工 程 实 践 结 题 报 告

题目 自动驾驶安全测试

小组成员 吴志强 SA21225465

王茂安 SA21225427

黄雨轩 SA21225223

吴雷 SA21225455

指导教师 郭燕

所在学院(系) 软件学院

自动驾驶安全测试

吴志强、王茂安、黄雨轩、吴雷

中国科学技术大学软件学院　230026

摘　要

随着汽车智能化的不断提高，高级别的自动驾驶软件逐渐应用于多种安全攸关的场合，智能网联汽车安全问题日益严峻，自动驾驶面临着在传统安全和AI等多方面的安全攻击问题。本课题的目的是首先通过收集和整理自动驾驶面临的安全风险类别，并调研各种用于自动驾驶的模拟软件以便于后续进行自动驾驶测试。了解各种地图仿真软件以便得到想要的测试场景，通过Carla和自定义地图得到了车辆发生碰撞的错误场景。

1. 自动驾驶的安全问题调研。
2. 自动驾驶仿真模拟器的调研。
3. OpenPilot 的源码阅读。
4. 仿真地图的生成。
5. Carla中模拟自动驾驶发生碰撞。

关键词：自动驾驶；仿真模拟；安全测试；场景构建

Automatic driving safety test

Wu Zhiqiang, Wang Maoan, Huang Yuxuan, Wu Lei

School of software, University of science and technology of China 230026

Abstract

With the continuous improvement of vehicle intelligence, high-level automatic driving software is gradually applied to a variety of safety critical occasions. The safety problem of intelligent Internet connected vehicles is becoming increasingly serious. Automatic driving is facing security attacks in many aspects, such as traditional security and AI. The purpose of this topic is to collect and sort out the safety risk categories faced by automatic driving, and investigate various simulation software for automatic driving, so as to facilitate the subsequent automatic driving test. Understand various map simulation software to get the desired test scenario, and get the wrong scenario of vehicle collision through Carla and user-defined map.

1. Research on the safety of autonomous driving.
2. Research on autopilot simulation simulator.
3. Read the source code of openpilot.
4. Generation of simulation map.
5. Carla simulates the collision of automatic driving.

Key words: automatic driving; analogue simulation; Safety test; Scene construction

目录

一、自动驾驶安全问题调研

1.1 外部攻击引起安全问题

1.1.1 云端威胁

智能网联汽车的云平台面临着各种恶意威胁，除了需要病毒防护、中间件安全防护以及访问控制防护外，还需要重视数据安全防护问题，防止车主存储到云端的数据，特别是隐私数据意外丢失或者被别人窃取非法利用。目前云平台数据安全问题主要面临以下三大挑战：

- 数据的隐私性：通过智能终端及设备采集并上传到云平台的数据，涉及车主车辆相关隐私数据，如何保证云平台存储的用户隐私信息不被泄露？
- 数据的完整性：如何保证存储在云端的用户数据完整性不被破坏？
- 数据的可恢复性：用户对云平台的数据进行访问时，如何无差错响应用户的请求，如果遇到安全攻击事件，服务商如何保证出错数据的可恢复性？

1.1.2 传输威胁

“车-X”(人、车、路、互联网等) 通过 WiFi、移动通信网、DSRC 等无线通信手段与其他车辆、交通专网、互联网等进行连接。这些无线通信方式本身就存在网络加密、认证等方面的安全问题。网络传输安全威胁指车联网终端与网络中心的双向数据传输安全威胁，主要存在以下三大安全风险：

- 认证风险：没有验证发送者的身份信息、伪造身份、动态劫持等
- 传输风险：车辆信息没有加密或强度不够、密钥信息暴露、所有车型使用相同的对称密钥
- 协议风险：通信流程伪装，把一种协议伪装成另一种协议

在自动驾驶情况下，汽车会按照 V2X 通信内容判断行驶路线，攻击者可以利用伪消息诱导车辆发生误判，影响车辆自动控制。

1.1.3 终端威胁

1.1.3.1 T-BOX 安全威胁

T-BOX 实现车内网和车际网之间的通信，负责将数据发送到云服务器。从某种程度上来说，T-BOX 的网络安全系数决定了汽车行驶和整个智能交通网络的安全。常规条件下，汽车消息指令在 T-BOX 内部生成，在传输层面对指令进行加密处理，无法直接看到具体信息内容。但恶意攻击者通过分析固件内部代码能够轻易获取加密方法和密钥，实现对消息会话内容的破解。

1.1.3.2 IVI 安全威胁

车载信息娱乐系统。IVI 能够实现包括三维导航、实时路况、辅助驾驶、故障检测、车辆信息，车身控制、无线通讯、基于在线的娱乐功能及 TSP 服务等一系列应用。但 IVI 的高集成度使其所有接口都有可能成为黑客的攻击节点，IVI 的被攻击面将比其他任何车辆部件都多。攻击者既可以借助软件升级的特殊时期，获得访问权限进入目标系统，也可以将 IVI 从目标车上拆下来，分解 IVI 单元连接，通过对电路、接口进行逆向分析，获得内部源代码。

1.1.3.3 终端升级安全威胁

智能网联车由于潜在安全漏洞，需要及时更新。需要通过 OTA 升级的方式来增强自身安全防护能力。但 OTA 升级过程中也面临着各种威胁风险，包括：

- 升级过程中，篡改升级包控制系统，或者升级包被分析发现安全漏洞
- 传输过程中，升级包被劫持，实施中间人攻击
- 生成过程中，云端服务器被攻击，OTA 成为恶意软件源头，另外 OTA 升级包还存在被提权控制系统、ROOT 设备等隐患

1.1.3.4 车载 OS 安全威胁

车载电脑系统通常采用嵌入式 Linux、QNX、Android 等作为操作系统，由于操作系统代码庞大，且存在不同程度的安全漏洞，操作系统自身的安全脆弱性将直接导致业务应用系统的安全智能终端面临被恶意入侵控制的风险。此外，一些通用的应用程序自身的安全漏洞及由于配置不当所造成的安全隐患，都会导致车载网络整体安全性下降。

1.1.3.5 接入风险：基于车载诊断系统接口(OBD)的攻击

OBD 接口是汽车 ECU(电子控制单元)与外部进行交互的唯一接口。它能读取很多汽车敏感信息。作为总线上的一个节点，不仅能监听总线上面的消息，而且还能伪造消息来欺骗 ECU，从而达到改变汽车行为状态的目的。通过在汽车 OBD 接口植入具有无线收发功能的恶意硬件，攻击者可远程向该硬件发送恶意 ECU 控制指令，后果不堪设想。

1.1.3.6 车内无线传感器安全威胁

传感器存在通讯信息被窃听、被中断、被注入等潜在威胁，甚至通过干扰传感器通信设备还会造成无人驾驶汽车偏行、紧急停车等危险动作。

1.1.3.7 车内网络传输安全威胁

车载网络通信协议安全防护措施较弱。如果黑客攻入了车内网络，则可以任意控制 ECU，或者通过发送大量错误报文，导致 CAN 总线失效，进而导致 ECU 失效。

1.1.4 移动 APP 安全威胁

市场上大多数智能网联汽车远程控制 APP 安全强度不够，甚至很多连最基础的软件防护和安全保障都不具备。只需对那些 APP 进行逆向分析挖掘，便可以发现 APP 内的核心内容，包括存放在 APP 中的密钥，重要控制接口等。

1.2 自动驾驶技术本身安全问题

1.2.1 感知系统的安全问题

- 激光雷达会受天气（雨天，雾天）影响，空气中的悬浮物对光传播产生影响，从而影响感知结果精度
- 毫米波雷达无法检测上过漆的塑料或者木头，对金属表面敏感，一个弯曲的金属表面会被认为是一个很大的金属面，因此路上的小易拉罐可能会被认为是很大的障碍，在大桥和隧道表现不佳
- 超声波雷达使用机械波，对温度敏感，超声波散射角大，方向性较差，无法精确描述障碍物位置

1.2.2 感知融合算法的安全问题

由于各个传感器都有局限性且实现的功能不同，难以互相替代，需要多个传感器之间取长补短，来提升定位的正确率和精确度，于是诞生了多传感器融合。但由于环境的复杂多变，传感器可得到的数据十分多样，某些情况下当前使用的感知融合方案可能会出错。

1.2.3 决策模块的安全问题

在一套完整的自动驾驶系统中，如果将感知模块比作人的眼睛和耳朵，那么决策规划就是自动驾驶的大脑。大脑在接收到传感器的各种感知信息之后，对当前环境作出分析，然后对底层控制模块下达指令，这一过程就是决策规划模块的主要任务。 在应对环境多变性、检测不准确性、交通复杂性以及交规约束性等诸多车辆行驶不利因素时，决策模块未必能作出正确的决策。综合车辆运行环境及车辆信息，结合行驶目的做出具有安全性、可靠性以及合理性的驾驶行为是决策的难点，亦是实现自动驾驶的难点。

