学　号：	0122204950320

武汉理工大学

课 程 设 计

题　　目	测控系统软件课程设计
学　　院	机电工程学院
专　　业	测控技术与仪器
班　　级	测控 2201
姓　　名	林鸿伟
指导教师	周晓

2024 年 **2** 月 **20** 日

课程设计任务书

学生姓名：林鸿伟　　专业班级：测控 2201

指导教师：周晓　　工作单位：机电学院测控系

题　目：智能驾驶舱软件系统设计

一、应用背景

各大汽车制造商和科技公司都在竞相研发智能驾驶技术，以实现更加安全、高效、舒适的出行体验。不久的将来，数以万计的智能汽车在街道上飞驰。这些车辆不再是简单的交通工具，而是一台台具备高度自主能力的“移动智能终端”，不仅能够感知周围环境，还能够实时分析海量数据，进行决策和规划。

“测控”在其中扮演着至关重要的角色。每辆智能汽车都配备有多种传感器，包括摄像头、激光雷达、毫米波雷达等。这些传感器不断地采集车辆周围环境数据，如道路状况、其他车辆的位置和速度、行人等，并实时传输到数据处理中心。高性能的计算机集群对数据进行处理和分析，通过机器学习和深度学习识别关键信息，结合高精度地图数据和交通流量数据，预测交通情况，为车辆提供最优的行驶路径。作为工程师，您将有机会参与这个激动人心的领域，为未来智能交通贡献自己的力量。通过运用 C++面向对象的思维和方法，您将设计开发高效、稳定的驾驶舱软件系统，满足智能驾驶的需求。在这个过程中，您将不断学习和实践，提升自己的编程能力，为未来智能驾驶时代创造更多的可能性。

二、功能需求

必做功能：

1. 建立适合智能驾驶舱内不同传感器的对象模型，完成类设计。每个传感器都有其独特的数据格式和传输方式。抽象类应包含初始化、启动、停止、获取数据等方法，以便在程序实现时对不同传感器的统一管理。
2. 基于智能驾驶舱内的显示屏，设计一个友好的软件界面。界面应至少包括数据展示区、采集控制按钮（启动、停止、暂停、继续等）和智能驾驶云端中心服务器的推送信息显示。
3. 通过应用程序模拟各类型传感器模块工作，生成含有随机误差的测量值。这些应用程序应能根据界面指令完成测量，并通过网络上传到云中心服务器。需要设计者了解各类型传感器的数据生成原理，并通过编程实现对这些数据的精确模拟和传输。注意，所采集的数据不仅驾驶舱要用，云中心服务器也需要。
4. 数据读写功能。测量数据以 CSV 格式保存到驾驶舱文件中。同时支持从文件中读入到系统并显示。用户可以根据时间、传感器种类等设定查询条件，实现对大量数据的快速检索和分析。
5. 支持接收中心服务器信息。驾驶舱能够接收中心服务器下发的报警、路径推荐、路况等信息，在驾驶舱中进行显示。

选做功能：

6. 在完成以上指定功能基础上，可以通过查阅最新研究成果，自由发挥并扩展软件功能，以提升系统的智能性。如允许多个驾驶员之间进行数据共享和交流；如在智能驾驶舱系统中加入 AI 分析功能，利用机器学习算法，建立异常检测模型，以自动预测危险，通知驾驶员及早采取相应措施；通过 AI 分析驾驶员的行为模式和习惯，建立行为分析模型，以实现对驾驶行为的评估和预测，如是否处于疲劳状态；结合车辆行驶数据，建立风险评估模型，对车辆行驶风险进行综合评估；结合动态路况，建立路径优化模型，提供最优的行驶路径建议。

