnActor = carla.command.SpawnActor
53. number_of_walkers = 20
54. # ==============================================
    ============================
55. # -- ClientSideBoundingBoxes ----------------------
    ------------------------
56. # ==============================================
    ============================
57. argparser = argparse.ArgumentParser(
58.         description=__doc__)
59. argparser.add_argument(
60.         '--filterw',
61.         metavar='PATTERN',
62.         default='walker.pedestrian.*',
63.         help='pedestrians filter (default: "walker.pedestrian.*")')
64. args = argparser.parse_args()
```

```
65.
66. class ClientSideBoundingBoxes(object):
67.     """
68.     This is a module responsible for creating 3D bounding boxes a
    nd drawing them
69.     client-side on pygame surface.
70.     """
71.
72.     @staticmethod
73.     def get_bounding_boxes(vehicles, camera):
74.         """
75.         Creates 3D bounding boxes based on carla vehicle list and
    camera.
76.         """
77.
78.         bounding_boxes = [ClientSideBoundingBoxes.get_bounding_b
    ox(vehicle, camera) for vehicle in vehicles]
79.         # filter objects behind camera
80.         bounding_boxes = [bb for bb in bounding_boxes if all(bb[:, 2]
     > 0)]
81.         return bounding_boxes
82.
```

```
    @staticmethod
    def draw_bounding_boxes(display, bounding_boxes, color):
        """
        Draws bounding boxes on pygame display.
        """

        bb_surface = pygame.Surface((VIEW_WIDTH, VIEW_HEIGHT))
        bb_surface.set_colorkey((0, 0, 0))
        for bbox in bounding_boxes:
            points = [(int(bbox[i, 0]), int(bbox[i, 1])) for i in range(8)]
            # draw lines
            # base
            pygame.draw.line(bb_surface, color, points[0], points[1])
            pygame.draw.line(bb_surface, color, points[0], points[1])
            pygame.draw.line(bb_surface, color, points[1], points[2])
            pygame.draw.line(bb_surface, color, points[2], points[3])
```

```
99.             pygame.draw.line(bb_surface, color, points[3], points[0])
100.
101.            # top
102.            pygame.draw.line(bb_surface, color, points[4], points
    [5])
103.            pygame.draw.line(bb_surface, color, points[5], points
    [6])
104.            pygame.draw.line(bb_surface, color, points[6], points
    [7])
105.            pygame.draw.line(bb_surface, color, points[7], points
    [4])
106.            # base-top
107.            pygame.draw.line(bb_surface, color, points[0], points
    [4])
108.            pygame.draw.line(bb_surface, color, points[1], points
    [5])
109.            pygame.draw.line(bb_surface, color, points[2], points
    [6])
110.            pygame.draw.line(bb_surface, color, points[3], points
    [7])
111.        display.blit(bb_surface, (0, 0))
```

```
112.        @staticmethod
113.        def get_bounding_box(vehicle, camera):
114.            """
115.            Returns 3D bounding box for a vehicle based on camer
    a view.
116.            """
117.
118.            bb_cords = ClientSideBoundingBoxes._create_bb_points(v
    ehicle)
119.            cords_x_y_z = ClientSideBoundingBoxes._vehicle_to_senso
    r(bb_cords, vehicle, camera)[:3, :]
120.            cords_y_minus_z_x = np.concatenate([cords_x_y_z[1, :],
    -cords_x_y_z[2, :], cords_x_y_z[0, :]])
121.            bbox = np.transpose(np.dot(camera.calibration, cords_y
    _minus_z_x))
122.            camera_bbox = np.concatenate([bbox[:, 0] / bbox[:, 2],
    bbox[:, 1] / bbox[:, 2], bbox[:, 2]], axis=1)
123.            return camera_bbox
124.
125.        @staticmethod
126.        def _create_bb_points(vehicle):
127.            """
```