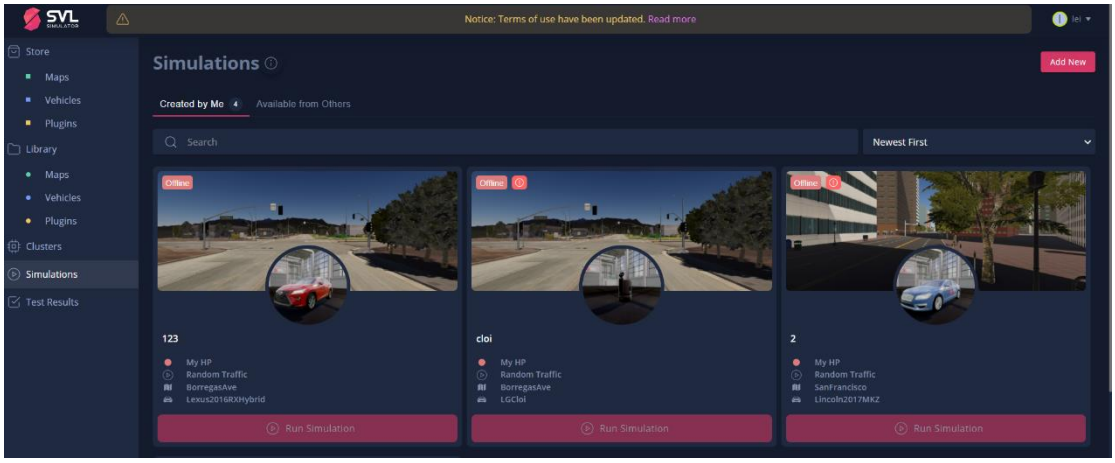
二、自动驾驶仿真器调研

2.1 LGSVL

开源软件，免费使用。 LGSVL 是基于游戏引擎 Unity 开发的一款主要用于自动驾驶开发和测试，支持包括仿真环境、传感器以及通讯内容的自定义。 LGSVL 由模拟软件、软件工具、支持定制用例的内容和插件生态系统以及支持大规模模拟和场景测试的云环境组成，允许开发人员进行调试、执行模块化测试和执行集成测试。

2.1.1 界面

SVL Web 界面



SVL 软件模拟界面

2.1.2 LGSVL 的关键功能：

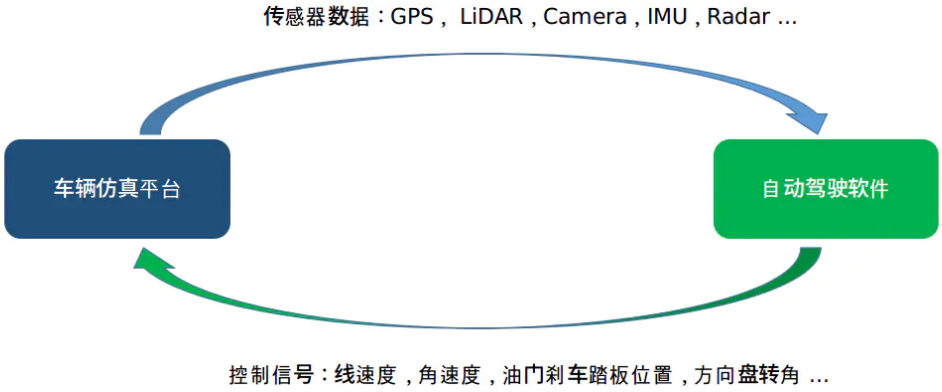
- 端到端模拟
- 实时高性能模拟
- 逼真的环境模拟
- 多车模拟
- 可拓展性和可定制性
- 程序化道路网络生成和环境创建工具
- 高清地图导入、导出和注释工具
- 具有丰富的模型库：
- 地图
- 车辆（轿车、赛车、机器人）
- 插件（激光雷达、GPS、摄像机等）

2.1.3 支持与其他自动驾驶软件进行闭环仿真测试：

- Apollo
- Autoware

这里所谓的“闭环”，就是测试平台中的仿真车辆将传感器检测数据发送给自动驾驶软件，自动驾驶软件经过一系列感知、定位、决策、控制环节，将最终的控制

信号传递回仿真车辆，驱动汽车按照期望的路线行驶。来自车辆传感器的检测数据和来自自动驾驶软件的控制信号构成了一个闭合的信息流。闭环测试框架如下图所示。



2.1.4 支持调用 Python API：

可以通过 Python API 来操纵场景中的环境布置、车辆移动、传感器配置、天气和时间状态等。

2.1.5 系统配置要求：

单独运行 LGSVL：

- 4GHz 四核 CPU
- Nvidia GTX 1080 (8GB memory)
- Windows 10 64 bit
- 也可在 Linux 系统运行，效率可能稍低

与 Apollo 或者 Autoware 在一台主机上运行：

- 至少 10GB 以上的 GPU

Apollo 或者 Autoware 在不同系统上运行：

- 两个系统分别安装 LGSVL 和自动驾驶软件
- 两个系统网络连接状态良好

2.2 Carla

2.2.1 介绍

CARLA 是一个开源的自动驾驶模拟器。它是从零开始构建的，作为一个模块化和灵活的 API 来解决一系列涉及到自动驾驶问题的任务。

Carla 模拟器由可扩展的客户端-服务器架构组成。服务器负责与模拟本身相关的所有事情：传感器渲染、物理计算、世界状态及其参与者的更新等等。客户端由一组客户端模块组成，这些模块控制场景中行为者的逻辑并设置世界条件。这是通过利用 CARLA API（在 Python 或 C++ 中）来实现的。

2.2.2 核心模块

- **Traffic manager：** Carla 专门构造了 Traffic Manager 这个模块来模拟类似现实世界负责的交通环境。通过这个模块，用户可以定义不同车型、不同行为模式、不同速度的车辆在路上。
- **Sensors：** Carla 里面有各种各样模拟真实世界的传感器模型，包括相机、激光雷达、声波雷达、IMU、GNSS 等等。为了让仿真更接近真实世界，它里面的相机拍出的照片甚至还有畸变和动态模糊效果。用户一般将这些 Sensor attach 到不同的车辆上来收集各种数据。
- **Recorder：** 用来记录仿真每一个时刻的状态，可以用来回顾、复现等等。
- **ROS bridge and Autoware implementation：**这个模块可以让 Carla 与 ROS 还有 Autoware 交互。
- **Open assets：** CARLA 为城市环境提供了不同的地图，可以控制天气条件，并提供一个包含大量要使用的参与者的蓝图库。这些元素可以自定义，并且可以按照简单的指南生成新元素。
- **Scenario runner：** 提供参考实例可以让你更加方便的搭建一个需要的测试场景。

2.2.3 核心概念

2.2.3.1 World and client

客户端和世界是 CARLA 的两个基本要素。

客户端是 CARLA 架构中的主要元素之一。它们连接到服务器、检索信息和命令更改。这是通过脚本完成的。客户端对象的主要目的是获取或改变世界，并应用命令。

世界主要有如下几个部分 Actors in the simulation and the spectator Blueprint library Map Simulation settings Snapshots Weather and light manager ，可以通过编程来生成行为者，自定义天气状况，改变地图上的灯光以及访问一些用于模拟的高级配置等。

2.2.3.2 Actors and blueprints

行为者不仅包括车辆和步行者，还包括传感器、交通标志、交通信号灯和观众。行为者是在模拟中扮演角色的任何东西。蓝图允许用户将新行为者顺利地合并到模拟中。它们是带有动画和一系列属性的已经制作好的模型。其中一些是可以修改的，而另一些则不是。这些属性包括车辆颜色、激光雷达传感器中的通道数量、步行者的速度等等。

官方文档中有简单介绍对应传感器、观众、交通标志和信号灯、车辆以及行人的生成和处理。

2.2.3.3 Maps and navigation

地图主要是代表模拟世界的物体，城镇。有八张地图。地图包括城镇的 3D 模型及其道路定义。地图的道路定义基于 OpenDRIVE 文件，这是一种标准化的、带注释的道路定义格式。OpenDRIVE 标准 1.4 规定了道路、车道、路口等。

同样，官方文档中也提供了简单的实例。针对地图的修改，车道、路口、环境对象的定义等。

2.2.3.4 Sensors and data

传感器等待某个事件发生，然后从模拟中收集数据。它们需要一个定义如何管理

数据的函数。根据具体情况，传感器检索不同类型的传感器数据。

官方文档给出了参考资料。其中 Python API 定义了许多对于世界中的元素改变、控制的方法。包括行为者的控制、道路控制、灯光管理等。官方文档给出简单的例子关于设置传感器的属性，详细的属性设置可以在 sensors reference 中查看。

2.2.3.5 自定义地图

官方文档中有一部分是关于地图的自定义。其中有提到对于交通灯和交通标志的自定义，建筑的自定义，天气的自定义以及道路的自定义。Carla 允许使用不同的材料、纹理以及添加网格和贴纸以获得你想要的外观。

参考：https://carla.readthedocs.io/en/0.9.12/

2.3 PreScan

2.3.1 介绍

PreScan 是西门子公司旗下汽车驾驶仿真软件产品，PreScan 是以物理模型为基础，开发 ADAS 和智能汽车系统的仿真平台。支持摄像头、雷达、激光雷达、GPS，以及 V2V/V2I 车车通讯等多种应用功能的开发应用。 PreScan 基于 MATLAB 仿真平台，主要用于（ADAS）汽车高级驾驶辅助系统和无人自动驾驶系统的仿真模拟软件，其包括多种基于雷达，摄像头，激光雷达，GPS，V2V 和 V2I 车辆/车路通讯技术的智能驾驶应用。