三、设计要求

（1） 要求独立完成设计内容。

（2） 根据设计任务要求，查找有关文献资料，充分论证比较，确定方案，绘制功能模块和工作流程图。要有必要的设计图和文字说明，并能满足评分细则给出的各项要求。

（3） 编写并调试完整的程序代码

（4） 按课程设计说明书撰写规范，提交一份课程设计说明书（6000 字左右），设计说明书应详细说明设计思路、工作流程、实现界面及核心代码。

（5） 设计书中必须有明确的章节分析，如下：

a) 分析方案中相关的社会、健康、安全、法律、文化及环境等制约因素；

b) 设计方案中需要关注环境与可持续发展问题；

c) 列举相关的环境标准、规范，论证方案符合环保要求；

d) 阐述相关的经济决策，说明项目管理方法；

e) 要有明确的不少于 3 篇的参考文献分析过程。

四、进度安排

1 布置任务、理解需求、文献查阅： 1 天

2 软件总体方案设计：2 天

3 代码编写及运行调试：4 天

4 撰写设计报告、答辩：3 天

指导教师签名： 年 月 日

系主任（或责任教师）签名： 年 月 日

本科生课程设计成绩评定表

姓　名	林鸿伟	学号	0122204950320
课程设计题目：智能驾驶舱软件系统设计			
课程设计答辩或质疑记录：			

序号	目标	评　价　内　容	分值	得分
1	2.1	明确设计目标，任务需求，分析实现过程中可能存在的多种问题，了解影响方案的各种因素，提出合理的解决方案，并有效解决。	20	
2	2.2	阐述相关的经济决策，说明项目管理方法，合理分工并确定进度计划，设计符合经济决策和项目管理的原理。	10	
3	1.1	设计方案论证要考虑社会、健康、安全、法律、文化及环境等制约因素，设计方案中需要关注环境与可持续发展问题。	10	
4	1.2	能根据任务学习新知识、新技术，设计新功能并应用于设计中。	10	
5	3.1	完成了数据采集、传输，分析，存储，显示等功能。	10	
6	3.2	完成了代码调试，实现了所有设计功能，运行结果正确。	10	
7	3.3	流程框图规范，图文详实，说明书符合规范要求。	20	
8	4.1	进行口头报告，态度诚恳，交流顺畅，能正确回答老师的提问。	10	
合计			100	

其它事项说明			
评阅教师（签名）		评阅时间	年　　月　　日

目录

第 1 章 任务背景分析

1.1 研究背景

工信部、公安部、自然资源部、住建部、交通部五部联合发布“车路云一体化”应用试点工作的通知，旨在推动智能网联汽车技术的发展和应用。试点工作将在全国范围内展开，试点期为 2024 至 2026 年。通过实施智能网联汽车“车路云一体化”应用试点，加速相关基础设施建设，探索多因素协同，促进智能驾驶软件的多场景应用，加快智能网联汽车技术的突破和产业化发展，非常符合本项目的应用场景。


中华人民共和国中央人民政府
www.gov.cn
首页 | 简 | 繁 | EN | 登录 | 邮箱 | 无障碍
首页 > 政策 > 国务院政策文件库 > 国务院部门文件 字号：默认 大 超大 | 打印 收藏 留言
标 题：五部委关于开展智能网联汽车“车路云一体化”应用试点工作的通知
发文机关：工业和信息化部 公安部 自然资源部 住房城乡建设部 交通运输部
发文字号：工信部联通装〔2023〕268号
来 源：工业和信息化部网站
主题分类：工业、交通\其他
公文种类：通知
成文日期：2024年01月15日
工业和信息化部 公安部 自然资源部 住房和城乡建设部 交通运输部关于开展智能网联汽车“车路云一体化”应用试点工作的通知
工信部联通装〔2023〕268号
各省、自治区、直辖市及新疆生产建设兵团工业和信息化主管部门、公安厅（局）、自然资源主管部门、住房和城乡建设厅（局、委）、交通运输厅（局、委）：
为贯彻落实《新能源汽车产业发展规划（2021—2035年）》（国办发〔2020〕39号），推动网联云控基础设施建设，探索基于车、路、网、云、图等高效协同的自动驾驶技术多场景应用，加快智能网联汽车技术突破和产业化发展，工业和信息化部、公安部、自然资源部、住房城乡建设部、交通运输部（以下统称五部门）联合开展智能网联汽车“车路云一体化”应用试点工作，试点期为2024—2026年。有关事项通知如下：

图 1-1 工信部等五部联合发布“车路云一体化”应用试点工作的通知

1.2 项目分析

在国家车联网产业标准体系建设指南（智能网联汽车）（2023 版）中明确指出要建设标准智能网联汽车体系框架。其中，智能驾驶软件系统[1]是一种综合了人工智能、计算机视觉、状态机、高精度地图、雷达和定位系统等多种组件的先进车载电脑系统。这一系统利用雷达、摄像头等各类传感器来感知周围环境，并将所获取的信息作为输入信号传递给车载计算机和服务器。本项目应用计算机视觉，并结合机器学习实现 MDS 驾驶员检测，基于 Matplotlib 库实现数据可视化，利用改良服务器实现 V2X 车车通信。

第 2 章 方案设计

2.1 需求分析

根据以上的功能需求结合国家车联网产业标准体系建设指南（智能网联汽车）（2023 版）与工信部等五部联合发布“车路云一体化”应用试点工作的通知，得出以下需求。

2.1.1 传感器设计

传感器需要设计适用于智能驾驶舱内不同传感器的对象模型，并完成相应的类设计，且每个传感器都具有不同的数据格式和传输方式，为了实现统一管理，抽象类需要包含初始化、启动、停止等功能。应用程序还需要模拟各类型传感器模块的工作，生成包含随机误差的测量值，当传感器通过测量得到数据结果时，通过串口将数据传输给智能驾驶舱的控制面板并显示。

