

自动驾驶人机交互系统研究

摘 要

随着硬件的更新、人工智能和移动互联网技术的应用和推广，自动驾驶逐渐进入人们的视野。当今，整个汽车产业的最新发展方向就是自动驾驶，因为其可以对驾驶的安全性和舒适性进行全面的提升以满足更高层次的市场需求，并且能在未来对生活方式、工业生产以及社会管理等多方面带来变革。随着自动驾驶的不断发展，对自动驾驶人机交互系统的研究也逐渐走上日程。现阶段，即使是研究自动驾驶系统的相关人士，在配置和启动自动驾驶系统方面就存在诸多困难，更不用说让用户能够轻松的使用自动驾驶技术了。研究一种对用户和开发者都具有良好体验的自动驾驶人机交互系统，在一定程度上能够降低自动驾驶的使用门槛，提升自动驾驶相关项目的推进效率。

本课题与现实紧密联系，针对自动驾驶系统配置困难、使用麻烦的问题进行了较好的解决，能够在多数情况下实现自动化配置。能够降低使用自动驾驶系统的门槛，提高自动驾驶系统相关开发者的工作效率，并能够和其他技术进行结合与拓展。

关键词：自动驾驶 人机交互 车联网

Research of human-computer interaction system for autonomous driving

Abstract

With the development of hardware, artificial intelligence and the application and popularization of mobile Internet technology, self-driving has gradually entered people's vision. Today, the latest development in the entire automotive industry is self-driving, because it can improve the safety and comfort of driving to meet higher market demand, and in the future for lifestyle, industrial production and social management and other aspects of the transformation. With the continuous development of self-driving, the research on the interactive system of self-driving human-computer is gradually on the agenda. At this stage, even those involved in self-driving systems have many difficulties in configuring and activating autonomous driving systems, let alone making it easy for users to use autonomous driving technology. To study a self-driving human-computer interaction system that has a good experience for both users and developers, it can reduce the threshold of use of autonomous driving and improve the efficiency of self-driving-related projects.

This topic is closely related to reality, and the problem of difficulty and use of self-driving system can be solved well, and the automation configuration can be realized in most cases. It can lower the threshold for using self-driving systems, improve the efficiency of developers associated with autonomous driving systems, and be able to integrate and expand with other technologies.

Key Words: Autonomous Driving; Human-Computer Interaction; Internet of Vehicles

目 录

1. 引言	1
1.1 课题研究的背景及意义	1
1.2 国内外研究及发展现状	1
1.3 本文的主要工作与结构	2
2. 主要技术介绍	3
2.1 硬件环境概要	3
2.1.1 车体和控制器	3
2.1.2 车载传感器	4
2.1.3 车载计算单元	5
2.2 软件环境概要	6
2.2.1 ROS 与 C++	6
2.2.2 .NET Core 与 C#	6
3. 系统设计与实现	8
3.1 系统框架结构	8
3.2 开发环境	9
3.2.1 硬件平台	9
3.2.2 软件平台	9
3.3 车辆控制器服务实现	9
3.3.1 CAN 适配器硬件抽象层	10
3.3.2 CAN 协议调试	10
3.3.3 车辆控制器协议抽象层	11
3.3.4 车辆控制器调试	12
3.3.5 通用控制部分	13
3.4 摄像头服务实现	14
3.4.1 图像采集	15
3.4.2 畸变矫正	15
3.4.3 图像缩放	15

3.4.4 发布图像.....	16
3.5 一键启动实现.....	16
3.5.1 自动驾驶算法框架.....	16
3.5.2 一键启动脚本.....	16
3.6 Web 服务器实现.....	16
3.6.1 Web 服务器.....	16
3.6.2 Web 通信.....	17
3.6.3 人机交互 Web 服务器到 Web 客户端的车载摄像头内容传输	17
3.7 信息交互实现.....	18
3.7.1 车辆信息展示.....	18
3.7.2 车辆当前位置和一键召唤.....	18
3.7.3 车辆基本控制.....	19
3.8 远程调试实现.....	20
3.8.1 远程调试服务器.....	20
3.8.2 远程调试 Web 客户端.....	21
3.8.3 反向代理.....	21
4. 系统测试.....	22
4.1 运行截图.....	22
4.2 性能概述.....	25
5. 系统开发对社会的影响.....	27
6. 设计总结.....	28
6.1 完成的工作.....	28
6.2 系统设计感悟.....	28
6.2.1 代码版本管理.....	28
6.2.2 多变的用户需求.....	29
6.3 改进方案.....	29
6.3.1 远程调试改进.....	29
6.3.2 多车管理支持.....	30
6.3.3 通信架构改进.....	30

6.3.4 用户体验改进.....	30
参考文献.....	31

1. 引言

1.1 课题研究的背景及意义

随着 5G 和车联网等新技术的逐步应用使自动驾驶技术日臻完善，为了更好的发展自动驾驶技术，对其人机交互系统的研究也逐渐成为国内外相关企业的关注重点。以目前来说，现有的自动驾驶人机交互系统都或多或少的存在拓展性不够、通用性不强、即使对开发者而言都不算友好的用户体验等现状导致我们无法直接使用这些现成的设施。于是这也说明研究一个适合用户和开发者的自动驾驶人机交互系统，并提供一个相对好用的实现的必要性。

1.2 国内外研究及发展现状

从 20 世纪 50 年代初，国外就开始了自动驾驶相关技术的研究，并在 80 年代得到了高速发展。其中美国是最早研究自动驾驶技术、技术水平最高且市场化发展最好的国家。由于自动驾驶技术的广泛应用前景，特别是在军事应用领域的巨大潜力，极大地促进了世界各国特别是西方发达国家的研发激情。除美国外，英国和德国也是较早一批进行自动驾驶方面的研究并取得突破发展的国家。我国自动驾驶汽车方面的技术研究起步的时间相对较晚，从 20 世纪 80 年代左右才正式开始。

在自动驾驶的民用方面，美国的谷歌公司走在了世界的前列，并且从一开始就把研发方向定位于全自动驾驶，而不是大多数自动驾驶汽车厂商所采用的智能辅助驾驶。近年来，美国的特斯拉公司也在自家的电动汽车上内置了自动驾驶系统，提升了电动汽车的智能化水平，不过特斯拉汽车的自动驾驶系统不能完全替代自动驾驶操作，即使开启自动辅助驾驶，仍然需要驾驶者时刻保持警惕状态。我国民用无人车项目起步较早的是百度公司，由百度研究院进行主体研发，其核心是一项被称为“百度汽车大脑”的无人驾驶技术，具有高精度导航地图、环境感知、智能决策和控制四个主要模块。

由于自动驾驶汽车发展时间较短，目前还处于技术研发阶段，国内外主要致力于自动驾驶汽车技术的研究，对于人机交互界面的研究相对较少。目前一部分自动驾驶汽车使用已有车型进行改造，通过安装所需的传感器和控制系统以实现自动驾驶功能，因此车内人机交互界面和原有汽车较为相似。虽然通过改装其他车型获得自动驾驶功能的方式能够较快地测试原型车了，然而这种功能重组的方式势必会破坏原本车型驾驶室内的人机交互界面央视，也不是专门针对自动驾驶场景进行设计的，因此无法评估出哪些因素会影响自动驾驶汽车的交互体验。随着自动驾驶技术的不断提升，设计出针对自动驾驶场景的人机交

互界面方案是自动驾驶技术发展的必然要求。

现阶段车载系统和智能自动驾驶的界面设计研究主要集中在工程技术层面，而从人机交互体验角度出发的研究较少，缺乏充分的可用性调研及标准，在设计时主要依靠设计师自身的经验，同时参考发展较为成熟的计算机/网页交互设计。互联网、云计算的发展使生活中各种产品设备作为移动终端成为移动互联网的一部分，软件系统嵌入到更多产品中，这些产品兼备计算机交互和自身传统交互特性，其设计过程有更多限制挑战，需要设计者对产品使用场景有更深的体会了解。交互设计发展应该以人为本。未来，随着智能网联时代的全面到来，智能汽车将覆盖更多适用人群，交互方式也将不断创新，人工智能正在向情感智能方向发展，具备更强的识别情感、理解情感、表达情感能力，多维度、多通道、个性化的智能情感交互设计成为趋势。

1.3 本文的主要工作与结构

为了更好的设计一个同时适合用户和开发者，且能够在现实项目中投入使用的自动驾驶人机交互系统，作者首先设计了一个与自动驾驶算法框架相结合的能够在“智小蜂”无人清扫车这样的真实硬件上运行的车联网平台，接着作者在此基础上进行本课题所描述的自动驾驶人机交互系统的开发。

本文首先对整套系统开发主要使用到的硬件、其内置的相关传感器和本文使用的编程技术进行基本的介绍。然后对本文工作所涉及到的车联网平台和自动驾驶人机交互系统的架构设计进行详细介绍，以及构造这些系统的时候遇到的问题和对应的解决方案，例如：在 Nvidia Jetson AGX Xavier 这样的性能有限的嵌入式设备上运行本系统所需要进行的性能方面的优化调整，车联网平台硬件抽象层的设计，自动驾驶人机交互系统的方案选型等。最后会讨论作者在开发过程中遇到的一些麻烦事和对应的一些思考和总结、还有作者的心得、系统的不足和未来的一些改进方案。