```
128.        Returns 3D bounding box for a vehicle.
129.        """
130.
131.        cords = np.zeros((8, 4))
132.        extent = vehicle.bounding_box.extent
133.        cords[0, :] = np.array([extent.x, extent.y, -extent.z, 1])
134.        cords[1, :] = np.array([-extent.x, extent.y, -extent.z, 1]
            )
135.        cords[2, :] = np.array([-extent.x, -extent.y, -extent.z,
            1])
136.        cords[3, :] = np.array([extent.x, -extent.y, -extent.z, 1]
            )
137.        cords[4, :] = np.array([extent.x, extent.y, extent.z, 1])
138.        cords[5, :] = np.array([-extent.x, extent.y, extent.z, 1])
139.        cords[6, :] = np.array([-extent.x, -extent.y, extent.z, 1]
            )
140.        cords[7, :] = np.array([extent.x, -extent.y, extent.z, 1])
141.        return cords
142.
```

```
143.     @staticmethod
144.     def _vehicle_to_sensor(cords, vehicle, sensor):
145.         """
146.         Transforms coordinates of a vehicle bounding box to se
     nsor.
147.         """
148.
149.         world_cord = ClientSideBoundingBoxes._vehicle_to_worl
     d(cords, vehicle)
150.         sensor_cord = ClientSideBoundingBoxes._world_to_senso
     r(world_cord, sensor)
151.         return sensor_cord
152.
153.     @staticmethod
154.     def _vehicle_to_world(cords, vehicle):
155.         """
156.         Transforms coordinates of a vehicle bounding box to w
     orld.
157.         """
158.
159.         bb_transform = carla.Transform(vehicle.bounding_box.lo
     cation)
```

```
160.            bb_vehicle_matrix = ClientSideBoundingBoxes.get_matri
    x(bb_transform)
161.            vehicle_world_matrix = ClientSideBoundingBoxes.get_m
    atrix(vehicle.get_transform())
162.            bb_world_matrix = np.dot(vehicle_world_matrix, bb_vehi
    cle_matrix)
163.            world_cords = np.dot(bb_world_matrix, np.transpose(co
    rds))
164.            return world_cords
165.
166.        @staticmethod
167.        def _world_to_sensor(cords, sensor):
168.            """
169.            Transforms world coordinates to sensor.
170.            """
171.
172.            sensor_world_matrix = ClientSideBoundingBoxes.get_ma
    trix(sensor.get_transform())
173.            world_sensor_matrix = np.linalg.inv(sensor_world_matri
    x)
174.            sensor_cords = np.dot(world_sensor_matrix, cords)
175.            return sensor_cords
```

```
176.
177.     @staticmethod
178.     def get_matrix(transform):
179.         """
180.         Creates matrix from carla transform.
181.         """
182.
183.         rotation = transform.rotation
184.         location = transform.location
185.         c_y = np.cos(np.radians(rotation.yaw))
186.         s_y = np.sin(np.radians(rotation.yaw))
187.         c_r = np.cos(np.radians(rotation.roll))
188.         s_r = np.sin(np.radians(rotation.roll))
189.         c_p = np.cos(np.radians(rotation.pitch))
190.         s_p = np.sin(np.radians(rotation.pitch))
191.         matrix = np.matrix(np.identity(4))
192.         matrix[0, 3] = location.x
193.         matrix[1, 3] = location.y
194.         matrix[2, 3] = location.z
195.         matrix[0, 0] = c_p * c_y
196.         matrix[0, 1] = c_y * s_p * s_r - s_y * c_r
197.         matrix[0, 2] = -c_y * s_p * c_r - s_y * s_r
```

```
198.            matrix[1, 0] = s_y * c_p
199.            matrix[1, 1] = s_y * s_p * s_r + c_y * c_r
200.            matrix[1, 2] = -s_y * s_p * c_r + c_y * s_r
201.            matrix[2, 0] = s_p
202.            matrix[2, 1] = -c_p * s_r
203.            matrix[2, 2] = c_p * c_r
204.            return matrix
205.
206.
207.    # ==============================================
       ==============================
208.    # -- BasicSynchronousClient ---------------------
       ---------------------------
209.    # ==============================================
       ==============================
210.    def draw_transform(debug, trans, col=carla.Color(255, 0, 0),
        lt=-1):
211.            debug.draw_arrow(
212.                trans.location, trans.location + trans.get_forward_ve
       ctor(),
213.                thickness=0.05, arrow_size=0.1, color=col, life_time=
       lt)
```