在实现过程中需要实现传感器的准确切换，每一个传感器所获取的数据不同，因此要在子类传感器中定义专属的数据储存变量，并在传输数据时正确配对数据的格式与单位。

2.1.2 数据的读写与查询

智能驾驶舱通过串口获取传感器传输的数据，并以 csv 格式将数据进行保存到驾驶舱文件中。同时支持从文件中读入到系统中并显示。用户能够通过车辆用户名、传感器种类、传感器所测数值范围以及时间等设定查询条件，实现对大量数据的快速检索和分析。并支持以可视化的方式显示在驾驶舱客户端，帮助客户快速找到异常信息，数据通过网络传输到中心服务器中。

由于保存时间的字符串存在“：”需要使用 stringstream 实现分割，并转化为秒，实现时间查询。

2.1.3 网络传输与数据分析

智能驾驶舱客户端支持和云中心服务器通过 TCP 进行网络通信，实现对传感器信息的获取、分析以及对驾驶舱的预警。驾驶舱同时能够接收中心服务器发布的路况分析以及路径推荐，在驾驶舱中进行显示。此外，每一个智能驾驶舱客户端可以通过服务器下发的已连接客户端的端口号申请车辆之间的配对，利用服务器转发对应消息实现车辆之间的通信以及数据分享。

由于发送过程中需要实现持续接收，所以采用回调函数 OnReceive()避免线程占用，为了避免发送文字的混合，设置了多个 Socket 实现发送与接收的封装。

2.1.4 AI 模型分析

基于机器学习算法与将数据视觉，建立疲劳检测模型，通过 AI 分析驾驶员的行为模式和习惯，判断其是否处于疲劳状态，对处于疲劳状态的驾驶员在屏幕上弹出警告并进行语音提醒，同时对驾驶员疲劳状态数据进行保存与可视化处理，利用饼状图分析驾驶员各项疲劳状态指标的占比。

在实现过程中需要确定疲劳驾驶检测的标准，并将各项疲劳检测指标保存至相应 CSV 文件，实现数据可视化处理。

2.1.5 软件界面设计

界面主要包括驾驶舱界面、服务器界面与传感器界面。驾驶舱界面包括传感器数据显示区域、传感器控制区域、网络通信记录、网络通信控制区域数据处理操作等，服务器界面主要包括客户端连接记录，网络通信记录与服务器控制区域。传感器界面主要包括传感器初始化区域以及传感器日志显示区域。

2.2 总体方案设计

参考国家车联网产业标准体系建设指南（智能网联汽车）（2023 版），本项目主要由驾驶舱模块、服务器模块与传感器模块组成，驾驶舱模块包括登录模块、传感器数据显示模块、采集控制模块、文件管理模块、数据查询及可视化模块、AI 分析模块、服务器信息推送模块等。

软件由 MFC 库、STL 库、Microsoft Communications Control 库支持，在 C++ 环境下运行，数据可视化处理[2]与绘图功能由 Matplotlib 库支持，DMS 检测模块由 OpenCV 库结合 dlib 库[3]支持，利用 wxpython 实现图形化界面，在 Python 环境下运行。