2. 主要技术介绍

2.1 硬件环境概要

2.1.1 车体和控制器

本课题所使用的硬件设备是一辆线控车，具体外型与产品参数如下所示：

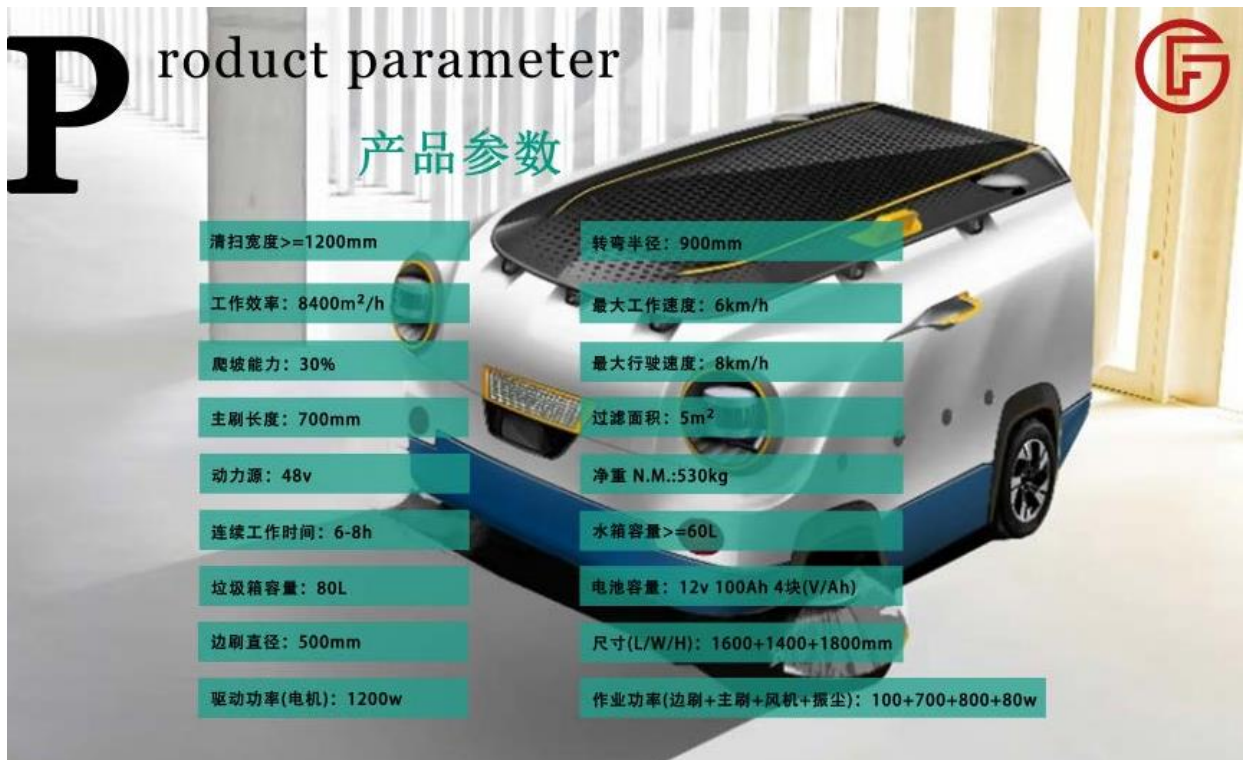


图 2.1.1 线控车产品参数

控制器通过 CAN 协议实现被车载计算单元控制，控制协议如下所示：

作用	标准报文 ID	报文内容
以 0.1 km/h 前进	0x0233	0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x11
以 2 km/h 后退	0x0233	0x00 0x00 0x00 0x00 0x00 0x14 0x00 0x21
转向复位	0x0223	0x00 0x00 0x00 0x00 0x00 0x00 0x04 0x01
向左转 90 度	0x0223	0x00 0x00 0x00 0x00 0x00 0xA6 0x03 0x01
向右转 90 度	0x0223	0x00 0x00 0x00 0x00 0x00 0x5A 0x04 0x01
开启扫盘	0x0203	0x00 0x00 0x00 0x00 0x08 0x00 0x00 0x00
升起扫盘	0x0203	0x00 0x00 0x00 0x00 0x14 0x00 0x00 0x00
制动开启	0x0213	0xFF 0x00 0x00 0x00 0x00 0x00 0x00 0x01
制动解除	0x0213	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01

图 2.1.2 线控车控制协议示例

控制器通过周立功 USB CAN 适配器与车载计算单元连接，外形与具体参数如下所示



隔离耐压支持	2500V
阻燃绝缘挡板	有
CAN 通道数量	2
操作系统支持	Windows 和 Linux

图 2.1.3 CAN 适配器外形与具体参数

2.1.2 车载传感器

本课题所使用的车载传感器主要有雾化喷洒器、GNSS、雷达、摄像头、工控机、扫盘等。

车载传感器的具体放置位置如下所示：

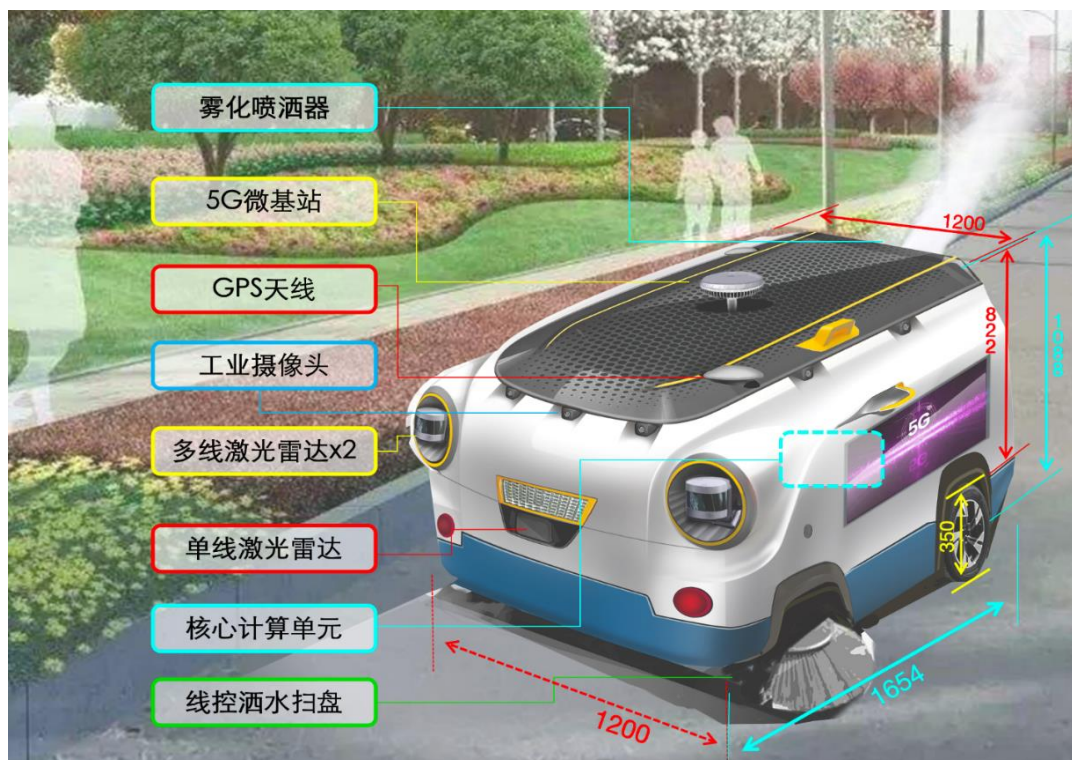


图2. 1. 4 车载传感器具体放置位置

车载传感器可实现的功能如下所示：

实现功能 传感器	障碍物检测	前向碰撞	行人碰撞	紧急制动	路径规划
1个前置摄像头	√				
1个前置摄像头 2个激光雷达	√	√	√	√	
1个前置摄像头 2个后置摄像头 2个激光雷达 1个毫米波雷达	√	√	√	√	√

2.1.5 车载传感器可实现的功能

2.1.3 车载计算单元

本课题采用的车载计算单元为 Nvidia Jetson AGX Xavier，外形和产品参数如下所示：



	Jetson AGX Xavier
GPU	512 核 Volta GPU（具有 64 个 Tensor 核心） 11 TFLOPS (FP16) 22 TOPS (INT8)
DL 加速器	(2x) NVDLA 引擎 5 TFLOPS (FP16) 10 TOPS (INT8)
CPU	8 核 ARM v8.2 64 位 CPU、8 MB L2 + 4MB L3
内存	32GB 256 位 LPDDR4x 2133MHz - 137GB/s
显示接口	三个多模式 DP 1.2/eDP 1.4/HDMI 2.0
存储空间	32GB eMMC 5.1

视觉加速器	7 通道 VLIW 视觉处理器
视频编码	8x 4K @ 30 (HEVC)
视频解码	12x 4K @ 30 (HEVC)
摄像头	16 通道 MIPI CSI-2, 8 通道 SLVS-EC D-PHY (40 Gbps) C-PHY (109 Gbps)
UPHY	3x USB 3.1、4x USB 2.0 1 x8 或 1 x4 或 1 x2 或 2 x1 PCIe (Gen4)
其他	UART、SPI、CAN、I2C、I2S、DMIC、GPIO
连接	10/100/1000 RGMII
尺寸	100 mm x 87 mm
机械特性	699 针连接器 一体式传热板

图 2.1.5 车载计算单元的外形和产品参数

2.2 软件环境概要

2.2.1 ROS 与 C++

ROS 即机器人操作系统（Robot Operating System）的英文缩写，是用于编写机器人软件的一种具有高度灵活性的软件框架。它包含了大量工具软件、库代码和通信协议约定，旨在简化创建复杂、鲁棒的机器人行为这一过程的难度与复杂度。

ROS 提供了它提供了操作系统应有的服务，包括硬件抽象，底层设备控制，常用函数的实现，进程间消息传递，以及包管理。它也提供用于获取、编译、编写、和跨计算机运行代码所需的工具和库函数。

本课题使用 ROS 实现车联网部分的开发，实现对车辆的实际控制和反馈、传感器信息的获取和反馈和为自动驾驶算法框架和人机交互体验部分提供强有力的支撑。

ROS 支持 C++ 和 Python，考虑到本课题对该部分的运行效率和实时性要求，本课题选择 C++ 作为该部分的开发用编程语言。

该部分的摄像头服务考虑到在车载计算单元运行的效率和尽可能复用 ROS 的上游依赖，遂采用 OpenCV 和 CUDA 的组合。

该部分使用 JetBrains 公司开发的跨平台 C++ 集成开发环境 CLion 进行代码编写和调试。

2.2.2 .NET Core 与 C#

.NET Core 是一个跨平台的支持多种编程语言的开发框架，由微软公司出品，目前主要包括 ASP.NET Core、Windows Forms、WPF。 .NET Core 提供对 Windows、macOS 和

Linux 的支持，开发者可以通过微软公司开发的 Visual Studio 和 Visual Studio Code 进行基于 .NET Core 的应用开发。

本课题使用 ASP.NET Core 实现人机交互部分的开发，实现对车辆运行状态的可视化并实现对车辆的基本控制。

.NET Core 支持 C#、VB.NET、F# 等多种编程语言，考虑到代码的可读性和可维护性，本课题选用 C# 作为人机交互部分的 Web 服务器的开发用编程语言。