```
214.
215.
216.  def draw_waypoint_union(debug, w0, w1, color=carla.Color(2
    55, 0, 0), lt=5):
217.      debug.draw_line(
218.          w0.transform.location + carla.Location(z=0.25),
219.          w1.transform.location + carla.Location(z=0.25),
220.          thickness=0.1, color=color, life_time=lt, persistent_lines
    =False)
221.      debug.draw_point(w1.transform.location + carla.Location(z
    =0.25), 0.1, color, lt, False)
222.
223.  class BasicSynchronousClient(object):
224.      """
225.      Basic implementation of a synchronous client.
226.      """
227.
228.      def __init__(self):
229.          self.client = None
230.          self.world = None
231.          self.camera = None
232.          self.car = None
```

```
233.
234.            self.display = None
235.            self.image = None
236.            self.capture = True
237.
238.        def camera_blueprint(self):
239.            """
240.            Returns camera blueprint.
241.            """
242.
243.            camera_bp = self.world.get_blueprint_library().find('sens
    or.camera.rgb')
244.            camera_bp.set_attribute('image_size_x', str(VIEW_WIDTH)
    )
245.            camera_bp.set_attribute('image_size_y', str(VIEW_HEIGH
    T))
246.            camera_bp.set_attribute('fov', str(VIEW_FOV))
247.            return camera_bp
248.
249.        def set_synchronous_mode(self, synchronous_mode):
250.            """
251.            Sets synchronous mode.
```

```
252.             """
253.
254.             settings = self.world.get_settings()
255.             settings.synchronous_mode = synchronous_mode
256.             self.world.apply_settings(settings)
257.
258.         def setup_car(self):
259.             """
260.             Spawns actor-vehicle to be controled.
261.             """
262.             vehicle_list = []
263.             car_bp = self.world.get_blueprint_library().filter('vehicle.*
    ')[0]
264.             location = random.choice(self.world.get_map().get_spaw
    n_points())
265.             self.car = self.world.spawn_actor(car_bp, location)
266.             self.car.set_autopilot(True)
267.
268.             spawn_points = self.world.get_map().get_spawn_points()
269.             vehicle_bps = self.world.get_blueprint_library().filter('vehi
    cle.*.*')
```

```
270.            vehicle_bps = [x for x in vehicle_bps if int(x.get_attribute
    ('number_of_wheels')) == 4]
271.
272.            walkers_list = []
273.            picked_spawn_points = []
274.            for i in range(NUMBER_OF_VEHICLES):
275.                point = spawn_points[i]
276.                vehicle_bp = np.random.choice(vehicle_bps)
277.                try:
278.                    vehicle = self.world.spawn_actor(vehicle_bp, point)
279.                    picked_spawn_points.append(point)
280.                    vehicle_list.append(vehicle)
281.                except:
282.                    print('failed')
283.                    pass
284.            tm = self.client.get_trafficmanager()
285.            tm.global_percentage_speed_difference(20.0)
286.            tm_port = tm.get_port()
287.            for v in vehicle_list:
288.                v.set_autopilot(True, tm_port)
289.                tm.ignore_lights_percentage(v, 50)
```

```
            tm.distance_to_leading_vehicle(v, 2)
            tm.ignore_walkers_percentage(v, 80)
            tm.vehicle_percentage_speed_difference(v, -20)

        blueprintsWalkers = self.world.get_blueprint_library().filter("walker.pedestrian.*")
        # 1. Take all the random locations to spawn
        spawn_points = []
        for i in range(50):
            spawn_point = carla.Transform()
            spawn_point.location = self.world.get_random_location_from_navigation()
            if (spawn_point.location != None):
                spawn_points.append(spawn_point)

        # 2. Build the batch of commands to spawn the pedestrians
        batch = []
        for spawn_point in spawn_points:
            walker_bp = random.choice(blueprintsWalkers)
            batch.append(carla.command.SpawnActor(walker_bp, spawn_point))
```