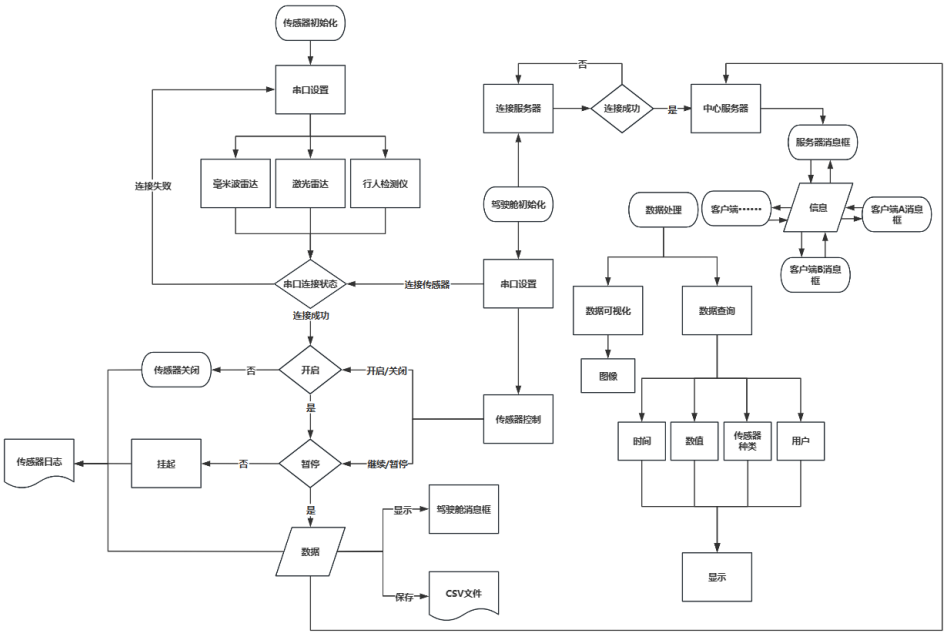

图 2-1 主体操作流程图

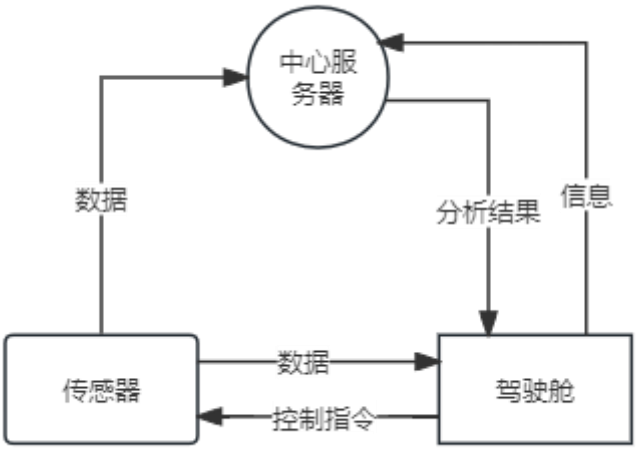

图 2-2 数据传输流程

2.3 制约因素分析及环境相关论证

在社会与驾驶员安全方面，智能化汽车软件的应用为驾驶员提供了便捷的操作方式。不仅可以实时可视化接收到的传感器数据，还可以对数据进行分析并对驾驶员实现预警。MDS 检测达到了辅助驾驶的目的，在推动我国新能源汽车智能化方面做出了贡献，进一步保障用户的安全。同时，软件设计时从驾驶员需求

出发，自由控制传感器的运行与停止，为用户提供了更加灵活的选择，而且通过服务器转发实现车际数据交流，有利于构建良好的驾驶生态。

在环境保护与可持续发展方面，智能驾驶技术有望改善车辆行驶路线规划，从而减少交通拥堵和汽车尾气排放，有助于减少空气污染和温室气体排放，促进环境保护，同时可以优化交通流量，减少交通拥堵，提高道路利用率，缩短行车时间，提升交通效率，助力城市的可持续发展。

在法律方面，本次课程设计所采用的库、算法等资源均来自开源项目或统一公用标准，不存在商业法律侵权等纠纷的风险。在程序设计方面，参考了《计算机软件开发规范》GB8566-88[4]、《信息处理-程序构造及其表示法的约定》GB/T13502-92 等软件开发相关的国家标准的部分条例，以确保软件开发与现行要求吻合。

2.4 项目计划与管理方法

2.4.1 项目计划

本项目的开发主要过程如表 2-1 所示，按照设定的三个模块进行项目实现。第一步在工信部以及中国人民政府网寻找相关政策背景，在知网查找相关文献，在查找项目相关背景后，我转去查找串口通信与 TCP 通信的架构，在通信架构完成搭建后，我开始实现传感器类的设计以及数据的设计，然后开始实现服务器的界面与功能，完成服务器与客户端信息收发，最后实现车车通信与 MDS 检测。

表 2-1 项目进度

时间	项目
2.8-2.9	理解项目需求、查阅相关资料文献
2.9-2.12	实现传感器类设计，串口数据传输
2.13	实现驾驶舱与服务器界面初步设计
2.14-2.16	数据的保存、查询与可视化
2.17	驾驶舱与服务器界面优化
2.18-2.19	服务器数据传输、预警
2.20-2.21	优化服务器，实现车车通信
2.22-2.23	MDS 检测的改进与调试
2.24	优化数据查询模块
2.25	完善设计说明书
2.26	答辩

2.4.2 管理方法

项目主要分为三个文件，分别是保存传感器的 A_Sensor,保存客户端的 A_DataSrv 以及保存驾驶舱的 A_Car。统一保存在文件夹的 A_ClassDesign。

在代码编写完成一个需求时，我会在 VS2019 中导出一个模板，实现项目的保存，为了放置电脑损坏的情况，我会定期将代码打包上传至云端，避免代码丢失，在项目完成后，我打包存放在了我的开源仓库。

第 3 章 对象模型

3.1 类的设计

由于服务器端和用户端都是由登录界面加各子界面构成，所以除了默认界面以外，还需要在资源视图中新建窗口。为了方便进行对话框的切换，继承 CDialog 来实现对话框的便捷切换，实现网络通信、串口通信、数据可视化等功能，并对其进行封装，提高了代码的运行效率，优化其可读性。