在 Web 浏览器显示的部分，本课题会充分利用以 HTML5、CSS3 和 JavaScript 为主的现代 Web 技术进行开发，并且采用响应式界面布局以实现支持跨设备的媲美原生应用的用户体验。

本课题所使用的 ASP.NET Core 推荐使用微软公司开发的 SignalR 开源库实现 Web 实时通信，其会自动选择服务器和客户端的最佳传输方式，其选择的优先级为 WebSocket、Server-Sent Events 和长轮询。于是这也是本课题所选用的在 Web 浏览器与 Web 服务器之间的基本通信方案之一。

由于本课题也使用了 ROS，于是采用基于 ROS Bridge 协议的 ROS# 开源库 ROS 部分的通信。

该部分使用微软公司开发的 Visual Studio 和 Visual Studio Code 进行代码编写和调试。

3. 系统设计与实现

3.1 系统框架结构

本课题自下而上地将项目主体分为车联网和人机交互两大部分，结构图如下所示：

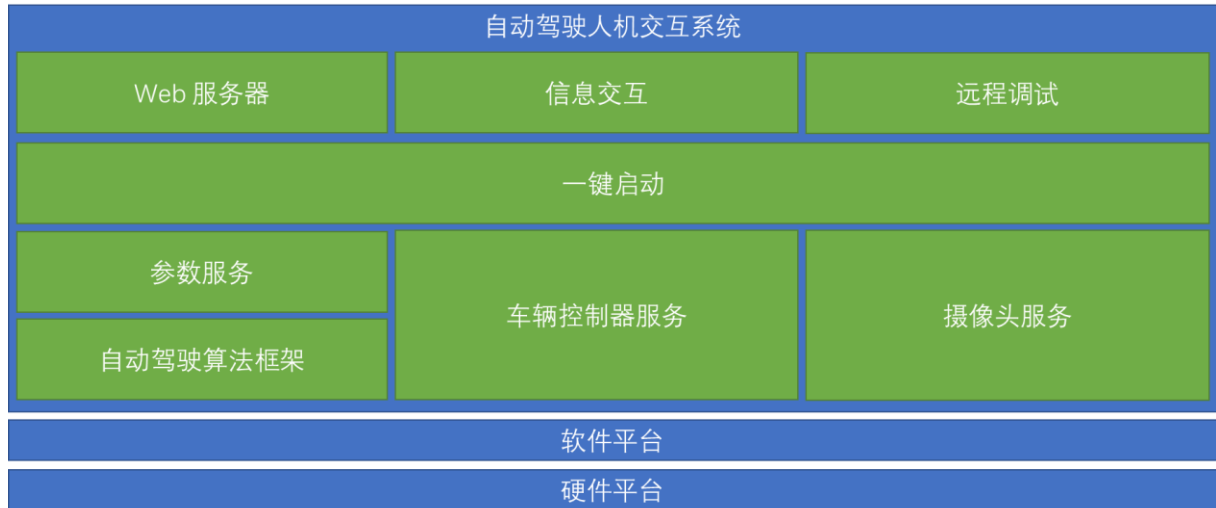


图 3.1.1 课题系统框架结构图

本课题现阶段基于 Wi-Fi 进行手持终端和车载计算单元的通信，基本通信示意图如下所示：

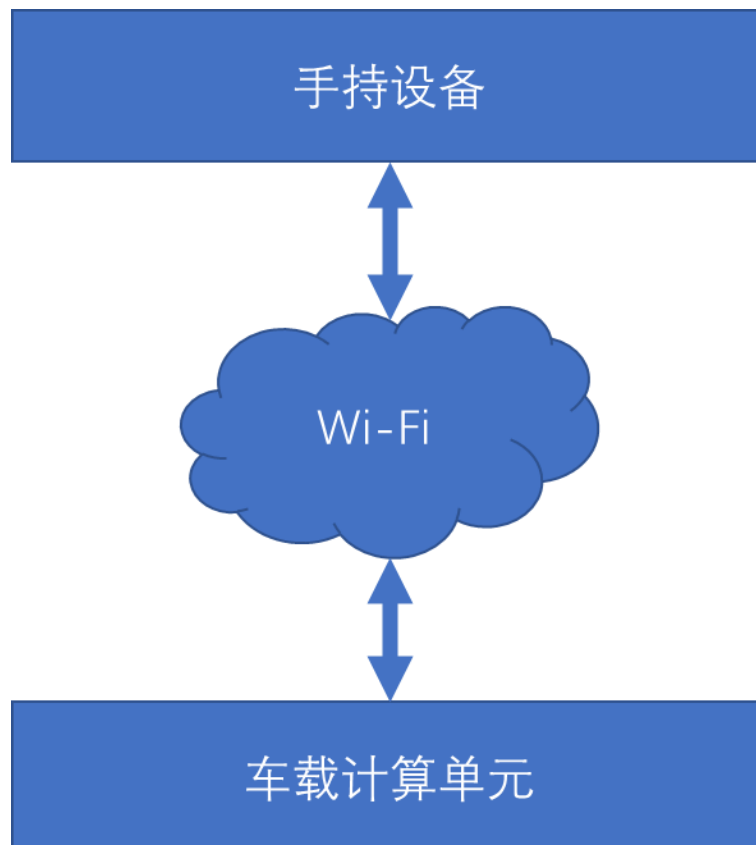


图 3.1.2 课题系统通信方式示意图

3.2 开发环境

3.2.1 硬件平台

在本文的 2.1 节硬件环境概要部分的基础上做以下补充：

1、由于本课题采用的车载计算单元性能有限且基于 ARM64 处理器，为了更好更快速的开发本课题所指系统，本课题采用基于 x86 处理器的工控机进行系统原型开发。

2、非系统原型开发阶段，则完全使用本课题所采用的车载计算单元。

3、进行系统原型开发的基于 x86 处理器的工控机配置为 Intel Xeon E3-1275 V5、Nvidia Geforce GTX 1080、32GB 物理内存和 256GB 固态硬盘。

3.2.2 软件平台

在本文的 2.2 节软件环境概要部分的基础上作以下补充：

1、车联网部分基于 Ubuntu 18.04 和 ROS Melodic，使用 Clion 2020.1 进行开发，需要单独周立功 USB CAN 适配器和速腾聚创激光雷达驱动包。

2、人机交互部分基于 ASP.NET Core 3.1，使用 Visual Studio 2019 进行开发。

3、采用 Git 客户端和本地 Git 仓库进行源代码版本管理。

3.3 车辆控制器服务实现

车辆控制器服务是本课题项目的核心部分之一，基本框架结构图如下所示：

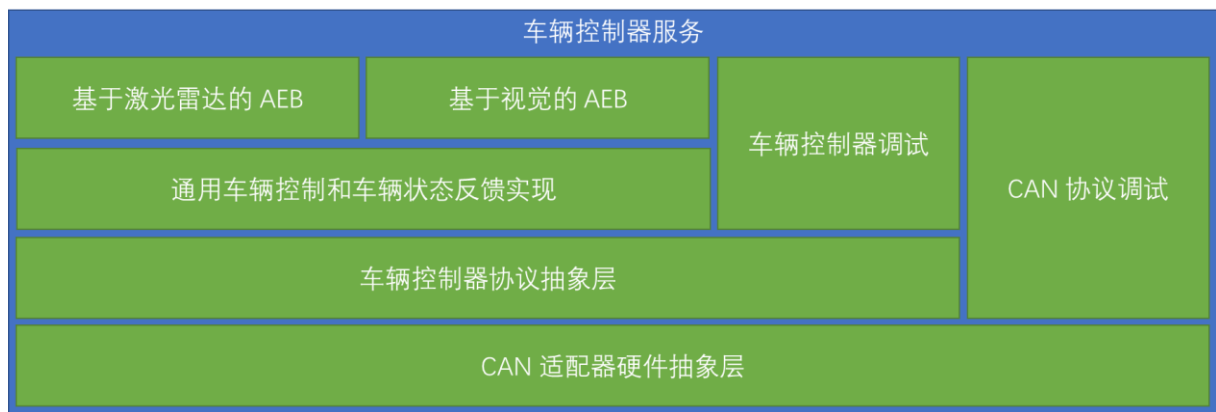


图 3.3.1 车辆控制器服务基本框架结构图

本课题的车辆控制器服务提供以下三大特性：

1、对 CAN 适配器驱动包进行封装，在车辆控制器服务内部提供通用的 CAN 适配器抽象接口，并为开发者提供基本的 CAN 协议调试功能。

2、对线控车控制器协议进行封装，基于 CAN 适配器抽象接口实现，在车辆控制器服务内部提供通用的车辆控制器抽象接口，并为开发者提供基本的车辆控制器调试功能。

3、在通用的车辆控制器抽象接口的基础上，与自动驾驶算法框架进行交互，实现对车辆的控制、车辆状态的获取、基于视觉和激光雷达的在紧急情况下自动制动（即 AEB）的

功能。

由于受到了 Microsoft COM 设计风格的影响，所以车辆控制器服务抽象层的接口都继承自共同的基类，定义如下所示：

```
class base_interface
{
public:
    virtual void release() = 0;
};
```

图 3.3.2 车辆控制器服务抽象层基类接口定义

3.3.1 CAN 适配器硬件抽象层

这是车辆控制器服务的最底层，主要作用是对不同种类的 CAN 适配器的操作实现封装成统一的抽象接口，主要目的是减少适配不同种类 CAN 适配器的难度和方便其他模块调用。该抽象层接口定义如下所示：

```
class can_adapter_interface : public base_interface
{
public:
    virtual void send_data_frame(
        bool is_standard_frame,
        std::uint32_t frame_id,
        std::vector<std::uint8_t> const& frame_data) = 0;
    virtual void receive_data_frames(
        std::map<std::uint32_t, std::vector<std::uint8_t>>& frames) = 0;
};
```