2.3.2 功能

调研过程中发现，网上大部分资料都是针对 Prescan 的场景搭建。静态的，PreScan 可以对环境、天气、路段、路面、路标、交通信号以及一些建筑进行设置。动态的，就是车辆行人的路线和速度等。PreScan 还提供了很多传感器，用于探测周围环境数据。

通常，PreScan 会结合 Matlab/Simulink、Carsim 一起使用。其中用 Matlab/Simulink

做控制算法的开发,用 Carsim 仿真软件提供汽车动力学模型、轮胎模型和制动器模型,利用 Prescan 软件建立测试场景与传感器模型，其中 PreScan 软件 MATLAB/Simulink 软件可以相互调用，具体来说就是 PreScan 中的各种传感器仿真数据传递到 Simulink 中。

参考：Prescan 中文培训文档 PreScan 官网资料 PreScan 仿真从入门到精通

2.4 AirSim

2.4.1 介绍

AirSim 是微软研究院开源的一个建立在虚幻引擎（Unreal Engine）上的无人机以及自动驾驶模拟研究项目。AirSim 实现为一个虚幻引擎的插件，它充分利用了虚幻引擎在打造高还原的逼真虚拟环境的能力，可以模拟阴影、反射等现实世界中的环境，以及虚拟环境可以方便产生大量标注数据的能力，同时提供了简单方便的接口，可以让无人机和自动驾驶的算法接入进行大量的训练。AirSim 的主要目标是作为 AI 研究的平台，以测试深度学习、计算机视觉和自主车辆的端到端的强化学习算法。最新的 AirSim 也提供了 Unity 引擎的版本，添加了激光雷达的支持。

2.4.2 主要功能

- **模拟完成车辆建模测试：**AirSim 包含车辆模拟、城市道路场景，还提供可以简化编程的 API 以及即插即用的代码，可以很快上手并用于自己的自动驾驶项目。
- **快速构建丰富场景：**AirSim 提供了详细的 3D 城市街景，以及包括交通信号灯、公园、湖泊、工地等丰富的场景。开发者可以在各种不同的场景下测试他们的系统，无论是在市中心，还是在城乡道路、郊野和工业区。开发者还可以利用 AirSim 的拓展性添加新的传感器、车辆，甚至使用不同的物理引擎。
- **一站式 AI 研究平台：**AirSim 提供包括 C++和 Python 等多语言的 API 接

口，使用者可以十分容易地将 AirSim 和众多机器学习工具共同使用。例如，开发者可以使用微软认知工具包(CNTK)和 AirSim 进行深度增强学习。同时，我们也看到使用新型的对数据量需求很大的机器学习算法进行训练时，在基于 Microsoft Azure 的 AirSim 下运行多个实例具有很大的潜力。

2.4.3 核心 API

- **图像类 API**：获取各种类型的图像、控制云台等
- **控制仿真运行**：可以控制仿真暂停或继续
- **碰撞 API**：获取碰撞信息，包括碰撞次数、位置、表面信息、渗透深度等
- **环境天气 API**：控制环境中的天气：下雨、雪、灰尘、雾气、道路湿滑程度等
- **环境风 API**：控制环境中的风速和风向等
- **雷达 API**：添加和控制环境中的雷达传感器
- **汽车 API**：控制汽车的行动及各种运动学参数

2.5 PanoSim

2.5.1 介绍

PanoSim 是一款面向汽车自动驾驶技术与产品研发的一体化仿真与测试平台，集高精度车辆动力学模型、高逼真汽车行驶环境与交通模型、车载环境传感器模型和丰富的测试场景于一体，支持与 Matlab/Simulink 联合无缝仿真，提供包括离线仿真、实时硬件在环仿真（MIL/SIL/HIL/VIL)和驾驶模拟器等在内的一体化解决方案；支持包括 ADAS 和自动驾驶环境感知、决策规划与控制执行等在内的算法研发与测试。PanoSim 具有很强的开放性与拓展性，支持定制化开发，操作简便友好。 PanoSim 软件中的驾驶环境为仿真系统提供场景要素，其包括三部分：静态场景（道路、建筑物、信号灯、障碍物等）、动态交通（随机车、干扰车、干扰行人）、天气环境（气象、光照）,提供车辆行驶所必须的场所。

2.5.2 主要模块

- **场景编辑器（FieldBuilder）**：主要用于创建或编辑仿真实验所需三维虚拟场景和环境（包括道路和道路网络结构、道路路面和车道信息、地形、周边建筑和交通设施等）还支持对车道线、道路路面纹理、附着属性、周边环境、地形高度等信息的灵活定制和同步预览，兼容多种格式复杂的路面特征信息，自动生成所见即所得的逼真三维虚拟实验场景。
- **车辆编辑器（VehicleBuilder）**：主要用于创建或编辑仿真实验所用车辆三维外形、车辆动力学和汽车结构参数。
- **实验设置运行（PanoSim MainGUI/Sensor Setup/Field Setup/SensorBuilder）**：承担 PanoSim 核心操作枢纽角色，涵盖实验分组管理、选择实验场景、设置环境条件（例如天气等）、摆放实验车辆、安装车载传感（例如像机、雷达等）、部署交通元素（例如路障、交通标志等）、设置交通模型（例如干扰交通、随机交通等）、设置实验参数、设置驾驶参数、实验运行跟踪（与 Matlab/Simulink 无缝集成）等多项子模块。
- **实验数据分析（PanoPlot/PanoAnim/DataManager）**：主要用于对实验过程数据进行后处理和分析报告（例如多维度图表数据分析、实验动画的回放等）

2.5.3 仿真关键步骤

- **场景构建（Scenario Building）**：通过场景编辑工具，完成测试所需要的场景。测试场景的多样性、覆盖性、典型性等都能够影响到测试结果的准确性，从而影响自动驾驶的安全与质量
- **传感器系统建模（Adding model sensor systems）**：PanoSim 提供多种传感器模型，可以灵活设置传感器参数等
- **添加控制系统（Adding control systems）**：根据车辆状态及传感器信号可在 MATLAB/Simulink 中建立自定义反馈控制算法设计
- **运行仿真实验**

2.6 CarMaker

2.6.1 CarMaker 介绍

CarMaker 是德国 IPG 公司旗下的一款专注于乘用车的动力学仿真软件，它包括了精准的车辆本体模型(发动机、底盘、悬架、传动、转向等)，同时也可以提供包括车辆，驾驶员，道路，交通环境的闭环仿真系统。CarMaker 既可针对 V 开发流程前期进行的模型在环、软件在环等离线仿真，也可以接入 ECU、子系统总成、网络等做硬件在环测试。 CarMaker 和同为 IPG 公司开发的分别针对多轴车辆和摩托车的动力学仿真软件 TruckMaker 和 MotorcycleMaker 一样，包含了道路三维模型，交通环境模型以及适应不同驾驶环境以及不同驾驶策略的驾驶员模型。

- **IPG Road**：可以模拟多车道、 十字路口等多种形式的道路， 并可通过配置 GUI 生成锥形、 圆柱形等形式的路障。可对道路的几何形状以及路面状况(不平度、 粗糙度) 进行任意定义。
- **IPG Traffic**：是交通环境模拟工具， 提供丰富的交通对象（车辆、行人、路标、交通灯、道路施工建筑等）模型。可实现对真实交通环境的仿真。测试车辆可识别交通对象并由此进行动作触发（如限速标志可触发车辆进行相应的减速动作）。
- **IPG Driver**：先进的、 可自学习的驾驶员模型。可控制在各种行驶工况下的车辆，实现诸如上坡起步、入库泊车以及甩尾反打方向盘等操作。并能适应车辆的动力特性（驱动形式、 变速箱类型等）、道路摩擦系数、风速、交通环境状况，调整驾驶策略。

CarMaker 具有支持高精地图的导入与导出，支持在高性能计算 (HPC) 集群上并行执行大量测试目录，支持在 Docker 容器中运行，具有良好的可移植性和可扩展性等特点。作为平台软件，CarMaker 可以与很多第三方软件进行集成， 如 ADAMS、 AVLCruise、 rFpro 等， 可利用各软件的优势进行联合仿真。同时 CarMaker 配套的硬件，提供了大量的板卡接口， 可以方便的与 ECU 或者传感器进行 HIL 测试。

2.6.2 CarMaker 与 Simulink 联合仿真

CarMaker 自带与 Simulink 联合仿真的功能，CarMaker for Simulink 即是将 IPG 的车辆动力学仿真软件 CarMarker 完全集成到 MathWorks 建模和仿真环境 Matlab/Simulink 中。使用 S 函数以及 MATLAB 和 Simulink API 函数将 CarMaker 功能添加到 Simulink 环境中。CarMaker 与 Simulink 的集成不是松耦合的协同仿真，而是两个应用程序的紧密联系组合，从而形成了兼具性能和稳定性的仿真环境。

在 Simulink 环境中使用 CarMaker 与使用标准 S 函数模块或内置 Simulink 模块没有什么不同。CarMaker 模块的连接方式与其他 Simulink 模块的连接方式一样，只需单击几下鼠标即可将现有 Simulink 模型添加到 CarMaker 车辆模型中。在集成的过程中， CarMaker 功能也得以保留，而且可以与 Simulink 结合使用。