3.2 传感器类设计

传感器类的主要成员如表 3-1 所示，采用串口传输数据，传输的数据类型有文字、图片（数据可视化）与视频（MDS 检测）。

表 3-1 传感器类名与功能

类名	功能
CSensor	基类
CPerSensor	行人检测类
CSpeedSensor	速度检测类
CDistanceSensor	车距检测类

传感器基类的成员变量主要为相应的传感器数据以及串口初始化所需要的基本变量，对于数据来说：成员函数主要包括传感器的初始化（Initialize()）、启动（Start()）、停止（stop（））以及数据的获取（GetData()）。对于串口配置来说：成员函数主要包括串口配置的初始化（Initialize()）以及串口配置的更改（modify（））。

```
#pragma once
#include"02_TCPClientDlg.h"
#include"string"
#include<vector>
using namespace std;
class CSensor
{
public:
    CMy02_TCPClientDlg *dlge=NULL;//主界面指针
    CSensor *pCSensor = NULL;//基类指针

    int m_portID;//端口号
    string m_Bund;//波特率
    string m_DataBit;//数据位
```

```
    string m_StopBit;//停止位
    string m_CheckBit;//校验位
    double m_iData;//双精度浮点数据
    vector<string> m_Data;//储存传感器信息

    CSensor();
    ~CSensor();
    virtual void modify(int portID, string Bund);//修改串口配置
    // 初始化传感器(串口号，波特率等）
    virtual bool Initialize() = 0;
    // 启动传感器
    virtual bool Start() = 0;
    // 停止传感器
    virtual bool Stop() = 0;
    // 获取传感器数据
    virtual int GetData()=0;
    //获取双精度浮点数据
    virtual double GetDate()=0;
    //数据存储
    virtual void DataCollect(double iData);
};
```

以行人检测类为例：

```
#include "stdafx.h"
#include "CPerSensor.h"
bool CPerSensor::Initialize()
{
    // 初始化速度传感器,
    maxPerson = 8;
    minPerson = 0;
    m_portID = 7;
    return TRUE;
}
// 实现启动方法
bool CPerSensor::Start()
{
    // 启动速度传感器的代码
    dlge->communications1.put_PortOpen(true);
    this->GetData();
    return TRUE;
}
// 实现停止方法
bool CPerSensor::Stop()
{
    // 停止速度传感器的代码
```

```
    dlge->communications1.put_PortOpen(FALSE);
    return TRUE;
}
// 实现获取数据方法
int CPerSensor::GetData()
{
    // 生成合理范围内的随机速度
    randomPerson = minPerson + (rand() / (double)RAND_MAX) * (maxPerson - minPerson);
    return randomPerson;
}
```

3.3 服务器类的设计

服务器主要包括 CConnSocket 与 CServerSocket 两个类，还有一个服务器类用于辅助车车通信，这一模块在选做功能中会具体讲解。其中，CServerSocket 是服务器的主要的功能实现类，CConnSocket 是专门用于储存连接服务器的客户端信息。主要继承的基类是 CSocket，能够利用回调函数实现连接、断开、接收，避免了循环占用线程以及多线程。但是由于本身不能两个线程访问，导致给后期代码编写留下一定问题。

CConnSocket：

```
#pragma once
#include"string"
using namespace std;
// CConnSocket 命令目标
class CMy02_TCPServerDlg;//避免头文件相互包含

class CConnSocket : public CSocket
{
public:
    CConnSocket(CMy02_TCPServerDlg* dlg = NULL);//构造，指向主服务器
    virtual ~CConnSocket();
    virtual void OnSend(int nErrorCode);  //发送
    virtual void OnReceive(int nErrorCode);//接收
    virtual void OnClose(int nErrorCode);//断开连接
    // 设置连接的客户端的 IP 和端口
    void SetClientAddr(CString ip, USHORT port);
private:
    CString m_ip;//保存连接的客户端 ip
    USHORT m_port//保存已连接客户端端口号;
    CMy02_TCPServerDlg* m_dlg;//服务器主页面指针
public:
```

```
    void Sender( CString msg);//发送
    void SendAll();  //向所有客户端发送
    void generateRandomLanguage();生成文本回答客户端发送的问题
};
```

在 CConnSocket 类中，主要的变量是 m_ip 与 m_port，用于储存连接的客户端端口号与 ip 地址，OnSend、Onreceive 与 Onclose 函数用于发送、接受信息以及检测断开连接的客户端，SetClientAddr()函数用于设置连接客户端的端口号。

CServerSocket：

```
#pragma once
//#define _WINSOCK_DEPRECATED_NO_WARNINGS 1
#include  "ConnSocket.h"

#include <list>
using namespace std;
// CServerSocket 命令目标
class CMy02_TCPServerDlg;//避免头文件相互包含
class CServerSocket : public CSocket
{
public:
    CServerSocket(CMy02_TCPServerDlg* dlg = NULL);
    virtual ~CServerSocket();
    // 接受到客户端连接的 回调函数
    virtual void OnAccept(int nErrorCode);
    // 关闭所有连接客户端
    void CloseAllConn();
private:
    CMy02_TCPServerDlg* m_dlg;//主窗口指针
    list<CConnSocket*> m_clientList;//存放连接客户端的端口号

public:
    void Sender(CString msg);//向客户端发送一定格式的信息
    void ClientInsrt();//向 listcontrol 插入信息
};
```