图 3.3.3 车辆控制器服务 CAN 适配器硬件抽象层接口定义

由图可知，CAN 适配器硬件抽象层接口为发送和接收数据帧提供了统一的抽象接口，在保证接口简单易懂的前提下覆盖了绝大部分情况下的 CAN 适配器使用场景。

3.3.2 CAN 协议调试

CAN 协议调试能够大幅减少自动驾驶相关开发者适配新车协议的难度，由于车辆控制器服务基于 ROS 环境开发，于是采用 ROS Message 作为交互方式，这样既方便通过 ROS Bridge 和 ROS# 与人机交互部分进行通信，同时还方便开发者直接使用 rostopic 命令行工具直接进行调试。

其基本上是 CAN 适配器硬件抽象层接口定义对应的 ROS Message 版本，对外暴露的 ROS Topic 的作用如下所示：

ROS Topic	作用
/cslg_vehicle_service/can_sender	向 CAN 适配器发送报文
/cslg_vehicle_service/can_receiver	从 CAN 适配器接收报文

图 3.3.4 CAN 协议调试对外暴露的 ROS Topic

CAN 协议调试对外暴露的 ROS Topic 对应的 ROS Message 定义如下所示：

类型: cs1g_vehicle_service/can_frame

uint32 id

uint8[] data

类型: cs1g_vehicle_service/can_sender_message

ROS Topic: /cs1g_vehicle_service/can_sender

bool is_standard_frame

can_frame frame

类型: cs1g_vehicle_service/can_receiver_message

ROS Topic: /cs1g_vehicle_service/can_receiver

can_frame[] frames

图 3.3.5 CAN 协议调试对外暴露的 ROS Topic 对应的 ROS Message 定义

3.3.3 车辆控制器协议抽象层

这是车辆控制器服务的中层部分，主要作用是对不同种类的车辆协议实现封装成一个统一的抽象接口，主要目的是减少适配不同种类车辆协议的难度和方便其他模块调用。该抽象层接口定义如下所示：

```
enum class vehicle_gear_type
{
    neutral,
    drive,
    reverse
};

class vehicle_status_information
{
public:
    vehicle_gear_type gear = vehicle_gear_type::neutral;
    double steering_angle = 0.0;
    double velocity = 0.0;
    double acceleration_pedal = 0.0;
    double break_pedal = 0.0;
    std::map<std::string, bool> switches;
};

class vehicle_controller_interface : public base_interface
{
public:
    virtual void can_adapter_receive_handler(
        std::map<std::uint32_t, std::vector<std::uint8_t>>& frames) = 0;
    virtual vehicle_status_information get_vehicle_status() = 0;
```



```

virtual void set_gear(vehicle_gear_type value) = 0;
virtual void set_steering_angle(double value) = 0;
virtual void set_velocity(double value) = 0;
virtual void set_acceleration_pedal(double value) = 0;
virtual void set_break_pedal(double value) = 0;
virtual void set_switch(std::string name, bool value) = 0;
};

```

图 3.3.6 车辆控制器服务车辆控制器协议抽象层接口定义

由图可知，车辆控制其协议抽象层接口为控制车辆和获取车辆反馈信息提供了统一的抽象接口。为了能够满足可以在未来使用同样的接口适配不同种类的车辆，车辆控制其协议抽象层接口仅把挡位（gear）、转向角（steering_angle）、速度（velocity）、加速踏板（acceleration_pedal）和制动踏板（break_pedal）这些基本上所有种类的车辆都有的特性在接口中单独出来；关于其他部分的控制，譬如车辆灯光、喇叭、扫盘的开启和关闭，皆归类在物件开关（switch）中，内部使用键值对（std::map）实现，以相对简单实用的方式为以后的拓展性打下了基础。

3.3.4 车辆控制器调试

车辆控制器调试有两大主要作用：既为了减少自动驾驶相关开发者适配新车协议的难度，又给人机交互部分的车辆信息展示提供了接口。所以和 CAN 协议调试接口一样，车辆控制器调试接口同样也采用了 ROS Message 作为交互方式。

对外暴露的 ROS Topic 的作用如下所示：

ROS Topic	作用
/cslg_vehicle_service/vehicle_status	是否启动车辆
/cslg_vehicle_service/gear_command	挡位控制
/cslg_vehicle_service/steering_angle_command	转向角控制
/cslg_vehicle_service/velocity_command	速度控制
/cslg_vehicle_service/acceleration_pedal_command	加速踏板控制
/cslg_vehicle_service/break_pedal_command	刹车踏板控制
/cslg_vehicle_service/switch_command	物件开关控制
/cslg_vehicle_service/vehicle_receiver	车辆状态反馈

图 3.3.7 车辆控制器调试对外暴露的 ROS Topic

对应的 ROS Message 定义如下所示：

```

类型：cslg_vehicle_service/vehicle_status
ROS Topic: cslg_vehicle_service/vehicle_status

bool value

类型：cslg_vehicle_service/gear_message
ROS Topic: cslg_vehicle_service/gear_command

```

```

uint8 neutral = 0
uint8 drive = 1
uint8 reverse = 2
uint8 value

类型: cs1g_vehicle_service/steering_angle_message
ROS Topic: cs1g_vehicle_service/steering_angle_command

float64 value

类型: cs1g_vehicle_service/velocity_message
ROS Topic: cs1g_vehicle_service/velocity_command

float64 value

类型: cs1g_vehicle_service/acceleration_pedal_message
ROS Topic: cs1g_vehicle_service/acceleration_pedal_command

float64 value

类型: cs1g_vehicle_service/break_pedal_message
ROS Topic: cs1g_vehicle_service/break_pedal_command

float64 value

类型: cs1g_vehicle_service/switch_message
ROS Topic: cs1g_vehicle_service/switch_command

string name
bool value

类型: cs1g_vehicle_service/vehicle_receiver_message
ROS Topic: cs1g_vehicle_service/vehicle_receiver

gear_message gear
steering_angle_message steering_angle
velocity_message velocity
acceleration_pedal_message acceleration_pedal
break_pedal_message break_pedal
switch_message[] switches

```

图 3.3.8 车辆控制器调试的 ROS Topic 对应的 ROS Message 定义

3.3.5 通用控制部分

通用控制部分的作用主要包括：

- 1、为自动驾驶算法框架提供车辆控制和车辆状态反馈的支持。

2、提供基于激光雷达和视觉的 AEB 支持。

第一个作用主要是通过解析自动驾驶算法框架发布的 ROS Topic，使得自动驾驶算法框架能够顺利的控制车辆；通过发布自动驾驶算法框架所需要的车辆信息的 ROS Topic，使得自动驾驶算法框架能够顺利的获取车辆状态的反馈以实现车辆的闭环控制，从而改善车辆的行驶质量。

第二个作用的基于激光雷达的 AEB 支持是通过自动驾驶算法框架获取通过激光雷达测定的离车辆最近的障碍物距离，当满足阈值时则进入 AEB 模式使得车辆自动停下，为了减少反复刹车的情况，进入 AEB 模式后则会等待 3 秒，等待完毕后再开始新一轮的判断；基于视觉的 AEB 支持也是类似的原理，区别在于基于视觉的 AEB 是通过对下文即将描述的摄像头服务获取畸变矫正后的图像使用测距算法获取离车辆最近的障碍物距离；通过基于激光雷达和视觉的 AEB 支持双管齐下，实现了当车辆在前方遇到行人和障碍物的情况下令其安全的停车等待。

3.4 摄像头服务实现

摄像头服务是本课题项目的核心部分之一，基本流程图如下所示：

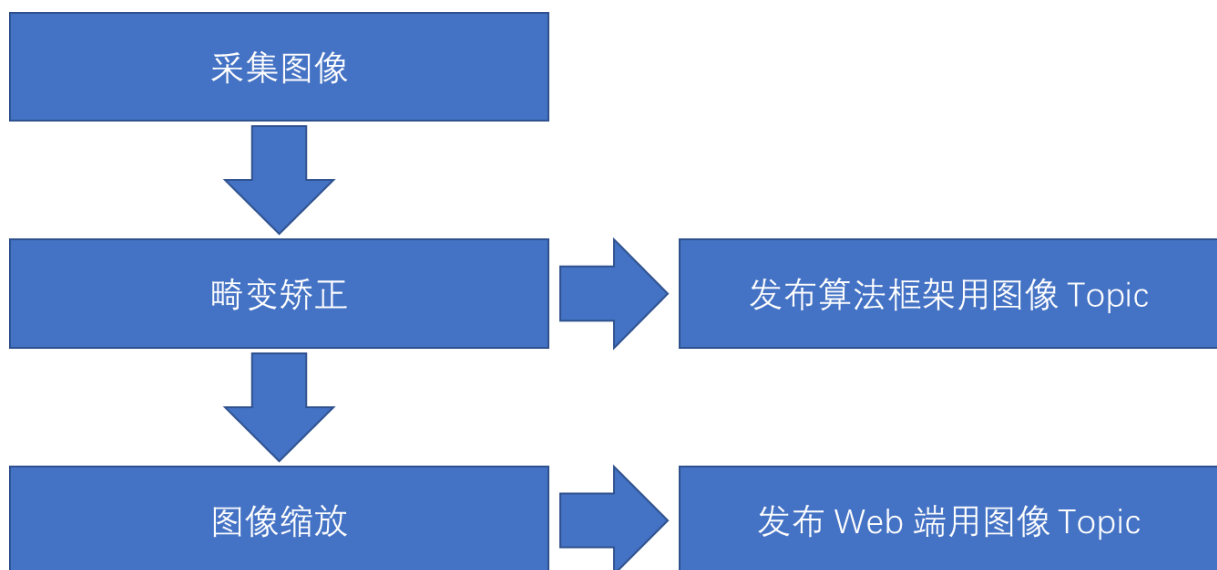


图 3.4.1 摄像头服务基本流程图

由流程图可知，本课题的摄像头服务提供以下特性：

- 1、从摄像头采集图像。
- 2、对图像进行畸变矫正。
- 3、对图像进行缩放。
- 4、输出算法框架用图像。
- 5、输出 Web 端用图像。

3.4.1 图像采集

图像采集是摄像头服务的根本，是一切操作的基础。

由于本课题采用的摄像头在 Linux 下是无需安装额外驱动的，于是我们只需要通过 OpenCV 库获取摄像头当前图像帧并保存到矩阵对象中以方便后续操作。核心实现如下所示：

```
cv::VideoCapture capture;  
cv::Mat raw_frame;  
  
capture.open(0);  
capture.read(raw_frame);
```