```
308.
309.            # 2.1 apply the batch
310.            results = self.client.apply_batch_sync(batch, True)
311.            for i in range(len(results)):
312.                if results[i].error:
313.                    logging.error(results[i].error)
314.                else:
315.                    walkers_list.append({"id": results[i].actor_id})
316.
317.            # 3. Spawn walker AI controllers for each walker
318.            batch = []
319.            walker_controller_bp = self.world.get_blueprint_library().
    find('controller.ai.walker')
320.            for i in range(len(walkers_list)):
321.                batch.append(carla.command.SpawnActor(walker_c
    ontroller_bp, carla.Transform(), walkers_list[i]["id"]))
322.
323.            # 3.1 apply the batch
324.            results = self.client.apply_batch_sync(batch, True)
325.            for i in range(len(results)):
326.                if results[i].error:
327.                    logging.error(results[i].error)
```

```
328.                 else:
329.                     walkers_list[i]["con"] = results[i].actor_id
330.             all_id = []
331.             # 4. Put altogether the walker and controller ids
332.             for i in range(len(walkers_list)):
333.                 all_id.append(walkers_list[i]["con"])
334.                 all_id.append(walkers_list[i]["id"])
335.             all_actors = self.world.get_actors(all_id)
336.
337.             # wait for a tick to ensure client receives the last trans
    form of the walkers we have just created
338.             self.world.wait_for_tick()
339.
340.             # 5. initialize each controller and set target to walk to
     (list is [controller, actor, controller, actor ...])
341.             for i in range(0, len(all_actors), 2):
342.                 # start walker
343.                 all_actors[i].start()
344.                 # set walk to random point
345.                 all_actors[i].go_to_location(self.world.get_random_loc
    ation_from_navigation())
346.                 # random max speed
```

```
347.                 all_actors[i].set_max_speed(1 + random.random())
348.                 # self.player = self.world.spawn_actor(random.choice
    (ped_blueprints), spawn_point_v2)
349.             # player_control = carla.WalkerControl()
350.             # player_control.speed = 1
351.             # pedestrian_heading = 90
352.             # player_rotation = carla.Rotation(0, pedestrian_headi
    ng, 0)
353.             # player_control.direction = player_rotation.get_forwar
    d_vector()
354.             # self.player.apply_control(player_control)
355.             # transform = spawn_point_v2
356.             # spectator.set_transform(carla.Transform(transform.lo
    cation))
357.         def setup_camera(self):
358.             """
359.             Spawns actor-camera to be used to render view.
360.             Sets calibration for client-side boxes rendering.
361.             """
362.
363.             camera_transform = carla.Transform(carla.Location(x=
    -5.5, z=2.8), carla.Rotation(pitch=-15))
```

```
364.            self.camera = self.world.spawn_actor(self.camera_bluepri
    nt(), camera_transform, attach_to=self.car)
365.            weak_self = weakref.ref(self)
366.            self.camera.listen(lambda image: weak_self().set_image(
    weak_self, image))
367.
368.            calibration = np.identity(3)
369.            calibration[0, 2] = VIEW_WIDTH / 2.0
370.            calibration[1, 2] = VIEW_HEIGHT / 2.0
371.            calibration[0, 0] = calibration[1, 1] = VIEW_WIDTH / (
    2.0 * np.tan(VIEW_FOV * np.pi / 360.0))
372.            self.camera.calibration = calibration
373.
374.        def control(self, car):
375.            """
376.            Applies control to main car based on pygame pressed k
    eys.
377.            Will return True If ESCAPE is hit, otherwise False to en
    d main loop.
378.            """
379.            keys = pygame.key.get_pressed()
380.            if keys[K_ESCAPE]:
```

```
381.             return True
382.
383.         control = car.get_control()
384.         control.throttle = 0
385.         if keys[K_w]:
386.             control.throttle = 1
387.             control.reverse = False
388.         elif keys[K_s]:
389.             control.throttle = 1
390.             control.reverse = True
391.         if keys[K_a]:
392.             control.steer = max(-1., min(control.steer - 0.05, 0)
    )
393.         elif keys[K_d]:
394.             control.steer = min(1., max(control.steer + 0.05, 0))
395.         else:
396.             control.steer = 0
397.         control.hand_brake = keys[K_SPACE]
398.
399.         car.apply_control(control)
400.         return False
```