在CServerSocket类中，主要设计的成员变量list<CConnSocket*> m_clientList，在本代码服务器中起到关键作用，用于储存客户端的所有信息，OnAccept()函数主要是用于监听之后的 Accept。

3.4 驾驶舱类的设计

由于驾驶舱程序需要实现的功能很多，且不同功能之间存在某些代码的复用现象，因此，驾驶舱客户端通过分解为 8 个不同的类来进行封装，每个类都负责

特定的功能，这降低了代码的冗余性，提高了可重用性，使得代码更易于理解和维护。程序包含 8 个封装好的类，如表 3-1 所示。

表 3-2 驾驶舱类名与功能

类名	功能
CConnSocket	实现与服务器连接
CMy02_TCPClientDlg	主对话框
DataSearch	实现数据检索
FileSave	实现文件保存
login	登录
CMSCOMM1	实现串口通信
ClientSocket1	实现车车通信
CarCommu	车车通信窗口

3.4.1 驾驶舱主界面类设计

由于主界面结合串口通信、TCP 网络通信与文件操作，代码较长，不全部展示，以下选取部分进行综合展示

串口数据接收：

此代码实现了串口数据的接收以及数据的保存，只截取了部分关键。

```
1.  if (chuankouhao.GetCurSel() == 0)
2.          {
3.              SpeedShow += m_TemDataStr;  //把 char 加入到所显示的字符
4.              Time.Format(_T("[%d:%d:%d] :"), t.wHour, t.wMinute, t.wSecond);
5.              unit[chuankouhao.GetCurSel()].Format(_T("km/h"));//单位
6.              msgs = Time + SpeedShow+unit[0];//串口消息
7.              if (20 <= (_ttoi(SpeedShow)))
8.              {
9.                  Time.Format(_T("%d:%d:%d"), t.wHour, t.wMinute, t.wSecond);
10.                 pFile = new FileSave;
11. // FileSave 存放了传感器种类、数据、传输时间一级单位
12.                 pFile->Sensor = "毫米波雷达";
13.                 pFile->dData = CT2A(SpeedShow.GetString());
14.                 pFile->time = CT2A(Time.GetString());
15.                 pFile->unit = CT2A(unit[chuankouhao.GetCurSel()].GetString());
16.                 m_Data.push_back(*pFile);/放入 vector 容器
17.                 SendSrv.push_back(*pFile);
18.             }
19.             CSpeedShower.SetWindowText(msgs);//前台显示
20.             CSpeedShower.LineScroll(CSpeedShower.GetLineCount() - 1, 0);//显示最新
```

3.4.2 驾驶舱数据检索类设计

数据检索的主要后台函数实现是通过函数 void DataSearch::performSearch()，函通过输入查询条件（用户名、传感器类型、时间范围、数据范围），之后在迭代器循环中通过 if 语句判断是否符合查询条件。

```
1.  void DataSearch::performSearch(const std::string& userQuery, const std::string& sensorType
    Query,
2.      const std::string& Stattime, const std::string& overime,double minData, double maxData) {


3.      m_listCtrl.DeleteAllItems();
4.      CString Data, Num;
5.      // 解析起始时间和结束时间
6.      tm startTime = parseTime(Stattime);
7.      tm endTime = parseTime(overime);
8.      // 遍历 vector 所有数据，检查每个数据条目是否符合查询条件，并将符合条件的添加
    到 listcontrol 中
9.      for (const auto& data : searchDataList) {
10.         bool match = true;
11.         // 检查用户
12.         if (!userQuery.empty() && data.user != userQuery) {
13.             match = false;
14.         }
15.         // 检查传感器类型
16.         if (!sensorTypeQuery.empty() && data.sensorType != sensorTypeQuery) {
17.             match = false;
18.         }
19.         // 检查时间范围
20.         tm dataTime = parseTime(data.time);
21.         int num = dataTime.tm_hour * 3600 + dataTime.tm_min * 60 + dataTime.tm_sec;
22.         int start= startTime.tm_hour * 3600 + startTime.tm_min * 60 + startTime.tm_sec;
23.         int over= endTime.tm_hour * 3600 + endTime.tm_min * 60 + endTime.tm_sec;
24.         if (num < start || num > over) {
25.             match = false;
26.         }
27.         // 检查数据范围
28.         if (!(this->maxData.IsEmpty() && this->minData.IsEmpty()) && (data.data < minData ||
    data.data > maxData)) {
29.             CString a;
30.             a.Format(_T("%lf"), data.data);
31.             match = false;
32.         }
33.         // 如果所有条件都匹配，则将数据添加到 listcontrol 中
34.         if (match) {
```

```
            int index = m_listCtrl.GetItemCount(); // 获取当前列表项数量
            m_listCtrl.InsertItem(index, _T("")); // 要有这个，不然显示不了
            Data.Format(_T("%.2lf"), data.data);
            Num.Format(_T("%d"), index + 1);
            // 使用 localtime_s 替代 localtime
                CString formattedTime;
                formattedTime.Format(_T("%02d:%02d:%02d"), dataTime.tm_hour, dataTime.tm_min, dataTime.tm_sec);
                // 插入用户、传感器类型、时间、数据和单位
                m_listCtrl.SetItemText(index, 0, Num);
                m_listCtrl.SetItemText(index, 1, CA2T(data.user.c_str()));
                m_listCtrl.SetItemText(index, 2, CA2T(data.sensorType.c_str()));
                m_listCtrl.SetItemText(index, 3, formattedTime);
                m_listCtrl.SetItemText(index, 4, Data);
                m_listCtrl.SetItemText(index, 5, CA2T(data.unit.c_str()));
                index++;//换行

        }
    }
}
```