图 3.4.2 图像采集核心实现

为了提升图像采集、摄像头服务、自动驾驶算法框架乃至整个系统的可靠性，本课题的实际实现会加入是否成功获取图像的相关判断和失败时重新开启摄像头的实现。这样做的额外好处是，在本课题系统下，车载摄像头支持热插拔和在一键启动时发生摄像头故障时解决问题的灵活性。

3.4.2 畸变矫正

基本上只要是和图像处理有关的任务，第一件事情必然是对原始图像做畸变矫正。本项目把畸变矫正直接集成入摄像头服务的目的有两点：

1、既然都要做畸变矫正，那么集成进去可以免除一部分内部通信的开销而且也更好做优化，当然由于减少了一个专用的畸变矫正进程，内存开销大幅降低。

2、在 Web 端显示畸变矫正后的图像不会有鱼眼效果，观感上更加符合实际情况。

畸变矫正部分主要根据摄像头标定参数矩阵以使用 OpenCV 的图像重映射方法实现。

由于这部分占用的 CPU 资源比较多，于是为了优化这部分的性能，使用 OpenCV 的 CUDA 部分对其进行加速，使得摄像头服务在车载计算单元上的 CPU 占用从 20% 降低到了 7%。

3.4.3 图像缩放

为了能够减少在 Web 上显示图像的传输负担，对畸变矫正后的图像进行缩放成为了必须要做的事情。

本项目将图像的宽度和高度分别都缩为原始图像对应参数的二分之一，按照比例缩放的好处是图像不会变形，缩为二分之一主要是清晰度和传输负担的考虑。

由于这部分的主要通过 OpenCV 的缩放方法实现，且该方法可以使用 OpenCV 的 CUDA 部分进行加速，为了尽可能腾出车载计算单元宝贵的 CPU 资源，所以这部分也是通过 CUDA 加速的。

3.4.4 发布图像

由于自动驾驶算法框架对图像的质量有要求，所以在原始图像做畸变矫正之后立马发布算法框架用图像 ROS Topic。当然这部分的图像没有压缩，原因是自动驾驶算法框架只支持原始位图。

在发布 Web 端用图像 ROS Topic 前，为了进一步减轻传输压力，我们将缩放过的图像采用 JPEG 算法进行压缩，质量设为在实际场景中看起来不会过分失真且体积足够小的 20%，毕竟在 Web 端显示的图像并不需要太高的质量，但需要一定的实时性。

发布图像原本应该采用 ROS 的 cv_bridge 库实现，但由于该库和启用了 CUDA 的 OpenCV 库存在冲突，因其依赖系统内置的不支持 GPU 加速的 OpenCV 库。所以本课题的摄像头服务自行实现了一份，这样做的额外好处是减去了一个依赖，使得摄像头服务可以完全不依赖 ROS 提供的额外基本设施，方便在非 ROS 环境下的代码复用。

3.5 一键启动实现

3.5.1 自动驾驶算法框架

由于自动驾驶算法框架实质上是一个个 ROS 节点，需要对这部分每一个模块都使用 ROS Launch 配置文件实现这些模块的自动加载。最后使用一个 ROS Launch 配置调用每一个模块对应的 ROS Launch 配置进行自动驾驶算法框架的整体自动化加载，这也是一键启动实现的基础设施之一。

为了能够在以后更好的维护自动驾驶算法框架所使用的参数，于是这部分还引入了参数服务来管理自动驾驶算法框架的参数。

3.5.2 一键启动脚本

由于已经提供可以加载整个自动驾驶算法框架的 ROS Launch 配置。在此基础上继续加入车辆控制器服务、摄像头服务、传感器和人机交互部分的 ROS Launch 配置并提供一个 ROS Launch 配置以实现整个系统的总体加载。

为了能在开机时自动调用，于是使用加载整个系统的 ROS Launch 配置的 shell 脚本与 Ubuntu 提供的 upstart 开机自启动设施进行对接。

3.6 Web 服务器实现

3.6.1 Web 服务器

因为人机交互采用现代 Web 技术实现，所以这也是本课题系统人机交互部分的基础。该部分主要基于非常成熟的且支持跨平台的 ASP.NET Core 开源框架且采用 C# 编写以保证开发效率和系统整体的可靠性。

3.6.2 Web 通信

本课题采用多种协议进行人机交互 Web 客户端的 Web 浏览器和人机交互 Web 服务器的通信，采用的方案如下所示：

1、由于车联网部分基于 ROS 环境实现，于是需要借助 ROS Bridge 与 ROS# 以 WebSocket 协议实现车联网部分和人机交互 Web 服务器的基础通信。

2、信息交互部分由于部分通信实现需要在 Web 客户端实现，这部分则主要借助 ROS Bridge 的 JavaScript 客户端库以 WebSocket 协议实现人机交互 Web 客户端和车联网部分的通信。

3、信息交互部分的其余实现则通过 SignalR 和 ASP.NET Core Post 请求处理程序使得人机交互 Web 服务器能够与 Web 浏览器进行信息交换。

4、车载摄像头内容从人机交互 Web 服务器到运行人机交互 Web 客户端的 Web 浏览器的传输则采用 HTTP 协议的 multipart/x-mixed-replace 机制实现。

3.6.3 人机交互 Web 服务器到 Web 客户端的车载摄像头内容传输

本部分采用 HTTP 协议的 multipart/x-mixed-replace 机制，使得 Web 浏览器的 Image 控件可以自动根据发送的内容更新图像，因此实现了非常简单的车载摄像头内容串流。其基本原理可以用发送的报文来描述，该部分如下所示：

```
Get /CameraImage HTTP/1.1\r\n
...
multipart/x-mixed-replace; boundary= CameraFrame\r\n
...
\r\n
--CameraFrame\r\n
Content-Type: image/jpeg\r\n
\r\n
（第 1 帧图像二进制内容）
\r\n
--CameraFrame\r\n
Content-Type: image/jpeg\r\n
\r\n
（第 2 帧图像二进制内容）
\r\n
...
--CameraFrame\r\n
Content-Type: image/jpeg\r\n
\r\n
（传输结束前最后一帧图像二进制内容）
\r\n
```

图 3.6.1 车载摄像头内容传输报文结构

3.7 信息交互实现

本课题主要采用现代 Web 技术以网页的方式通过 Web 浏览器进行人与车辆之间的信息交互，其主要提供以下功能：

- 1、远程调试，该部分会在后续的远程调试实现部分提到。
- 2、车辆信息展示。
- 3、车辆当前位置和一键召唤。
- 4、车辆基本控制

该部分主要采用 HTML5、CSS3、JavaScript ES6 这样的现代 Web 技术实现，使用 bootstrap 框架实现信息交互的用户界面框架，使用 WebGL 在 Web 浏览器上呈现车辆可以展示的信息。

由于人机交互部分采用响应式的界面布局，因此提供了非常友好的跨设备用户体验，譬如用户可以在手机和平板电脑上访问本课题系统的 Web 后台优雅地对车辆进行控制。

3.7.1 车辆信息展示

该部分主要采用 bootstrap 框架和 WebGL 接口实现，支持呈现以下内容：

- 1、车辆的基本信息，譬如当前挡位、当前转向角、当前速度、当前加速踏板百分比、当前刹车踏板百分比、车载其他开关的开启状态等。
- 2、车载激光雷达的实时点云数据，并以 RGB 方式表示激光雷达点云数据中每个点的反射强度。
- 3、点云地图，高精度向量图、规划好的路线和车辆所在规划路线的位置。
- 4、车载摄像头图像内容展示。

3.7.2 车辆当前位置和一键召唤

该部分支持显示车辆当前在现实地图中的位置，本课题采用高德地图作为现实地图，结合车载 GNSS 接收器获取到的位置，实现了在现实中车辆位置的准确显示，这也是实现一键召唤的基础。

一键召唤是一个可以让用户随时随地召唤线控车以节约时间成本和提升便携程度的对用户而言非常贴心的功能，一键召唤实现主要包含在车联网部分的自动驾驶算法框架和人机交互部分的信息交互实现。

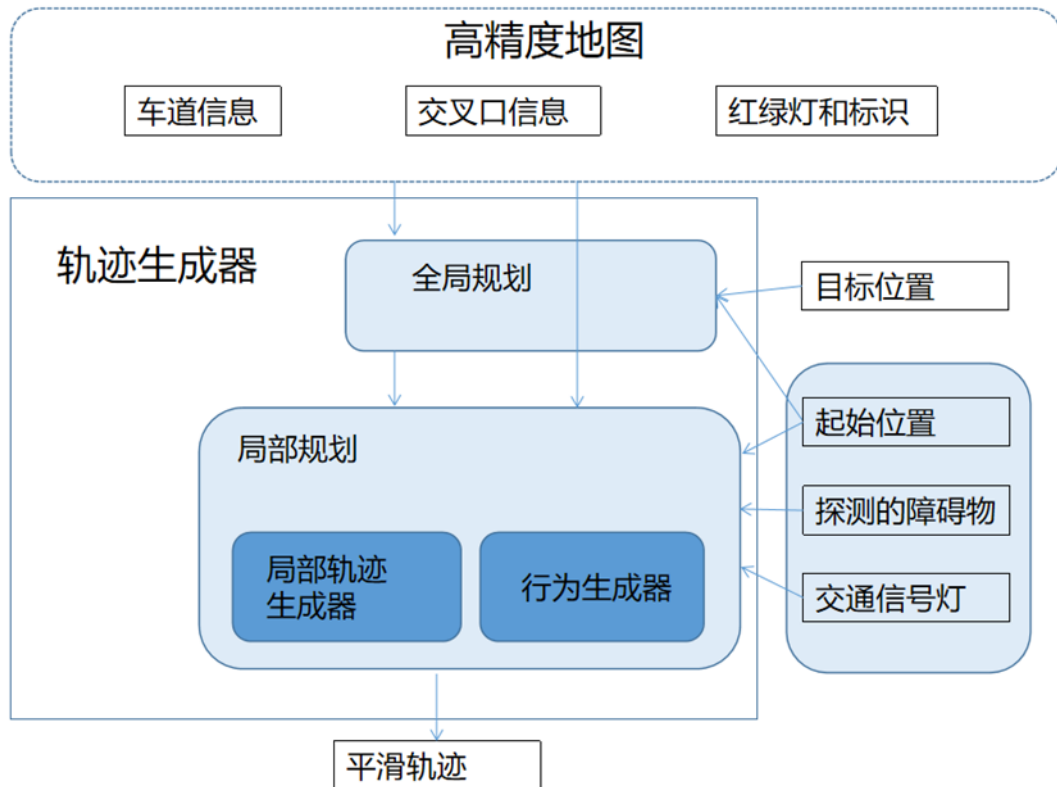


图 3.7.1 自动驾驶算法框架的一键召唤功能架构图

由图可知，在自动驾驶算法框架中，一键召唤分成路径规划和行为生成两个模块。

路径规划模块中，为了使生成的路径更加精准且所用成本更低，本课题的自动驾驶算法框架采用了全局规划与局部规划相结合的方式：首先系统中包含了一个全局规划器。在这个全局规划器的基础上，本课题将会提供矢量地图。用户只需要将起始位置和目的位置输入，此系统就会使用动态编程以距离成本为约束找到从起点到地图上目标点的全局参考路径。然后，系统使用此全局参考路径和当前位置为输入，就会生成一组无障碍的局部轨迹，对局部轨迹进行处理之后从中选择总体成本最低的轨迹。