```
401.
402.        @staticmethod
403.        def set_image(weak_self, img):
404.            """
405.            Sets image coming from camera sensor.
406.            The self.capture flag is a mean of synchronization - onc
    e the flag is
407.            set, next coming image will be stored.
408.            """
409.
410.            self = weak_self()
411.            if self.capture:
412.                self.image = img
413.                self.capture = False
414.
415.        def render(self, display):
416.            """
417.            Transforms image from camera sensor and blits it to
    main pygame display.
418.            """
419.
420.            if self.image is not None:
```

```
421.                 array = np.frombuffer(self.image.raw_data, dtype=n
    p.dtype("uint8"))
422.                 array = np.reshape(array, (self.image.height, self.ima
    ge.width, 4))
423.                 array = array[:, :, :3]
424.                 array = array[:, :, ::-1]
425.                 surface = pygame.surfarray.make_surface(array.swa
    paxes(0, 1))
426.                 display.blit(surface, (0, 0))
427.
428.
429.
430.         def game_loop(self):
431.             """
432.             Main program loop.
433.             """
434.
435.             try:
436.                 pygame.init()
437.
438.                 self.client = carla.Client('127.0.0.1', 2000)
439.                 self.client.set_timeout(5.0)
```

```
440.                self.world = self.client.load_world('Town02')
441.                self.setup_car()
442.                self.setup_camera()
443.                # abc = carla.TrafficLight()
444.                # abc.set_red_time(3.0)
445.                self.display = pygame.display.set_mode((VIEW_WIDT
    H, VIEW_HEIGHT), pygame.HWSURFACE | pygame.DOUBLEBUF)

446.                pygame_clock = pygame.time.Clock()
447.                map = self.world.get_map()
448.                self.set_synchronous_mode(True)
449.                vehicles = self.world.get_actors().filter('vehicle.*')
450.                walkers = self.world.get_actors().filter("walker.*")
451.                # vehicles.append(walkers)
452.                current_w = map.get_waypoint(self.car.get_location())

453.                debug = self.world.debug
454.                while True:
455.                    self.world.tick()
456.                    self.capture = True
457.                    pygame_clock.tick_busy_loop(20)
458.
```

```
459.
460.                     next_w = map.get_waypoint(self.car.get_location(),
    lane_type=carla.LaneType.Driving | carla.LaneType.Shoulder | carl
    a.LaneType.Sidewalk )
461.                     # Check if the vehicle is moving
462.                     if next_w.id != current_w.id:
463.                         vector = self.car.get_velocity()
464.                         # Check if the vehicle is on a sidewalk
465.                         if current_w.lane_type == carla.LaneType.Side
    walk:
466.                             draw_waypoint_union(debug, current_w, n
    ext_w, cyan if current_w.is_junction else red, 60)
467.                         else:
468.                             draw_waypoint_union(debug, current_w, n
    ext_w, cyan if current_w.is_junction else green, 60)
469.                         debug.draw_string(self.car.get_location(), str('%
    15.0f km/h' % (3.6 * math.sqrt(vector.x**2 + vector.y**2 + vecto
    r.z**2))), False, orange, 60)
470.                         #debug.draw_string(current_w.transform.locat
    ion, str('%15.0f km/h' % (3.6 * math.sqrt(vector.x**2 + vector.y**
    2 + vector.z**2))), False, orange, 60)
```

```
                draw_transform(debug, current_w.transform, white, 60)
            # Update the current waypoint and sleep for some time
            current_w = next_w

            if self.car.is_at_traffic_light():
                traffic_light = self.car.get_traffic_light()
                if traffic_light.get_state() == carla.TrafficLightState.Red:
                # world.hud.notification("Traffic light changed! Good to go!")
                    traffic_light.set_state(carla.TrafficLightState.Green)
            self.render(self.display)
            bounding_boxes = ClientSideBoundingBoxes.get_bounding_boxes(vehicles, self.camera)
            bounding_boxes1 = ClientSideBoundingBoxes.get_bounding_boxes(walkers, self.camera)
            ClientSideBoundingBoxes.draw_bounding_boxes(self.display, bounding_boxes, BB_COLOR)
```