3.5 串口通信设计

3.5.1 信息接收设计

对于串口通信，使用 m_MSCommControl 变量进行函数的调用，分别设置缓冲区，并进行初始化。

```
void Cchuankoutongxun1Dlg::OnOncommMscomm1()//自动接收程序
{
    VARIANT       m_Variant_Rec; //接收类型定义
    COleSafeArray  m_SafeArray_Rec;  //用于处理任意类型和维数数组的类
    LONG          m_DataLenth, nCount;  //定义接收的长度和计数用
    const int     m_RecByteLenth = 2048;
    BYTE          m_RecDataByte[m_RecByteLenth];  //定义接收缓冲区
    CString       m_TemDataStr;  //接收字符串，用于转换后显示
    if (communications1.get_CommEvent() == 2)  //串口事件值为 2，表示接收缓冲区字符
    {
        m_Variant_Rec = communications1.get_Input();  //获取缓冲区内容
        m_SafeArray_Rec = m_Variant_Rec;   //把缓冲区内容放到 SafeArray 数据结构中

        m_DataLenth = m_SafeArray_Rec.GetOneDimSize(); //获取一维数据的长度
        for (nCount = 0; nCount < m_DataLenth; nCount++)
            m_SafeArray_Rec.GetElement(&nCount, m_RecDataByte + nCount); //检索 SafeArray 中的内容，用不同 ncout 位置进行指针的偏移量，取出对应位置内容
```

```
17.
18.         for (nCount = 0; nCount < m_DataLenth; nCount++)
19.         {
20.             BYTE m_Buff = *(char*)(m_RecDataByte + nCount);  //从数组开头进行 ncount 偏移量，提取对应位置的内容
21.             m_TemDataStr.Format(_T("%C"), m_Buff); //把相关 BYTE 内容转化为 char
22.             xianshi += m_TemDataStr;  //把 char 加入到所显示的字符串中
23.         }
24.         xianshi += _T("\r\n");
25.         UpdateData(FALSE);
26.     }
27.
28.     communications1.put_InBufferCount(0);  //取出数据后，进行数据的清零
29.     communications1.put_OutBufferCount(0);
30. }
```

3.5.2 信息发送设计

传感器生成的数据主要有三种，分别是速度、车距与行人数，在数据生成的时候主要采用随机数生成，考虑到数据必须符合客观事实，所以限定了传感器所产生数据的最大值与最小值以及数据的类型。

对于传感器仿真，以车距传感器为例，根据分析，拟将车距范围定在 1-10m 之间。为了保证充分的随机性，使用(double)rand()/RAND_MAX 语句生成浮点型的随机小数，并通过加法实现固定范围的确定，RAND_MAX 是 C 语言中的一个宏，运行时可以生成随机数所能返回的最大数值，并利用。MaxDistance 与 min Distance 实现随机数之的范围控制。

```
1.  randomDistance = minDistance + (rand() / (double)RAND_MAX) * (maxDistance - minDistance);
```

由于传感器都有一个共同的基类 CSensor，所以利用 C++中多态的知识，利用 chuankouhao.GetCurSel()获取所选传感器类型，之后利用 Switch 语句将所选传感器的指针指向对应的传感器类，实现相应数据的生成与发送。

```
1.      if (communications1.get_PortOpen() == true)
2.     {
3.         switch (chuankouhao.GetCurSel())//传感器选择
4.         {
5.         case 0:
6.             {
7.                 m_Sensor = new CSpeedSensor();
8.                 Data->Speed = m_Sensor->GetData();
9.                 Data->msg.Format(_T("[%d:%d:%d] 速度:%d"), t.wHour, t.wMinute, t.wSecond, Data->Speed);
10.                Data->msg += "km/h    ";
```

```
            m_AutoSendText.Format(_T(" %d"), Data->Speed);
            communications1.put_Output(COleVariant(m_AutoSendText));
            AddMsg(Data->msg);
        }
        break;
    case 1:
        {
            m_Sensor = new CDistanceSensor();
            Data->Distance = m_Sensor->GetDate();
            Data->msg.Format(_T("[%d:%d:%d] 前车距离:%.2lf"), t.wHour, t.wMinute, t.wSecond, Data->Distance);
            Data->msg += "m";
            m_AutoSendText.Format(_T(" %.2lf"), Data->Distance);
            communications1.put_Output(COleVariant(m_AutoSendText));
            AddMsg(Data->msg);
        }
        break;
    case 2:
        {
            m_Sensor = new CDistanceSensor();
            Data->Person = m_Sensor->GetData();
            Data->msg.Format(_T("[%d:%d:%d] 周围行人:%d"), t.wHour, t.wMinute, t.wSecond, Data->Person);
            Data->msg += "人    ";
            m_AutoSendText.Format(_T(" %d"), Data->Person);
            communications1.put_Output(COleVariant(m_AutoSendText));
            AddMsg(Data->msg);
        }
        break;
    default:
        break;
    }
}
```