在行为生成模块，为了使线控车行驶更加安全，对周围的障碍物检测，交通信号灯检测更加敏锐，本课题采用行为生成器使用预先定义的交通规则和传感器数据来充当协调器，使用碰撞和交通规则成本计算以及最佳轨迹的选择来进行速度轮廓分析和考虑是否需要重新进行命令。

在信息交互实现中，一键召唤功能提供在线实时地图用来显示车辆的位置，在地图信息控件下方提供一个下拉框为用户提供可以进行一键召唤的候选点列表。用户可以根据实际情况选择其中一个候选点使得车辆顺利的自动行驶到指定的位置。

3.7.3 车辆基本控制

该部分主要提供以下特性：

- 1、车辆的启动与停止。
- 2、基于挡位、转向角度、速度的参照车辆摄像头实时内容的远程控制。
- 3、部分车载设备的开启和关闭，譬如清扫车的扫盘、消毒车的紫外线灯等。

3.8 远程调试实现

远程调试实现在本课题中起到了兜底作用，其作用如下所示：

- 1、使得本课题可以在绝大部分场景下都能通过本课题所描述的通信方式控制车载计算单元，同时也令本课题所设计系统更加完备。
- 2、大大方便开发者对线控车的调试，无需趴在线控车上看线控车内置的屏幕，仅需要优雅的拿出笔记本，通过 Wi-Fi 连接到车载计算单元即可坐在离车较远的阴凉处进行调试。

3.8.1 远程调试服务器

本课题采用久经考验的 `x11vnc` 作为远程调试服务器实现，其对车载计算单元的使用 X11 图形服务器完美适配，且在传输时可以根据画面内容实现动态压缩以进一步减小在使用远程调试功能时的传输压力。

由于 Linux 图形设施在未插入显示器的情况下不会启用 GPU，于是本课题使用 HDMI 虚拟显示器解决这个问题。由于使用了没有屏幕分辨率限制的虚拟显示器，于是开发者可以自由的调整屏幕分辨率使得远程调试更加方便。以下是 HDMI 虚拟显示器的实物外观：



图 3.8.1 HDMI 虚拟显示器实物

3.8.2 远程调试 Web 客户端

本部分采用久经多年考验的 noVNC 库,并使用现代 Web 技术构建 UI,使得在 Web 浏览器上对车载计算单元进行远程调试成为可能。

远程调试 Web 客户端通过 WebSocket 和 Web 服务器进行通信,Web 服务器通过下文所介绍的反向代理与远程调试服务器进行通信。

3.8.3 反向代理

由于 x11vnc 不支持直接通过 WebSocket 进行通信,于是需要将 WebSocket 转换成 x11vnc 支持的 TCP Socket 使得远程调试 Web 客户端顺利的远程调试服务器进行通信。

这样的转换方式称作反向代理,反向代理服务即外部网络客户端向内部服务器发出请求,即接收来自 Internet 上用户的连接请求,并将这些请求转发给内部网络上的服务器,然后将从内部服务器上得到的响应返回给 Internet 上请求连接的客户。

虽然 noVNC 提供了反向代理的官方实现即 Websockify,由于 Websockify 是一个使用 Python 实现的脚本,而且只因为反向代理而多创建一个进程和多暴露一个通信端口并不是一个明智的选择,于是这里本部分以 ASP.NET Core 中间件的方式实现一个 Websockify 的替代品,这样既可以少创建一个进程、少暴露一个通信端口降低了车载计算单元的 CPU 负载而且还间接为之后提升安全性而做准备。