使用 CarMaker GUI 进行模拟控制和参数调整，以及定义机动和道路配置。IPGControl 可用于数据分析和绘图。 IPGMovie 在三维空间中提供逼真的动画和多体车辆模型渲染，使车辆模型栩栩如生。

CarMaker 的仿真结果可以使用 MATLAB cmread 实用程序访问。此实用程序将任何 CarMaker 仿真结果文件的数据加载到 MATLAB 工作区中。一旦数据在工作区中可用，MATLAB 的所有功能，例如图表绘制和平滑功能，都可以用于后处理和绘制仿真结果。使用 C 接口时需要 MATLAB Coder 和 Simulink Coder。

三、OpenPilot 源码阅读

OpenPilot 是由 comma.ai 开发的开放源代码半自动驾驶系统。OpenPilot 可以代替 OEM 的高级辅助驾驶系统，用来改善视觉感知与机电执行器控制。它让用户可以透过增加的计算能力、强化的侦测器以及不断更新的驾驶辅助功能来修改现有的汽车，这些功能会随用户递交的资料而持续改善。

功能：

- **自动车道置中** OpenPilot 使用经过驾驶用户资料训练过的机器学习来决定道路上最安全的路径。这可以改善在没有车道标线的道路上行驶的表现，并透过追踪前方的车道线来维持车道置中。
- **主动式车距维持定速** OpenPilot 能与前前方车辆保持安全的跟车距离。它可以在无用户干预的情况下以随停随走的方式行驶。它使用开放街图的道路曲率与速度限制资料以让车辆在急转弯时放慢速度，并将车辆的速度维持在目前的速限之下。
- **驾驶监控** OpenPilot 会监控驾驶脸部，如果驾驶分心了则会被警示。如果驾驶分心超过六秒，OpenPilot 会将车辆减速至停止，并以声音对用户发出警报。车道变换辅助驾驶开启方向灯时，OpenPilot 会使用此功能来变换车道，驾驶必须在方向盘上进一步动作以确认变换车道。在部分的品牌与车款上，OpenPilot 也会与盲点警示系统交互，以在盲点警示系统侦测到其他车辆时阻止变换车道。
- **软件更新** OpenPilot 通过 WiFi 或移动网络接收软件更新。

OpenPilot 已在不同品牌的汽车上被使用，包括讴歌 Acura、本田 Honda、雷克萨斯 Lexus、丰田 Toyota、现代 Hyundai、起亚 Kia、大众 Volkswagen、斯巴鲁 Subaru 以及日产 Nissan 等人们熟知的汽车品牌。

OpenPiot 可在 PC 端运行，OpenPilot 的所有服务都可以在个人电脑上正常运行，即使没有特殊的硬件或汽车也可以对 OpenPilot 进行开发或试验 ，可以在记录或模拟数据上运行 OpenPilot。 使用 OpenPilot 的工具，可以绘制日志，重放驱动器和观看全分辨率的摄像机流。 还可以使用 CARLA 模拟器在模拟中运行 OpenPilot，这允许 OpenPilot 在你的 PC 端上驾驶虚拟汽车，并在模拟器中可视

化。

3.1 源码目录结构

- **apk** 用于 UI 的 apk 文件
- **cereal** 用于 EON 上所有日志的消息传递规范
- **common** 我们在这里开发的库函数
- **installer/updater** 管理 openpilot 的自动更新
- **opendbc** 如何解读汽车数据的文件
- **panda** 用于在 CAN 和 LIN 上通信的文件
- **phonelib** EON 上使用的库
- **pyextra** EON 上使用的库
- **third_party** 外部引进的库
- **selfdrive** 驾驶汽车的代码
    - **assets** UI 的字体和图像
    - **boardd** 与汽车通信
    - **car** 用于读取状态和控制执行器的汽车特定代码
    - **common** 守护进程的共享 C/C++代码
    - **controls** 感知、计划和控制
    - **lib**
        - **lateral_mpc** 横向的汽车 ADAS 摄像头
        - **longitudinal_mpc** 纵向的汽车 ADAS 摄像头
        - **alertmanger.py** 报警管理-程序
        - **driver_helpers.py** 驾驶助手
        - **latcontrl.py** 横向控制-程序
        - **latcontrol_helpers.py** 横向控制-驾驶助手
        - **longcontrol.py** 纵向控制-油门和刹车
        - **pathplanner.py** 路径规划器-决定在哪里开车
        - **pid.py** PID 算法库
        - **planner.py** 规划器

├── **radar_helpers.py** 雷达助手
├── **speed_smother** 速度平滑库
├── **vehicle_model.py** 车辆型号
├── **controlsd.py** 实际上驾驶汽车
├── **radard.py** 处理雷达数据
├── **Debug** 帮助您调试和执行汽车端口的工具
├── **Locationd** 精确定位
├── **Logcatd** 从 andriod 获取 logcat 信息
├── **Loggerd** 汽车数据记录器和上传器
├── **Proclogd** 进程记录信息
├── **Sensord** IMU/GPS 接口代码
├── **Test** 汽车模拟器通过虚拟操作运行代码
├── **Ui** The UI
├── **Visiond** 视觉通道-与相机交谈，运行模型，保存视频
├── **Manager.py** 管理进程
├── **Messaging.py** 消息传递
├── **Thermal.py** CPU,GPU 等散热

3.2 Openpilot\selfdrive\controls\lib\longitudinal_planner.py

3.2.1 get_max_accel 函数

```
A_CRUISE_MAX_VALS = [1.2, 1.2, 0.8, 0.6]
A_CRUISE_MAX_BP = [0., 15., 25., 40.]


def get_max_accel(v_ego):
return interp(v_ego, A_CRUISE_MAX_BP, A_CRUISE_MAX_VALS)
```

由 A_CRUISE_MAX_VALS 和 A_CRUISE_MAX_BP 可以得到车辆驾驶时速度

与最大加速度之间的线性函数，对当前驾驶车辆速度插值，得到当前速度下的最大加速度。

3.2.2 limit_accel_in_turn 函数

根据现有的横向加速度，得到所允许的纵向加速度限制，避免在转弯失去目标时加速。

```
def limit_accel_in_turns(v_ego, angle_steers, a_target, CP):
a_total_max = interp(v_ego, _A_TOTAL_MAX_BP, _A_TOTAL_MAX_V)
a_y = v_ego*2 * angle_steers * CV.DEG_TO_RAD / (CP.steerRatio *
CP.wheelbase)
a_x_allowed = math.sqrt(max(a_total_max**2 - a_y**2, 0.))
return [a_target[0], min(a_target[1], a_x_allowed)]
```

根据对当前驾驶车辆速度的线性插值得到总加速度，由当前速度和转向角度计算得出横向加速度，从而得到纵向加速度。

3.2.3 class Planner():

方法 def update(self, sm, CP):

```
v_ego = sm['carState'].vEgo
a_ego = sm['carState'].aEgo
v_cruise_kph = sm['controlsState'].vCruise
v_cruise_kph = min(v_cruise_kph, V_CRUISE_MAX)
v_cruise = v_cruise_kph * CV.KPH_TO_MS
```

根据订阅的车辆状态信息得到当前车辆的速度与加速度，由控制状态信息得到巡航速度

```
enabled = (long_control_state == LongCtrlState.pid) or (long_control_state
==
LongCtrlState.stopping)
```

```
if not enabled or sm['carState'].gasPressed:
self.v_desired = v_ego
self.a_desired = a_ego
```

当纵向控制采用 PID 算法或者纵向控制状态为停止时，enable 为真。当 enable 不为真或者踩油门的状态下，期望速度与加速度为当前车辆的速度与加速度

```
accel_limits = [A_CRUISE_MIN, get_max_accel(v_ego)]
accel_limits_turns = limit_accel_in_turns(v_ego,
sm['carState'].steeringAngleDeg,
accel_limits, self.CP)
```

加速度与转向时加速的限制范围

```
if force_slow_decel:
# if required so, force a smooth deceleration
accel_limits_turns[1] = min(accel_limits_turns[1], AWARENESS_DECEL)
accel_limits_turns[0] = min(accel_limits_turns[0], accel_limits_turns[1])
```

如果要求平稳减速，加速的上限不能大于 AWARENESS_DECEL（0.2m/s^2).

```
ccel_limits_turns[0] = min(accel_limits_turns[0], self.a_desired + 0.05)
accel_limits_turns[1] = max(accel_limits_turns[1], self.a_desired - 0.05)
self.mpc.set_accel_limits(accel_limits_turns[0], accel_limits_turns[1])
self.mpc.set_cur_state(self.v_desired, self.a_desired)
self.mpc.update(sm['carState'], sm['radarState'], v_cruise)
self.v_desired_trajectory = np.interp(T_IDXS[:CONTROL_N], T_IDXS_MPC,
self.mpc.v_solution)
self.a_desired_trajectory = np.interp(T_IDXS[:CONTROL_N], T_IDXS_MPC,
self.mpc.a_solution)
self.j_desired_trajectory = np.interp(T_IDXS[:CONTROL_N], T_IDXS_MPC[:-1],
self.mpc.j_solution)
```