```
484.                     ClientSideBoundingBoxes.draw_bounding_boxes(se
    lf.display, bounding_boxes1, BB_COLOR1)
485.                     pygame.display.flip()
486.
487.                     pygame.event.pump()
488.                     if self.control(self.car):
489.                         return
490.
491.             finally:
492.                 self.set_synchronous_mode(False)
493.                 self.camera.destroy()
494.                 self.car.destroy()
495.                 self.car1.destroy()
496.                 self.player.destroy()
497.                 pygame.quit()
498.     # ==============================================
    ==============================
499.     # -- main() ---------------------------------------
    -----------------------------
500.     # ==============================================
    ==============================
501.     def main():
```

```
    """
    Initializes the client-side bounding box demo.
    """

    try:
        client = BasicSynchronousClient()
        client.game_loop()
    finally:
        print('EXIT')


if __name__ == '__main__':
    main()
```

附录 D　车辆横纵向控制程序

```
1. import numpy as np
2. import math
3. import glob
4. import os
5. import sys
6. import cv2
7. try:
8.     sys.path.append(glob.glob('../carla/dist/carla-*%d.%d-%s.egg' %
   (
9.         sys.version_info.major,
10.         sys.version_info.minor,
11.         'win-amd64' if os.name == 'nt' else 'linux-x86_64'))[0])
12. except IndexError:
13.     pass
14.
15. import carla
16. from carla import Transform, Location, Rotation
17. from carla import Map
```

```
18. from carla import Vector3D
19. import random
20. import time
21. actor_list = []
22.
23.
24. class Controller2D(object):
25.     def __init__(self):
26.         self._conv_rad_to_steer = 2 / np.pi
27.         self.alpha_previous          = 0
28.         self._wheelbase         = 2.5
29.         self.steering_previous      = 0
30.         self._steering_diff = np.linspace(-10, 10, 21, endpoint=Tru
    e)
31.         self._Kmpc = 0.3
32.     def set_steer(self, input_steer_in_rad):
33.         # Covnert radians to [-1, 1]
34.         input_steer = self._conv_rad_to_steer * input_steer_in_rad
35.         # Clamp the steering command to valid bounds
36.         steer           = np.fmax(np.fmin(input_steer, 1.0), -1.0)
37.         return steer
38.
```

```
39.     def get_alpha(self, v1, v2, ld):
40.         inner_prod = abs(v1[0] * v2[0] + v1[1] * v2[1])
41.         return np.arccos(inner_prod / ld)
42.
43.     def get_lookahead_dis(self, x, y, waypointx, waypointy):
44.         a = math.pow((x - waypointx), 2) + math.pow((y - waypo
    inty), 2)
45.         return math.sqrt(a)
46.
47.     def get_steering_direction(self, yaw, yaw1):
48.         if yaw > yaw1:
49.             return -1
50.         return 1
51.
52.     def get_heading_error(self, current_yaw, v1):
53.         waypoint_delta_x = v1[0]
54.         waypoint_delta_y = v1[1]
55.         waypoint_heading = np.arctan(waypoint_delta_y / waypoin
    t_delta_x)
56.         heading_error_mod = divmod((waypoint_heading - current
    _yaw), np.pi)[1]
```

```
57.         if heading_error_mod > np.pi / 2 and heading_error_mod <
    np.pi:
58.             heading_error_mod -= np.pi
59.         return heading_error_mod
60.
61.     def get_predicted_wheel_location(self, x, y, steering_angle, yaw,
    v):
62.         wheel_heading = yaw + steering_angle
63.         wheel_traveled_dis = v * (self._current_timestamp - self.var
    s.t_previous)
64.         return [x + wheel_traveled_dis * np.cos(wheel_heading), y +
    wheel_traveled_dis * np.sin(wheel_heading)]
65.
66.     def get_distance(self, x1, y1, x2, y2):
67.         return np.sqrt((x1 - x2)**2 + (y1 - y2)**2)
68.
69.     def get_cte_heading_error(self, v):
70.         proportional_cte_error = self._Kcte * self._closest_distance
71.         cte_heading_error = np.arctan(proportional_cte_error/v)
72.         cte_heading_error_mod = divmod(cte_heading_error, np.pi)[
    1]
```