4. 系统测试

4.1 运行截图



图 4.1.1 登录界面



图 4.1.2 主界面

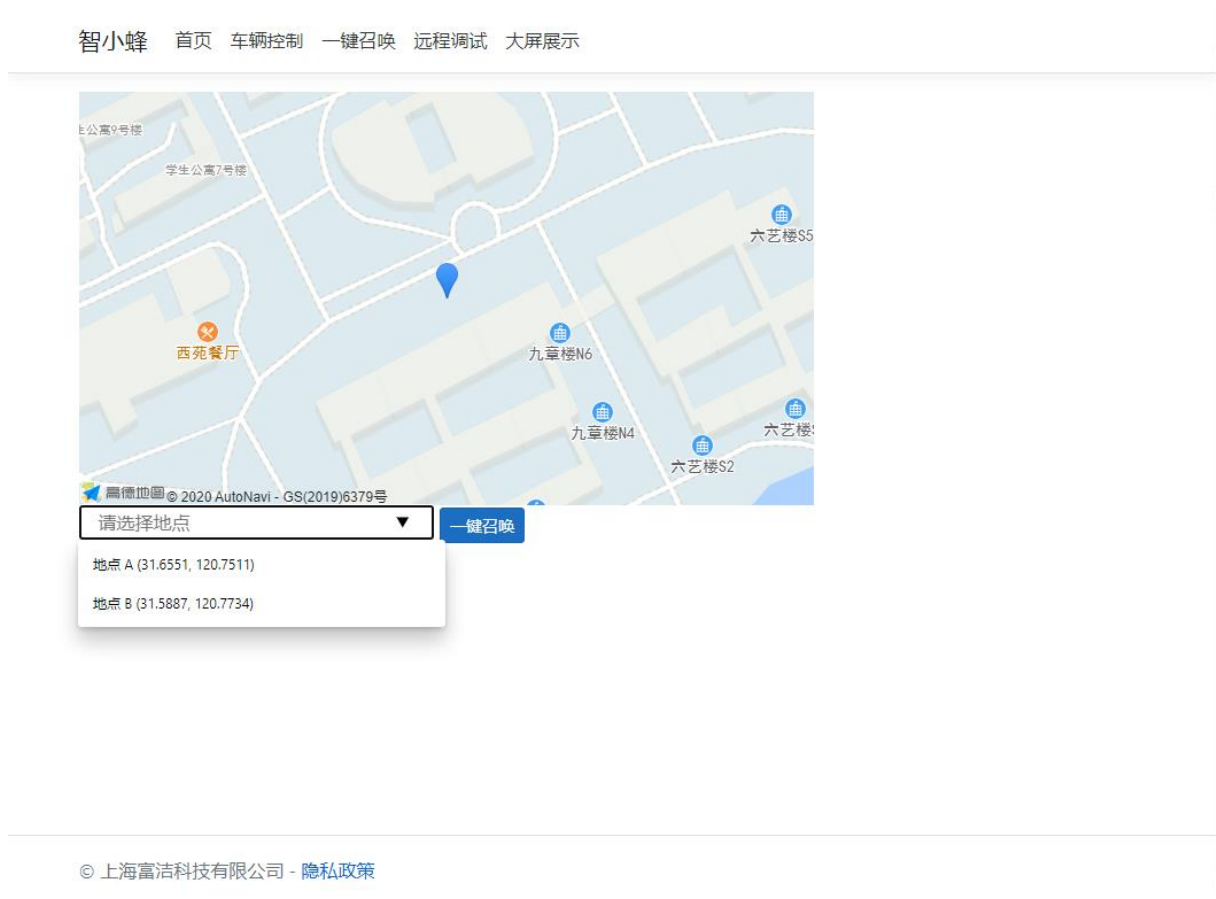


图 4.1.3 一键召唤界面

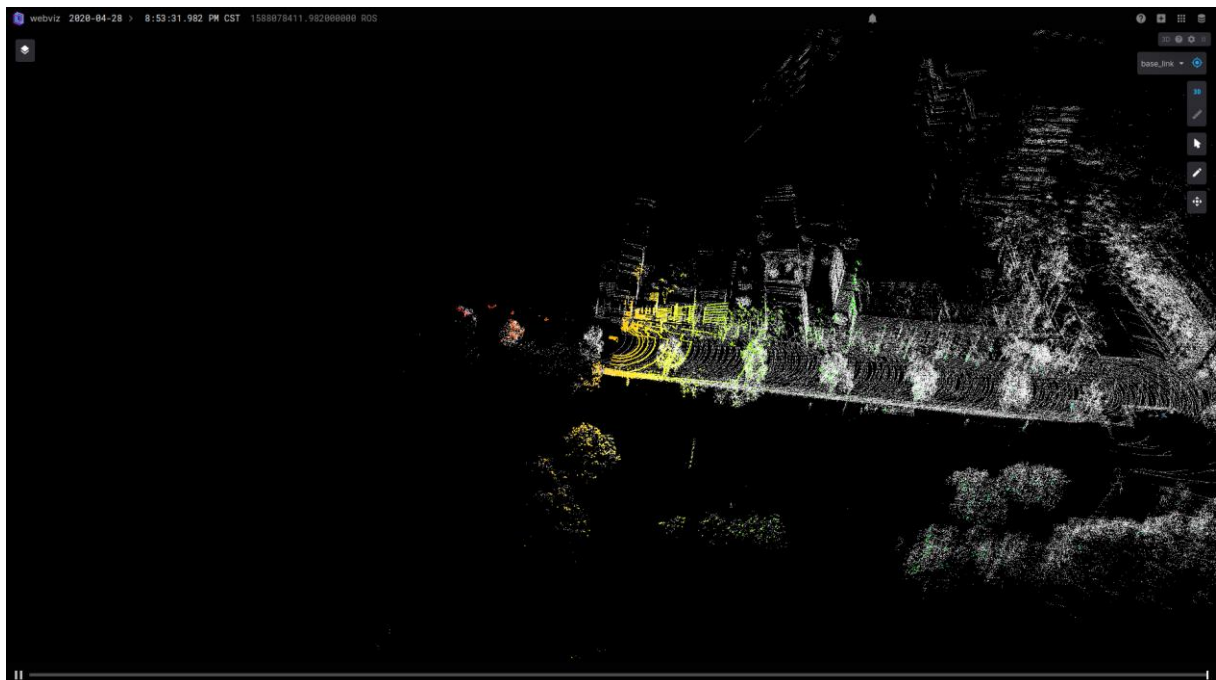


图 4.1.4 大屏展示界面

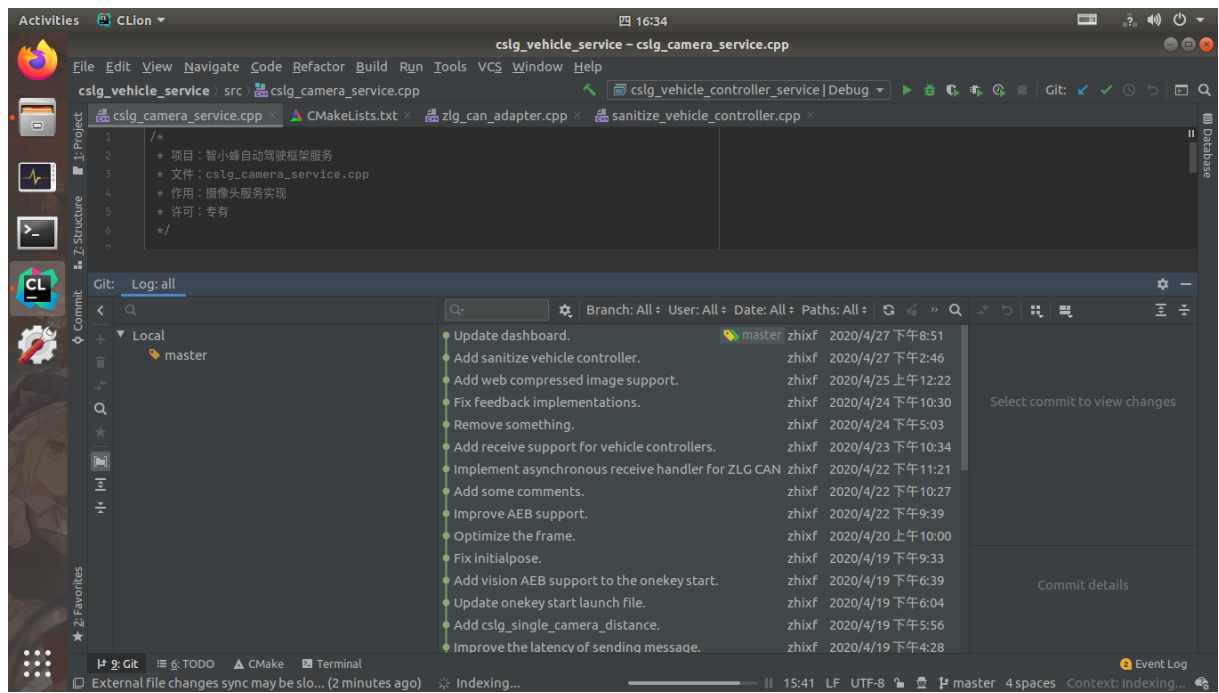


图 4.1.5 远程调试界面



图 4.1.6 线控车调试实况



挡位:	N 挡
转向角度 (负数为右, 正数为左):	71
速度 (km/h):	0.9
加速踏板开度 (从 0 到 1):	0
刹车踏板开度 (从 0 到 1):	0
扫盘:	关闭
AEB:	关闭

图 4.1.7 车辆控制界面

4.2 性能概述

在本课题使用的车载计算单元下运行本课题系统，在开启全部模块的情况下，CPU 占用保持在 30% 左右，GPU 占用保持在 70% 左右，不仅使得全系统能在性能有限的车载计算单元上流畅运行，而且还有大量的计算资源可以用作调试和后续功能的添加。

本课题系统的视觉部分可以保证以 15 帧每秒的速率处理图像，可以保证视觉部分能够正常运行和保持系统整体的响应速度。

本系统在实时传输车载摄像头内容的时候，只需要使用 4 Mbps 的网络带宽就可以实

现在 Web 端以 60 帧每秒的速率显示。

本系统在进行远程调试的时候在保证用户流畅体验的情况下只需要使用 20 Mbps 的网络带宽。

5. 系统开发对社会的影响

本系统对社会的影响主要有：

1. 该系统可以大大降低包括自动驾驶在内的智能网联汽车的开发门槛，可以大大加快开发进度，因此可以更快的创造新的工作岗位可以缓解当前的就业压力。
2. 在有限场景下实行自动驾驶可以降低因驾驶员失误引起的交通事故的概率。现阶段的技术、法律和伦理等基础暂时还不够支撑全场景下的自动驾驶。
3. 远程驾驶可以降低包括清理泥石流、塌方抢修、扑灭森林大火等工作的危险性，而且相对自动驾驶来说，在技术、法律、伦理等方面的障碍要少很多。本系统可以推进远程驾驶相关技术的开发。
4. 国内需要建立非常健全的智能网联汽车的隐私保护和事故认定等相关事项的法律条文。

6. 设计总结

6.1 完成的工作

本文实现了一个可以投入实际使用的自动驾驶人机交互系统。

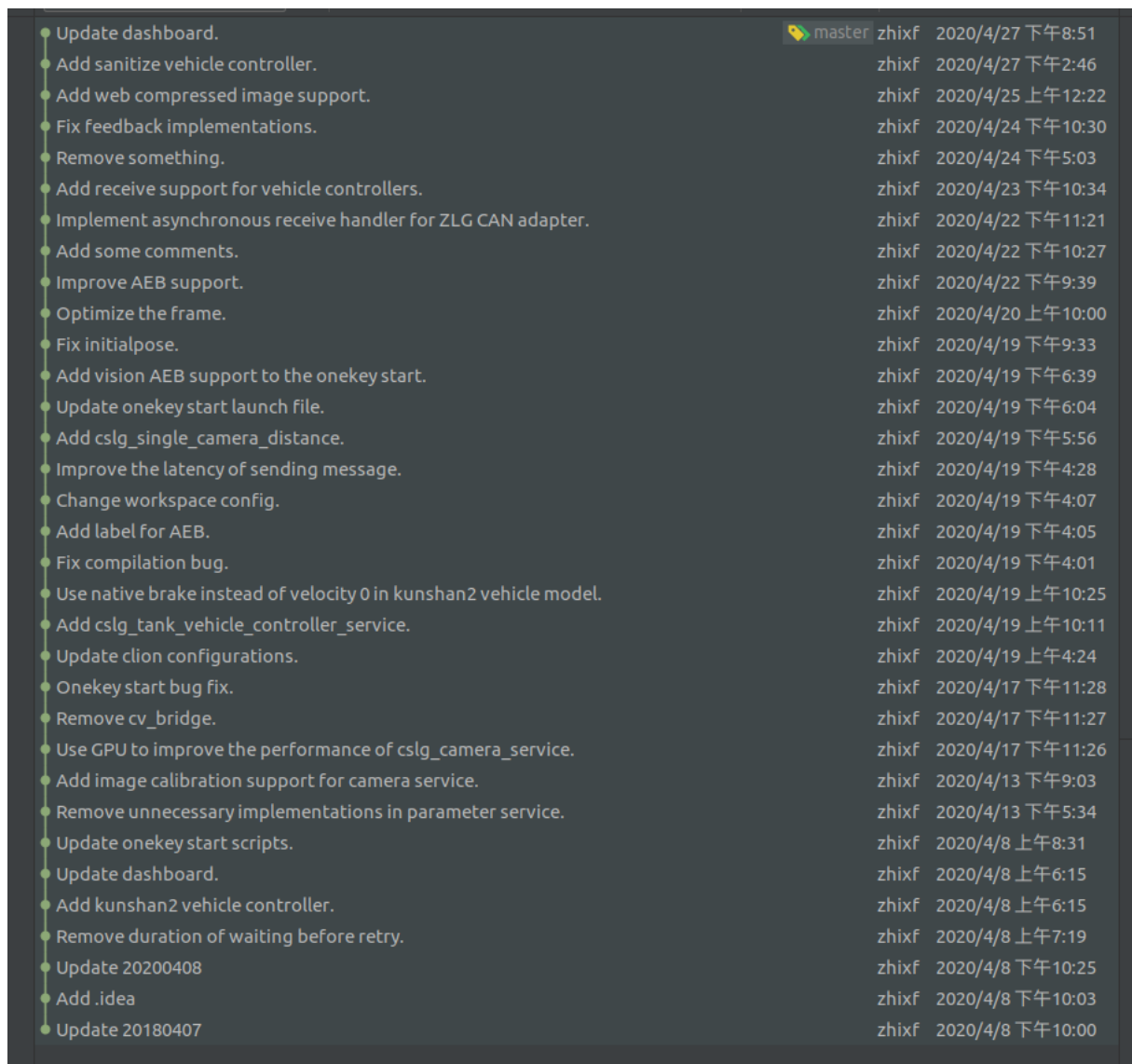
其在 45W 的 arm 低功耗嵌入式设备上实现同类项目在 480 W 的 x86 高性能工业控制用工作站的效果，而且提供了一个还算好用的用户体验方案。