修订加速度边界,并由 MPC 算法设置加速度限制，当前状态和进行更新。并由线性插值得到期望速度，加速度和加速度变化率的预测。

```
self.fcw = self.mpc.crash_cnt > 5
if self.fcw:
cloudlog.info("FCW triggered")
```

如果与前导车距离小于安全距离超过 5 次，启动 FCW（前方碰撞预警系统）。

```
a_prev = self.a_desired
self.a_desired = float(interp(DT_MDL, T_IDXS[:CONTROL_N],
self.a_desired_trajectory))
self.v_desired = self.v_desired + DT_MDL * (self.a_desired + a_prev)/2.0
```

更新期望加速度和期望速度作为下一次迭代的初始值

3.3 selfdrive\controls\lib\longitudinal_mpc_lib\long_mpc.py

FCW：为了在合适的时刻触发 FCW（前方碰撞预警系统），需要预测在不久的将来是否需要任何紧急制动。Openpilot 使用模型预测控制(MPC)来构建一个高级规划器，它计算接下来 10 秒的所需速度轨迹。这使用了一种优化算法，基于领先汽车的雷达和视觉信息（例如相对位置和速度）以及当前驾驶汽车的速度和加速度。 这种优化算法试图让汽车远离领头车的“危险区”，同时最大限度地减少乘客因加速和急动而产生的不适。这个“危险区”被定义为驾驶员有不到 1.8 秒的时间避开的距离。如果领头车突然开始非常用力地刹车，则构成碰撞。

3.2.1 方法 def process_lead(self, lead)

```
v_ego = self.x0[1]
if lead is not None and lead.status:
x_lead = lead.dRel
v_lead = lead.vLead
```

```
a_lead = lead.aLeadK
a_lead_tau = lead.aLeadTau
else:
x_lead = 50.0
v_lead = v_ego + 10.0
a_lead = 0.0
a_lead_tau = _LEAD_ACCEL_TAU
```

如果前面有汽车，则从雷达中读取前车信息，否则虚拟出一个前导车以使 MCP 能够正常运行

```
min_x_lead = ((v_ego + v_lead)/2) * (v_ego - v_lead) / (-MIN_ACCEL * 2)
x_lead = clip(x_lead, min_x_lead, 1e8)
v_lead = clip(v_lead, 0.0, 1e8)
a_lead = clip(a_lead, -10., 5.)
lead_xv = self.extrapolate_lead(x_lead, v_lead, a_lead, a_lead_tau)
return lead_xv
```

当前车与前车之间的距离过小时，MPC 将不会收敛，所以修订前车的距离，速度，加速度在限制范围内 方法 def extrapolate_lead(self, x_lead, v_lead, a_lead, a_lead_tau)

```
a_lead_traj = a_lead * np.exp(-a_lead_tau * (T_IDXS**2)/2.)
v_lead_traj = np.clip(v_lead + np.cumsum(T_DIFFS * a_lead_traj), 0.0, 1e8)
x_lead_traj = x_lead + np.cumsum(T_DIFFS * v_lead_traj)
lead_xv = np.column_stack((x_lead_traj, v_lead_traj))
```

根据当前的到的前车距离，前车速度，前车加速度计算得出前车的距离轨迹，速度轨迹与加速度轨迹

3.2.2 方法 def update(self, carstate, radarstate, v_cruise):

```
self.run()
if (np.any(lead_xv_0[:,0] - self.x_sol[:,0] < CRASH_DISTANCE) and
radarstate.leadOne.modelProb > 0.9):
self.crash_cnt += 1
else:
self.crash_cnt = 0
```

3.3 车道规划 Oenpilot\selfdrive\controls\lib\lane_planner.py

3.3.1 parse_model(self, md)

从驾驶模型中接收左右车道线，左车道和右车道概率和标准偏差

```
left and right ll x is the same
self.ll_x = md.laneLines[1].x
```

左车道和右车道在纵向上位置相同

```
only offset left and right lane lines; offsetting path does not make sense
self.lll_y = np.array(md.laneLines[1].y) - self.camera_offset
self.rll_y = np.array(md.laneLines[2].y) - self.camera_offset
```

仅偏移左右车道线，偏移路径没有意义

```
self.lll_prob = md.laneLineProbs[1]
self.rll_prob = md.laneLineProbs[2]
self.lll_std = md.laneLineStds[1]
self.rll_std = md.laneLineStds[2]
```

左右车道线的概率和标准偏差

```
if len(md.meta.desireState):
self.l_lane_change_prob =
md.meta.desireState[log.LateralPlan.Desire.laneChangeLeft]
self.r_lane_change_prob =
md.meta.desireState[log.LateralPlan.Desire.laneChangeRight]
```

向左变换车道和向右变换车道的概率

3.3.2 get_d_path(self, v_ego, path_t, path_xyz)

```
Reduce reliance on lanelines that are too far apart or will be in a few
seconds
path_xyz[:, 1] -= self.path_offset
l_prob, r_prob = self.lll_prob, self.rll_prob
width_pts = self.rll_y - self.lll_y
prob_mods = []
for t_check in [0.0, 1.5, 3.0]:
width_at_t = interp(t_check * (v_ego + 7), self.ll_x, width_pts)
prob_mods.append(interp(width_at_t, [4.0, 5.0], [1.0, 0.0]))
mod = min(prob_mods)
l_prob *= mod
r_prob *= mod
```

减少对相距太远或将在几秒钟内出现的车道线的依赖,根据左右车道线的横向位置的到对应点的车道宽度，由插值得到对应时间车道的宽度，从而得到概率模型，更新左右车道的概率

```
l_std_mod = interp(self.lll_std, [.15, .3], [1.0, 0.0])
r_std_mod = interp(self.rll_std, [.15, .3], [1.0, 0.0])
l_prob *= l_std_mod
r_prob *= r_std_mod
```

减小对不确定车道线的依赖

```
#Find current lanewidth
self.lane_width_certainty.update(l_prob * r_prob)
current_lane_width = abs(self.rll_y[0] - self.lll_y[0])
self.lane_width_estimate.update(current_lane_width)
speed_lane_width = interp(v_ego, [0., 31.], [2.8, 3.5])
self.lane_width = self.lane_width_certainty.x * self.lane_width_estimate.x + \
(1 - self.lane_width_certainty.x) * speed_lane_width
```

由当前车道的宽度以及根据速度插值得到的车道宽度经过计算得到新的车道宽度

```
clipped_lane_width = min(4.0, self.lane_width)
path_from_left_lane = self.lll_y + clipped_lane_width / 2.0
path_from_right_lane = self.rll_y - clipped_lane_width / 2.0
```

车道宽度不能超过 4m，计算出汽车路径相对于左右车道线的位置

```
self.d_prob = l_prob + r_prob - l_prob * r_prob
lane_path_y = (l_prob * path_from_left_lane + r_prob * path_from_right_lane)
/
(l_prob + r_prob + 0.0001)
safe_idxs = np.isfinite(self.ll_t)
if safe_idxs[0]:
lane_path_y_interp = np.interp(path_t, self.ll_t[safe_idxs],
lane_path_y[safe_idxs])
path_xyz[:,1] = self.d_prob * lane_path_y_interp + (1.0 - self.d_prob) *
path_xyz[:,1]
else:
cloudlog.warning("Lateral mpc - NaNs in laneline times, ignoring")
return path_xyz
```

由车道线概率和路径相对车道线的位置计算出车道路径的 y 方向的位置，根据车道线时间和路径位置的关系得到不同时间下的路径坐标，再结合当前路径的坐标更新路径坐标。

3.4 基于角度的横向控制

openpilot\selfdrive\controls\lib\latcontrol_angle.py

3.4.1 update(self, active, CS, CP, VM, params, desired_curvature,desired_curvature_rate)

```
if CS.vEgo < 0.3 or not active:
angle_log.active = False
angle_steers_des = float(CS.steeringAngleDeg)
```

如果当前车速小于 0.3 或者横向控制关闭的状态下，期望的转向角度等于从车状态读取的转向角度度数，并且关闭角度日志。

```
else:
angle_log.active = True
angle_steers_des =
math.degrees(VM.get_steer_from_curvature(-desired_curvature, CS.vEgo))
angle_steers_des += params.angleOffsetDeg
```

否则，打开角度日志，预期的转向角度等于由期望曲率和当前车速得到的转向角度加上角度偏移度数 openpilot\selfdrive\controls\controlsd.py

```
state_control(self, CS):
angle_control_saturated = self.CP.steerControlType ==
car.CarParams.SteerControlType.angle and \
abs(actuators.steeringAngleDeg - CS.steeringAngleDeg) >
STEER_ANGLE_SATURATION_THRESHOLD
```

```
if angle_control_saturated and not CS.steeringPressed and self.active:
self.saturated_count += 1
else:
self.saturated_count = 0
```

当转向控制类型为基于角度的控制并且期望角度与当前车转向角度之差大于转向角度饱和临界值时，角度控制饱和值为真 当角度控制饱和值为真且当前驾驶员为没有操纵转向盘时，饱和计数器加 1

```
# Send a "steering required alert" if saturation count has reached the limit
if (lac_log.saturated and not CS.steeringPressed) or \
(self.saturated_count > STEER_ANGLE_SATURATION_TIMEOUT):
if len(lat_plan.dPathPoints):
# Check if we deviated from the path
left_deviation = actuators.steer > 0 and lat_plan.dPathPoints[0] < -0.1
right_deviation = actuators.steer < 0 and lat_plan.dPathPoints[0] > 0.1
if left_deviation or right_deviation:
self.events.add(EventName.steerSaturated)
```