```
73.         if cte_heading_error_mod > np.pi/2 and cte_heading_error_
    mod < np.pi:
74.             cte_heading_error_mod -= np.pi
75.         return cte_heading_error_mod
76.
77.     def calculate_steering(self, x, y, yaw, waypoint, vf, m):
78.         waypointx = waypoint.transform.location.x
79.         waypoint_y = waypoint.transform.location.y
80.         if m == 1:
81.             yaw1 = waypoint.transform.rotation.yaw
82.             ld = self.get_lookahead_dis(x, y, waypointx, waypoint_y)


83.             v1 = [waypointx - x, waypoint_y - y]
84.             v2 = [np.cos(yaw), np.sin(yaw)]
85.             alpha = self.get_alpha(v1, v2, ld)
86.             lf = 0.1 * vf + 2.0
87.             if math.isnan(alpha):
88.                 alpha = self.alpha_previous
89.             if not math.isnan(alpha):
90.                 self.alpha_previous = alpha
91.             steering = self.get_steering_direction(yaw, yaw1) * np.ar
    ctan(
```

```
92.                 (2 * self._wheelbase * np.sin(alpha)) / lf)
93.             if math.isnan(steering):
94.                 steering = self.steering_previous
95.             if not math.isnan(steering):
96.                 self.steering_previous = steering
97.             return self.set_steer(steering)
98.         elif m == 2:
99.             v1 = [waypointx - x, waypoint_y - y]
100.                yaw1 = waypoint.transform.rotation.yaw
101.                heading_error = self.get_heading_error(yaw1, v1)
102.                cte_error = self.get_steering_direction(yaw, yaw1)*se
    lf.get_cte_heading_error(vf)
103.                steering = heading_error + cte_error
104.                return steering
105.            elif m == 3:
106.                steering_list = self.vars.steering_previous + self._steeri
    ng_diff * self._pi/180
107.                min_dis = float("inf")
108.                steering = self.vars.steering_previous
109.                for i in range(len(steering_list)):
110.                    predicted_wheel_location = self.get_predicted_whe
    el_location(x, y, steering_list[i], yaw, vf)
```

```
                    dis_to_last_waypoint = self.get_distance(predicted_wheel_location[0], predicted_wheel_location[1], waypointx, waypoint_y)
                    if dis_to_last_waypoint < min_dis:
                        min_dis = dis_to_last_waypoint
                        steering = steering_list[i]
            return steering

    def main():
        client = carla.Client('localhost', 2000)
        client.set_timeout(10.0)
        world = client.load_world('Town02')
        map1 = world.get_map()
        blueprint_library = world.get_blueprint_library()
        vehicle_bp1 = blueprint_library.filter('tt')[0]
        vehicle_bp1.set_attribute('color', '255,255,255')
        spawn_point_v1 = Transform(Location(x=-3.7, y=300, z=1),
                                   Rotation(pitch=0, yaw=-120, roll=0))

        vehicle1 = world.spawn_actor(vehicle_bp1, spawn_point_v1)
```

```
        actor_list.append(vehicle1)
        controller = Controller2D()
        try:
            while True:
                x_v1 = vehicle1.get_location().x
                y_v1 = vehicle1.get_location().y
                z = vehicle1.get_transform()
                z1 = z.rotation.yaw
                x_v1 = x_v1 - 1.5*np.cos(z1)
                y_v1 = y_v1 - 1.5*np.sin(z1)
                z2 = vehicle1.get_velocity()
                vf0 = math.pow(z2.x, 2) + math.pow(z2.y, 2)
                vf = math.sqrt(vf0)
                waypoint = map1.get_waypoint(vehicle1.get_location
    (), project_to_road=True,
                                             lane_type=carla.LaneType.Driv
    ing)
                c = controller.calculate_steering(x_v1, y_v1, z1, way
    point, vf, 2)
                vehicle1.apply_control(carla.VehicleControl(throttle=
    0.5, steer=c))
        finally:
```

```
146.             for actor in actor_list:
147.                 actor.destroy()
148.             print("All cleaned up!")
149.
150.     if __name__ == '__main__':
151.
152.         try:
153.             main()
154.         except KeyboardInterrupt:
155.             print('\nCancelled by user. Bye!')
```