6.2 系统设计感悟

6.2.1 代码版本管理

关于管理这样一个高复杂度的系统的代码开发，引入代码版本管理系统变得尤为重要。

本课题采用 Git 作为代码版本管理系统，本课题对于系统的部分开发历史如下所示：



● Update dashboard.	zhixf	2020/4/27 下午8:51
● Add sanitize vehicle controller.	zhixf	2020/4/27 下午2:46
● Add web compressed image support.	zhixf	2020/4/25 上午12:22
● Fix feedback implementations.	zhixf	2020/4/24 下午10:30
● Remove something.	zhixf	2020/4/24 下午5:03
● Add receive support for vehicle controllers.	zhixf	2020/4/23 下午10:34
● Implement asynchronous receive handler for ZLG CAN adapter.	zhixf	2020/4/22 下午11:21
● Add some comments.	zhixf	2020/4/22 下午10:27
● Improve AEB support.	zhixf	2020/4/22 下午9:39
● Optimize the frame.	zhixf	2020/4/20 上午10:00
● Fix initialpose.	zhixf	2020/4/19 下午9:33
● Add vision AEB support to the onekey start.	zhixf	2020/4/19 下午6:39
● Update onekey start launch file.	zhixf	2020/4/19 下午6:04
● Add cs1g_single_camera_distance.	zhixf	2020/4/19 下午5:56
● Improve the latency of sending message.	zhixf	2020/4/19 下午4:28
● Change workspace config.	zhixf	2020/4/19 下午4:07
● Add label for AEB.	zhixf	2020/4/19 下午4:05
● Fix compilation bug.	zhixf	2020/4/19 下午4:01
● Use native brake instead of velocity 0 in kunshan2 vehicle model.	zhixf	2020/4/19 上午10:25
● Add cs1g_tank_vehicle_controller_service.	zhixf	2020/4/19 上午10:11
● Update clien configurations.	zhixf	2020/4/19 上午4:24
● Onekey start bug fix.	zhixf	2020/4/17 下午11:28
● Remove cv_bridge.	zhixf	2020/4/17 下午11:27
● Use GPU to improve the performance of cs1g_camera_service.	zhixf	2020/4/17 下午11:26
● Add image calibration support for camera service.	zhixf	2020/4/13 下午9:03
● Remove unnecessary implementations in parameter service.	zhixf	2020/4/13 下午5:34
● Update onekey start scripts.	zhixf	2020/4/8 上午8:31
● Update dashboard.	zhixf	2020/4/8 上午6:15
● Add kunshan2 vehicle controller.	zhixf	2020/4/8 上午6:15
● Remove duration of waiting before retry.	zhixf	2020/4/8 上午7:19
● Update 20200408	zhixf	2020/4/8 下午10:25
● Add .idea	zhixf	2020/4/8 下午10:03
● Update 20180407	zhixf	2020/4/8 下午10:00

图 6.2.1 车联网部分代码版本历史

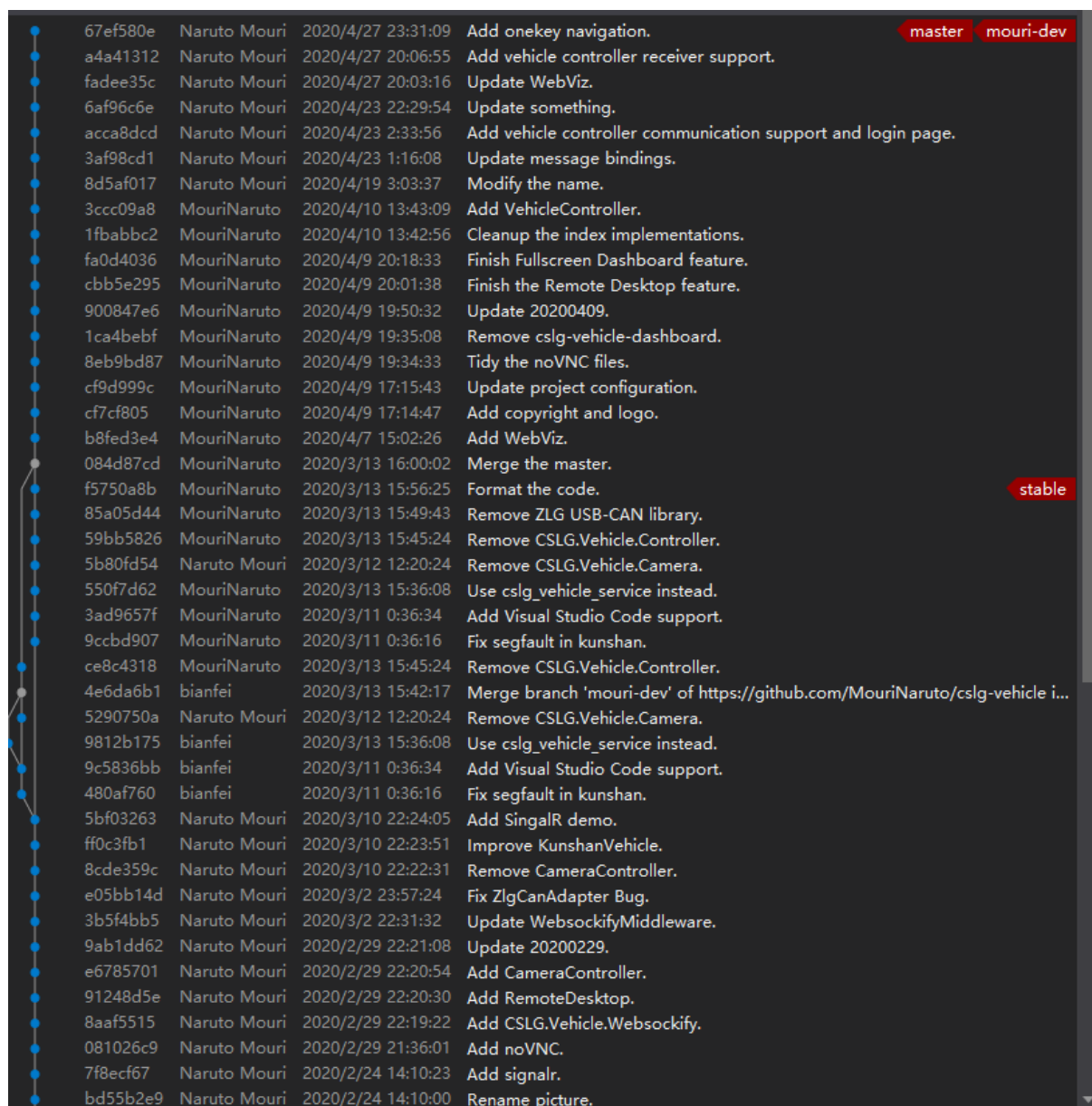


图 6.2.2 人机交互部分代码版本历史

6.2.2 多变的用户需求

在参与项目开发的过程中，由于用户需求进行了多次调整。譬如添加一些新功能、新的车辆控制器的支持等。对此表示整个项目的可拓展性变得相当重要，因此对项目进行合理的分层，每一层通过抽象好的接口进行通信是非常必要的。

在这种情况下，提供一个好用且灵活的调试机制也非常重要。

6.3 改进方案

由于本课题的开发时间有限且因为新冠肺炎疫情的影响，还有一些特性未能添加。下文会简要的说一下改进方案以弥补相应的遗憾。

6.3.1 远程调试改进

虽然采用了 x11vnc 这样的成熟方案作为远程调试服务器，但其稳定性不尽如人意，

在连续使用远程调试功能一天以上很有可能会导致崩溃。

对于此，更换远程调试服务器方案可以缓解问题，譬如采用 TigerVNC Server。

6.3.2 多车管理支持

本课题系统现阶段只能管理一辆车，因此需要引入中央服务器等硬件设备和对软件进行大幅度改进以支持对多车协同管理。

6.3.3 通信架构改进

为了尽可能降低延迟，车载计算单元和中央服务器之间可以使用基于 UDP 的协议。譬如 QUIC，这是 HTTP/3 的底层传输协议，且已经存在靠谱的跨平台开源实现。

由于车载计算单元向中央服务器传输的数据量很大且内容稀疏，可以在传输过程中利用 GZIP、LZ4 等高效且压缩率高的压缩算法进行压缩。

为了增加车辆网各模块的吞吐量、大幅减少内存拷贝次数、提升响应速度和大幅减少内存占用，可以学习百度的 Apollo 框架使用共享内存机制替换掉基于 HTTP XML 的 ROS 通信机制。

Web 服务器和客户端的图像传输实现可以使用更加成熟的 WebRTC 协议实现。

6.3.4 用户体验改进

本课题的界面还可以进行大幅改善，可以采用谷歌 Fuchsia OS 项目的支持跨设备体验且支持生成 Web 页面的 Flutter 界面框架进行开发，以实现更好的支持跨设备的用户体验。

引入性能强悍的中央服务器后，可以采用服务端渲染机制减轻客户端呈现界面的性能和带宽负担。

参考文献

- [1]黎兰平,郭修远.自动驾驶汽车车外人机交互界面设计研究[J].包装工程,2020,41(02):57-64.
- [2]张大权,张惠,杨冬雨,赵默涵.关于 L3 自动驾驶的人机交互设计研究[J].汽车文摘,2020(01):32-35.
- [3]张诗墨,暴宏志,杜才贞.自动驾驶工况下汽车虚拟人人机交互探索[J].汽车电器,2019(06):11-13.
- [4]王海丰. 无人驾驶电动汽车驾驶室内的人机交互界面设计[D].东南大学,2017.
- [5]谭浩,孙家豪,关岱松,周茉莉,齐健平,赵颖.智能汽车人机交互发展趋势研究[J].包装工程,2019,40(20):32-42.
- [6]方慧君. 结合 HMI 的高度自动化智能汽车的多维度造型设计[D].清华大学,2016.
- [7]孙勇义.Apollo 自动驾驶技术与安全设计[J].人工智能,2018(06):48-58.
- [8]于悦. 车载导航装置人机交互系统研发[D].吉林大学,2007.
- [9]杨文灵. 基于情境意识的智能汽车人机交互设计研究[D].湖南大学,2017.
- [10]李谟秧. 汽车人机交互界面用户自定义手势设计研究与应用[D].湖南大学,2015.
- [11]何珂言,郁淑聪,孟健.智能网联时代的汽车交互设计[J].汽车与配件,2019(15):78-79.
- [12]宋蒙,刘琪,许幸荣,王题.C-V2X 技术在智能网联行业中的应用探讨[J].中兴通讯技术:1-6[2020-02-29].
- [13]2020 中国(深圳)智能网联与新能源汽车技术展[J].中国电子商情(基础电子),2020(Z1):67.
- [14]中国信息通信研究院. 车联网白皮书[R].2019.
- [15]杨宏.车联网/智能网联汽车技术与标准研究[R].中国电子技术标准化研究院.[2019-10-23].