如果饱和计数已达到限制，则发送“需要转向的警报” 如果饱和计数超过限制范围，检测当前车辆是否从路径偏航，如果偏航，则添加转向饱和事件，发送如下警告

```
EventName.steerSaturated: {
ET.WARNING: Alert(
"TAKE CONTROL",
"Turn Exceeds Steering Limit",
AlertStatus.userPrompt, AlertSize.mid,
Priority.LOW, VisualAlert.steerRequired, AudibleAlert.chimePrompt, 1.,
1.,1.),
}
```

检测前车与当前车距离小于安全距离的次数，如果超过一定次数，则会触发FCW

3.5 结构体及参数定义 openpilot\cereal\car.capnp

主要定义了 4 种结构体用以存储在自动驾驶过程中需要保存和利用的各种参数：

- 结构体 CarEvent(导致驾驶车状态发生变化的事件)
- 结构体 CarState(存储驾驶车的状态)
- 结构体 CarControl(驾驶车控制)
- 结构体 CarParams(存储汽车参数)

3.6 openpilot-lgsvl_bridge\selfdrive\controls\controlsd.py

3.6.1 函数 controlsd_thread(sm=None, pm=None)

```
if pm is None:
pm = messaging.PubMaster(['controlsState', 'carState', 'opToSim'])
```

发布的消息： controlsState：控制状态

```
dat = messaging.new_message()
dat.init('controlsState')
dat.valid = True
dat.controlsState = {
"pathPlanMonoTime": sm.logMonoTime['pathPlan'],
"enabled": isEnabled(state),
"active": isActive(state),
"vEgo": CS.vEgo,
"vEgoRaw": CS.vEgoRaw,
"angleSteers": CS.steeringAngle,
"curvature": VM.calc_curvature((CS.steeringAngle -
sm['pathPlan'].angleOffset) * CV.DEG_TO_RAD, CS.vEgo),
```

```
"steerOverride": CS.steeringPressed,
"state": state,
"engageable": not bool(get_events(events, [ET.NO_ENTRY])),
"vCruise": float(v_cruise_kph),
}
pm.send('controlsState', dat)
```

包括当前车速度，转向角，曲率，是否使用方向盘，巡航速度等

```
carState：汽车状态
cs_send = messaging.new_message()
cs_send.init('carState')
cs_send.valid = CS.canValid
cs_send.carState = CS
cs_send.carState.events = events
pm.send('carState', cs_send)
opToSim：openpilot 发送给仿真器的消息
ctrl_event = messaging.new_message()
ctrl_event.init('opToSim')
ctrl_event.opToSim.steering = apply_angle_ratio
pm.send('opToSim', ctrl_event)
```

主要是所施加的转向角

```
if sm is None:
sm = messaging.SubMaster(['pathPlan', 'gpsLocation', 'simState'],
ignore_alive=['gpsLocation'])
```

订阅的消息： pathPlan:路径规划 gpsLocation:GPS 定位 simState:仿真器状态

```
CarInterface, CarController = interfaces['mock']
CP = CarInterface.get_params('mock', None)
CI = CarInterface(CP, CarController)
```

获取模拟汽车的参数和接口

```
LoC = LongControl(CP, CI.compute_gb)
VM = VehicleModel(CP)
if CP.lateralTuning.which() == 'pid':
LaC = LatControlPID(CP)
elif CP.lateralTuning.which() == 'indi':
LaC = LatControlINDI(CP)
elif CP.lateralTuning.which() == 'lqr':
LaC = LatControlLQR(CP)
```

汽车的纵向控制与横向控制

```
sm['pathPlan'].sensorValid = True
sm['pathPlan'].posenetValid = True
vEgo = 0.
steeringAngle = 0.
CS = update_state(CI, CC, vEgo, steeringAngle)
last_angle = CS.steeringAngle
```

设置传感器有效，姿势网有效，从模拟汽车接口读入汽车状态并得到汽车方向盘最新转向角度主循环：

```
start_time = sec_since_boot()
events = data_sample(CS, sm, state)
```

获取启动时间，采样数据并计算汽车事件

```
if sm['pathPlan'].mpcSolutionValid and sm['pathPlan'].sensorValid and
sm['pathPlan'].paramsValid:
path_plan = sm['pathPlan']
logger.debug("Recv'd PathPlan: angle offset %s, desired steer angle %s",
"{:.2f}".format(path_plan.angleOffset),
"{:.4f}".format(path_plan.angleSteers))
```

```
angle_steers_des = path_plan.angleSteers - path_plan.angleOffset
angle_steers_cur = CS.steeringAngle - path_plan.angleOffset
```

路径规划得到的转向角与从车辆状态信息得到的转向角分别减去转向角偏移，得到方向盘的期望转向角与当前转向角

```
if last_angle * apply_angle > 0. and abs(apply_angle) > abs(last_angle):
angle_rate_lim = interp(vEgo, ANGLE_DELTA_BP, ANGLE_DELTA_V)
else:
angle_rate_lim = interp(vEgo, ANGLE_DELTA_BP, ANGLE_DELTA_VU)
apply_angle = clip(apply_angle, last_angle - angle_rate_lim, last_angle +
angle_rate_lim)
```

根据当前车速计算不同情况下转向角变化的的范围，并得到施加角度的限制范围区间

```
apply_angle_ratio = -apply_angle/ANGLE_MAX_V[0]
logger.debug("speed %s, last steer angle %s, applying steer angle %s",
"{:.2f}".format(vEgo), "{:.2f}".format(last_angle),
"{:.2f}".format(apply_angle))
last_angle = apply_angle
```

LGSVL 接受 [-1, 1]范围内的角度（不是增量），所以要计算得到施加转向角与最大转向角的比值，正数表示向右，负数表示向左

```
ctrl_event = messaging.new_message()
ctrl_event.init('opToSim')
ctrl_event.opToSim.steering = apply_angle_ratio
pm.send('opToSim', ctrl_event)
```

将转向角发布给 opToSim

```
sm.update()
if sm.updated['simState']:
vEgo = sm['simState'].speed
steeringAngle = -sm['simState'].steer * ANGLE_MAX_V[0]
logger.debug("Recv'd SimState from LGSVL")
elif sm.updated['pathPlan']:
logger.debug("Failed to recv SimState, but a PathPlan")
else:
logger.debug("Failed to recv SimState")
```

从订阅消息 simState 得到仿真器中汽车的状态，包括汽车速度，方向盘转向角度，并将读取记录写入日志

```
CS = update_state(CI, CC, vEgo, steeringAngle)
data_send_no_can(sm, pm, CS, CI, CP, VM, state, events,v_cruise_kph, AM,
LaC, LoC,start_time, events_prev)
```

更新汽车状态，发布控制状态消息以及汽车状态消息

3.7 openpilot-lgsvl_bridge\start_sim.py

```
sim_config = "simconfig.ini"



[log]
dir={full_path_to_dir_to_store_simulation_results}/data
[scenario]
map=sl_local
time_of_day=12
vehicle=Jaguar XE 2015
velocity_x=0.0
velocity_y=0.0
velocity_z=20.0
```

读入模拟器配置，模拟器参数包括帧目录，地图，车辆，三个方向上的速度，模拟器内时间

```
simargs = {"frame_dir": frame_dir,
"sim_map": sim_map,
"vehicle": vehicle,
"sim_vel_x": sim_vel_x,
"sim_vel_y": sim_vel_y,
"sim_vel_z": sim_vel_z,
"time_of_day": time_of_day}
Managed_processes 增加了
"lgsvlsim": (True, ["selfdrive.controls.lgsvlsim"]
def manager_thread
```

订阅消息

```
sm = messaging.SubMaster(['simState', 'model'])
```

启动模拟器

```
for p in sim_processes:
start_managed_process(p)
```

保存某一帧的 model 消息，包括路径，左右车道

```
if sm.updated['model']:
polys = np.zeros((3,4))
polys[0,:] = np.array(sm['model'].path.poly)
polys[1,:] = np.array(sm['model'].leftLane.poly)
polys[2,:] = np.array(sm['model'].rightLane.poly)
trans = np.array(sm['model'].settings.inputTransform).reshape(3,3)
np.save(poly_dir + '/poly_' + str(sm['model'].frameId), polys)
np.save(trans_dir + '/trans_' + str(sm['model'].frameId), trans)
# Junjie: newly added, to be extended later
```

```
model = np.array([sm['model'].leftLane.prob,
sm['model'].rightLane.prob])
np.save(model_dir + '/model_' + str(sm['model'].frameId), model)
```

保存某一帧的 simState 消息，包括时间，速度，转向，预期转向角，位置，旋转角度等

```
if sm.updated['simState']:
frame_id = sm['simState'].frameId
sim_state = np.array(
[sm['simState'].time, sm['simState'].speed,
sm['simState'].steer, sm['simState'].desiredSteer] +
list(sm['simState'].position) +
list(sm['simState'].velocity) +
list(sm['simState'].attitude))
np.save(state_dir + '/state_' + str(sm['model'].frameId), sim_state)
```

3.8 openpilot-lgsvl_bridge\selfdrive\controls\simcontrol

与 simulator 建立连接 python 程序需要知道跟哪个 simulator 互动，这里就需要建立 python 程序和 simulator 的连接。这本质上是在 python 程序中实体化 Simulator 类的一个对象，送入两个参数：simulator 所在的 IP 地址和程序占用的端口，程序如下：

```
self.sim = lgsvl.Simulator(os.environ.get("SIMULATOR_HOST", "127.0.0.1"),
8181)
```

通过 Simulator 类的 load()函数实现加载仿真环境，在仿真运行过程中，有时并不是冷启动加载环境，而是已经在环境中，需要重置到初始状态，即只保留仿真环境，清空所有加载的车辆和行人，这时可以用 reset()函数，速度比 load()更快

```
scene_name = sim_map
if self.sim.current_scene == scene_name:
self.sim.reset()
else:
  self.sim.load(scene_name, seed=650387) # fix the seed
```

模拟车辆的速度与模拟环境的时间设置

```
sim_speed = math.sqrt(sim_vel[0]**2 + sim_vel[1]**2 + sim_vel[2]**2)
print(time_of_day)
self.sim.set_time_of_day(float(time_of_day), fixed=True)
print("Current time of day:", self.sim.time_of_day)
```

Spawns 获取仿真环境内适合车辆放置的位置信息，并选择第一个生成的位置作为放置车辆位置

```
spawns = self.sim.get_spawn()
state = lgsvl.AgentState()
state.transform = spawns[0]
state.velocity = lgsvl.Vector(sim_vel[0], sim_vel[1], sim_vel[2])
```

向仿真环境中添加本车

```
vehicle_name = vehicle
self.ego = self.sim.add_agent(vehicle_name, lgsvl.AgentType.EGO, state)
```

启动传感器

```
sensors = self.ego.get_sensors()
self.main_cam = None
self.free_cam = None
for s in sensors:
if s.name == "Main Camera":
```

```
s.enabled = True
self.main_cam = s
elif s.name == "Free Camera":
s.enabled = True
self.free_cam = s
else:
s.enabled = False
```

控制车辆的油门，转向，刹车，倒车，手刹

```
self.ctrl = lgsvl.VehicleControl()
self.ctrl.throttle = 0.0
self.ctrl.steering = 0.0
self.ctrl.braking = 0.0
self.ctrl.reverse = False
self.ctrl.handbrake = False
```

NPC 车辆向前的单位向量和向右的单位向量

```
forward = lgsvl.utils.transform_to_forward(spawns[0]) # forward unit
vector
right = lgsvl.utils.transform_to_right(spawns[0]) # right unit vector
```

获取模拟环境中适合 NPC 车辆的生成位置，并在该位置处添加 NPC 车辆

```
state = lgsvl.AgentState()
lateral_shift = -2.7
forward_shift = 274.0
state.transform.position = spawns[0].position + lateral_shift * right +
forward_shift * forward
state.transform.rotation = spawns[0].rotation
state.transform.rotation.y = spawns[0].rotation.y + 180.0
npc = self.sim.add_agent("BoxTruck", lgsvl.AgentType.NPC, state)
```

follow_closest_lane()命令让 NPC 沿当前车道行驶，如果当前车辆横跨在两条车道之间，则驶入最近的车道

```
npc.follow_closest_lane(True, 20)
def next_yuv_frame(self)
```

保存从 main camera 和 free camera 获得的帧图片，返回帧 id 和 yuv 图片。 def apply_control(self, steering) 控制车辆状态，包括转向，油门，速度等 def get_sim_state(self) 获取 simState，包括帧 id，仿真时间，本车速度，轮角比，车辆位置，速度，旋转角度

3.9 openpilot-lgsvl_bridge\selfdrive\controls\lgsvlsim.py

```
pm = messaging.PubMaster(['controlsState', 'frame', 'simState', 'carState'])
sm = messaging.SubMaster(['pathPlan'])
```

发布 frame 消息

```
frame_id, yuv_img = sim.next_yuv_frame()
frame_dat = messaging.new_message()
frame_dat.init('frame')
frame_dat.frame.frameId = frame_id
frame_dat.frame.image = yuv_img.tobytes()
pm.send('frame', frame_dat)
logger.debug("Sim time %5.2f, frame %d sent", sim.sim_time, frame_id)
```

发布 simState 消息

```
sim_dat = messaging.new_message()
sim_dat.init('simState')
sim_dat.simState.time = sim_state.time
sim_dat.simState.frameId = sim_state.frame_id
sim_dat.simState.speed = sim_state.speed
```

```
sim_dat.simState.steer = curr_steer_angle
sim_dat.simState.desiredSteer = path_plan.angleSteers
sim_dat.simState.position = sim_state.position
sim_dat.simState.velocity = sim_state.velocity
sim_dat.simState.attitude = sim_state.attitude
pm.send('simState', sim_dat)
```

3.10 manager 文件夹

3.10.1 process.py

函数：

- def launcher(proc):启动进程
- def nativelauncher(pargs, cwd):启动本地进程
- def join_process(process, timeout):
- def ensure_running(procs, started, driverview=False, not_run=None):检查进程运行状态

类：

- class ManagerProcess(ABC):管理进程
- def prepare(self):准备
- def start(self):启动
- def restart(self)：重启
- def check_watchdog(self, started):检查 watchdog

Watchdog，又称 watchdog timer，是计算机可靠性（dependability）领域中一个极为简单同时非常有效的检测（detection）工具。其基本思想是针对被监视的目标设置一个计数器和一个阈值，watchdog 会自己增加计数值，并等待被监视的目标周期性地重置计数值。一旦目标发生错误，没来得及重置计数值，watchdog 会检测到计数值溢出，并采取恢复措施（通常情况下是重启）。总结一下就是计数——溢出——触发。

-def stop(self, retry=True, block=True):停止进程

- def get_process_state_msg(self):获取当前进程信息

- class NativeProcess(ManagerProcess):本地进程（ManagerProcess 的子类）

- def__init__：初始化

- def prepare(self):准备

- def start(self):启动

- class PythonProcess(ManagerProcess):python 进程（ManagerProcess 的子类）
    - def**init**：初始化
    - def prepare(self):准备
    - def start(self):启动
- class DaemonProcess(ManagerProcess):守护进程（ManagerProcess 的子类）
    - def **init**：初始化
    - def prepare(self):准备
    - def start(self):启动
    - def stop(self, retry=True, block=True):停止

3.10.2 process_config.py——进程配置

DaemonProcess:manage_athenad

NativeProcess:camerad,clocksd,dmonitoringmodeld,logcatd,loggerd,modeld,navd,Proclogd,sensord,ubloxd,ui,soundd,locationd,boardd

PythonProcess:calibrationd,controlsd,deleter,dmonitoringd,logmessaged,pandad,paramsd,plannerd,radard,thermald,timezoned,tombstoned,updated,uploader

3.10.3 helpers.py——获取非阻塞标准输出

```
child_pid, child_pty = os.forkpty()
```

系统调用创建子进程

```
dat = os.read(child_pty, 4096)
```

从文件描述符 child_pty 中读取最多 4096 个字节，返回包含读取字节的字符串

```
sys.stdout.write(dat.decode('utf8'))
```

标准化输出

3.10.4 Manager.py——对进程进行管理

- def manager_init():初始化 为 panda 更新系统时间，设置参数，创建文件夹，设置 dongle 参数……
- def manager_prepare():准备 对于在 managed_process 中的进程 p，都执行 p.prepare 函数
- def manager_cleanup():清除 对于在 managed_process 中的进程 p，都执行 p.stop 函数，并发布日志
- def manager_thread(): 创建新的进程

```
subprocess.call("./bootlog", cwd=os.path.join(BASEDIR,
"selfdrive/loggerd"))
params = Params()
ignore = []
if params.get("DongleId", encoding='utf8') == UNREGISTERED_DONGLE_ID:
ignore += ["manage_athenad", "uploader"]
if os.getenv("NOBOARD") is not None:
ignore.append("pandad")
if os.getenv("BLOCK") is not None:
ignore += os.getenv("BLOCK").split(",")
ensure_running(managed_processes.values(), started=False,
not_run=ignore)
```

循环执行更新，发布进程消息

```
while True:
sm.update()
```

```
not_run = ignore[:]
if sm['deviceState'].freeSpacePercent < 5:
not_run.append("loggerd")
started = sm['deviceState'].started
driverview = params.get_bool("IsDriverViewEnabled")
ensure_running(managed_processes.values(), started, driverview, not_run)
# trigger an update after going offroad
if started_prev and not started and 'updated' in managed_processes:
os.sync()
managed_processes['updated'].signal(signal.SIGHUP)
started_prev = started
running_list = ["%s%s\u001b[0m" % ("\u001b[32m" if p.proc.is_alive() else
"\u001b[31m", p.name)
for p in managed_processes.values() if p.proc]
cloudlog.debug(' '.join(running_list))
# send managerState
msg = messaging.new_message('managerState')
msg.managerState.processes = [p.get_process_state_msg() for p in
managed_processes.values()]
pm.send('managerState', msg)
# TODO: let UI handle this
# Exit main loop when uninstall is needed
if params.get_bool("DoUninstall"):
break
```

main 函数：执行 manager_init, manager_prepare, manager_thread, manager_cleanup

```
def main():
prepare_only = os.getenv("PREPAREONLY") is not None
manager_init()
# Start UI early so prepare can happen in the background
if not prepare_only:
managed_processes['ui'].start()
```

```
manager_prepare()
if prepare_only:
Return
# SystemExit on sigterm
signal.signal(signal.SIGTERM, lambda signum, frame: sys.exit(1))
try:
manager_thread()
except Exception:
traceback.print_exc()
crash.capture_exception()
finally:
manager_cleanup()
if Params().get_bool("DoUninstall"):
cloudlog.warning("uninstalling")
HARDWARE.uninstall()
```

四、仿真地图的生成

大部分仿真器都是支持 OpenDRIVE 的。支持 OpenSCENARIO 的软件，目前看到的有 VTD 和 Carla、51Sim-One(51Sim-One1.2，2020 年发布)、Prescan(PreScan2019.3 版本开始)。LGSVL 暂时没有资料显示支持 OpenSCENARIO。

仿真器	是否支持 OPENDRIVE	是否支持 OPENSCENARIO	版本
VTD	支持	支持	VTD 2.2
Carla	支持	支持	Carla 0.9.8
51Sim-One	支持	支持	51Sim-One 1.2
Prescan	支持	支持	PreScan 2019.3
AirSim	没有资料显示支持	没有资料显示支持	-
LGSVL	支持	没有资料显示支持	LGSVL 2020.05
AADS	没有资料显示支持	没有资料显示支持	-
TAD Sim	没有资料显示支持	没有资料显示支持	-

4.1 OpenDRIVE

作为一个完整的仿真测试场景描述方案，OpenX 系列标准包括：OpenDRIVE、OpenSCENARIO 和 OpenCRG。仿真测试场景的静态部分（如道路拓扑结构、交通标志标线等）由 OpenDRIVE 文件描述，道路的表面细节（如坑洼、卵石路等）由 OpenCRG 文件描述，仿真测试场景的动态部分（如交通车的行 为）由 OpenSCENARIO 文件描述。 OpenDRIVE 是对路网结构的描述性文件，OpenDRIVE 将道路（roads）分为三个部分：道路参考线（reference line）、车道（lanes）和道路设施（features）。除此以外，还可以设置道路的高度（elevation），对于多条道路汇聚的位置需要用路口（junctions）来描述。 官方文档中提供了一

个 OpenDRIVE 独立模式，Carla 允许用户将任何 OpenDRIVE 文件作为现成的 Carla 地图摄取。为了做到这一点，模拟器将自动生成一个道路网格供 actor 导航。此模式仅使用 OpenDRIVE 文件运行完整模拟，而不需要任何其他几何图形或资源。为此，模拟器获取一个 OpenDRIVE 文件，并按程序创建一个临时三维网格来运行模拟。生成的网格以极简的方式描述道路定义。所有元素都将与 OpenDRIVE 文件相对应。 交通信号灯、停车场和停车场将实时生成。行人将在地图上显示的人行道和人行横道上运动。所有这些元素，以及路上的每一个细节，都是基于 OpenDRIVE 文件的。

OpenDRIVE 独立模式 *https://carla.readthedocs.io/en/latest/adv_opendrive/*

4.2 RoaderRunner

RoadRunner 使任何人都可以轻松创建环境，逼真的道路和道具。
可以用于生成包含回旋处，交叉路口和桥梁的复杂道路网络，自定义标牌，标记和道具，然后发布到 OpenDRIVE，Unreal，Unity 或 FBX 等，以实现全面的灵活性。现在可用于 Windows（7 x64 或更高版本）Linux（Ubuntu x64 16.04）以及更多版本。

4.3 使用 OpenStreetMap 生成地图

OpenStreetMap 是开源的世界开放许可证地图。这些地图的各个部分可以导出为 XML 格式的 .osm 文件。CARLA 可以将文件转换为 OpenDRIVE 格式，并使用 OpenDRIVE 独立模式导入它。 整个过程如下：

- 使用 OpenStreetMap 获取地图
- 转换为 OpenDRIVE 格式
- 导入 Carla

具体操作过程：

https://carla.readthedocs.io/en/latest/tuto_G_openstreetmap/#ingest-into-carla

五、Carla 模拟撞击事故

5.1 Carla 自定义地图

根据官方文档给出自定义地图教程，使用 RoadRunner 完成，暂时没有找到好的替代方案。

5.2 思路及流程

通过在路上画黑色线条的方法，从而伪造车道线，使车辆偏离原方向。使用 Carla 模拟器进行仿真。 因为 Carla 功能完善且源代码易于修改，所以选择 Carla 进行仿真。Carla 有足够的灵活性来创建逼真的模拟环境，具有强大的物理引擎，逼真的照明，包括道路、建筑、交通标志、车辆和行人等 3D 对象。 在实验的时候，只使用 RGB 相机，禁用激光雷达(光探测和测距)、语义分割和深度相机。Carla 还允许用户在各种天气条件下运行实验，如日落、阴天和下雨，这是由客户端输入决定的。为了保持帧速率和执行时间的一致性，使用固定的时间步长运行 CARLA。 Carla 提供了两个经过训练的端到端模型：条件模仿学习和强化学习。论文作者主要集中在攻击条件模仿学习模型上。

为了以系统的方式产生物理上可实现的敌手(故意做出某种行为来加大 AI 对事件判断的误差)，修改了 Carla 的源代码。

Carla 模拟器(v0.8.2)不允许生成在 Carla 蓝图库中不存在的对象到场景中(包括车辆、行人和道具的模型)。通过虚幻引擎 4 (UE4)，创建了一个新的对抗性平面蓝图，这是一个 200×200 像素平面或画布与动态 UE4 材料，可以覆盖在所需的道路部分。这个蓝图的关键属性是读取生成的攻击图像(.png 文件)，并将其实时放置在 CARLA 中。因此，该蓝图具有通过 HTTP 服务器连续读取图像的能力。这个画布允许使用带有 alpha 通道的图像，允许部分透明的攻击。当 CARLA 运行时，在 HTTP 服务器上找到的图像将覆盖在每个画布上。

最后，编译并封装了这个修改版的 CARLA。因此能够在 Carla 模拟器中放置物理攻击。定义一个模式生成器，用于生成相应的攻击模式。一旦生成模式，就通过 HTTP 服务器读取，并放置在 Carla 模拟器中。 由于 Carla 几乎是实时运行

的，实验需要很长时间才能运行。为了有效地使用所需的参数运行模拟，将 CARLA 设置转换为 OpenAI-Gym 环境。有了这个设置，就可以使用优化器来生成一组攻击，运行一个场景并得到结果，输出指标。

为了确保在不同设置下测试不同攻击的有效性，通过改变各种环境参数进行实验，如地图(训练地图和测试地图)、场景、天气(晴空、下雨、日落)、驾驶场景(直路、右转、左转)、端到端模型(IL 和 RL)和整个模式搜索空间。 在此描述了 6 个可用的 Carla 场景：

- **右转**：汽车沿着一条向右转 90 度的车道行驶
- **左转**：汽车沿着一条向左转 90 度的车道行驶
- **直道**：汽车沿着直道走
- **十字路口右转**：汽车在十字路口右转
- **十字路口左转**：汽车在十字路口左转
- **十字路口直行**：汽车直接直行通过十字路口

以 10 帧每秒(fps)的速度运行实验，并为每一帧相机收集以下数据(一个典型的实验需要 60 到 100 帧运行):安装的 RGB 摄像头的相机图像，车辆速度，预测加速度，转向和刹车，车辆在错误车道的百分比，车辆在人行道上(越道)的百分比，车辆的 GPS 位置，碰撞强度。为了彻底搜索设计空间，构建了一个 CARLA docker，允许同时运行多达 16 个 CARLA 实例，分布在 8 个 RTX gpu 上。

5.3 结果

车辆偏离原方向并发生碰撞，通过 Carla 可视化看到发生碰撞的结果。

参考文献

[1] Wang Y , Tan Y A , Zhang W , et al. An adversarial attack on DNN-based black-box object detectors[J]. Journal of Network and Computer Applications, 2020, 161:102634.

[2] Boloor A , Garimella K , He X , et al. Attacking Vision-based Perception in End-to-End Autonomous Driving Models[J]. Journal of Systems Architecture, 110.

[3]李玉峰, 陆肖元, 曹晨红,等. 智能网联汽车网络安全浅析[J]. 电信科学, 2020, 36(4):10.

[4]胡文, 姜立标. 智能网联汽车的多级安全防护方案设计和分析[J]. 网络安全技术与应用, 2017(2):4.

[5] Rong G , Shin B H , Tabatabaee H , et al. LGSVL Simulator: A High Fidelity Simulator for Autonomous Driving[C]// 2020.

[6] Jiao R , Liang H , Sato T , et al. End-to-end Uncertainty-based Mitigation of Adversarial Attacks to Automated Lane Centering[C]// 2021.

[7] HJCho, Behl M . Towards Automated Safety Coverage and Testing for Autonomous Vehicles with Reinforcement Learning[J]. arXiv, 2020.

[8] Haklay M , Weber P . OpenStreetMap: User-Generated Street Maps[J]. IEEE Pervasive Computing, 2008, 7(4):12-18.