第 4 章 图形界面

4.1 界面整体介绍

随着汽车功能的不断增加，传感器的种类与数据也日趋复杂，对于操作界面而言，采用更人性化的操作方式来给驾驶者和乘客提供一个高效率且安全简洁的操作界面是重中之重。

经过查阅资料新能源汽车厂商的界面设计，我重新思考了驾驶员的需求，以及未来驾驶舱应该是什么样子的问题。在设计智能驾驶舱软件的 UI 时，本项目紧密结合了驾驶行为，并以驾驶安全为核心进行设计。本项目考虑了驾驶员真正需要的功能和信息，并将其以用户友好的方式呈现，以提高驾驶舱的易用性和使用效果。此外，我也思考了服务器在整个系统中的角色和功能，确保其能够提供必要的支持和处理各种数据，以实现智能驾驶舱的高效运行。

图 4-1 智能驾驶仓界面设计

界面主要分为传感器控制与显示区域、服务器控制与显示区域、数据处理区域以及 AI 分析区域三大部分。

对于传感器控制与显示区域，使用 Edit Contorl 框用于传感器信息的显示，传感器控制区域的 Combo Box 用于选择传感器的类型，下方四个 Button 用于传感器的启动、暂停、继续与停止。

对于服务器控制与显示区域，利用 List Contorl 实现与服务器的 TCP 通信，

服务器通信的控制区域的 Button 用于服务器的连接与断开。

对于数据处理区域与 AI 分析区域，利用 DoModal()实现窗口的跳转。

图 4-2 服务器界面设置

服务器界面分为服务器控制区域、信息显示区域、客户端连接记录区域与自定义消息发送区域。

服务器控制区域用于控制服务器的开启与关闭，信息显示区域与客户端连接记录区域采用控件 List Contorl 进行设计，分别采用 Icon 与 Report 视图实现信息显示。

4.2 登陆界面设计

对于登录界面的 button，需要用第二个界面的类定义一个实例化类，然后调用 DoModal()函数即可实现窗口的跳转。

图 4-3 登录界面设置

利用控件 Check Box 实现密码的显示与隐藏

```
1.  void login::OnBnClickedCheck1()
2.  {
3.      // TODO: 在此添加控件通知处理程序代码
4.      if (IsDlgButtonChecked(IDC_CHECK1) == BST_CHECKED)  //如果被选中
5.      {
6.          UpdateData(true);
7.          CEdit* pEdit = (CEdit*)(this)->GetDlgItem(IDC_EDIT2);
8.          pEdit->SetPasswordChar(0);
9.          SetDlgItemText(IDC_EDIT2, password_value);
10.         UpdateData(false);
11.     }
12.     else
13.     {
14.         UpdateData(true);
15.         CEdit* pEdit = (CEdit*)(this)->GetDlgItem(IDC_EDIT2);
16.         pEdit->SetPasswordChar('*');//显示*
17.         SetDlgItemText(IDC_EDIT2, password_value);
18.         UpdateData(false);
19.     }
20.     GetDlgItem(IDC_EDIT2)->SetFocus();
21. }
```

4.3 查询界面设计

查询界面主要由查询结果显示、查询条件设置与查询控制三部分组成，利用控件 List Contorl 的 report 视图时间传感器数据的显示，利用控件 Combo Box 与 Edit Contorl 实现查询条件的设置。

图 4-4 查询界面设置

4.4 车车通信(V2X)界面设计

V2X 界面设计主要分为在线车辆显示区域、接收消息显示区域、V2X 通信设置区域与 V2X 数据共享区域。在线车辆显示区域利用 List Contorl 显示已连接服务器的客户端具体信息，采用 Edit Contorl 实现端口号锁定与接受消息显示。

图 4-5 V2X（车车通信）界面设计

4.5 传感器界面设计

传感器界面主要分为传感器设置区域与传感器日志显示区域，传感器设置区域利用控件 Combo box 实现传感器的选择，传感器所对应的串口号等串口通信参数已经在后台进行设置，传感器日志用于显示传感器工作状态与传感器数据

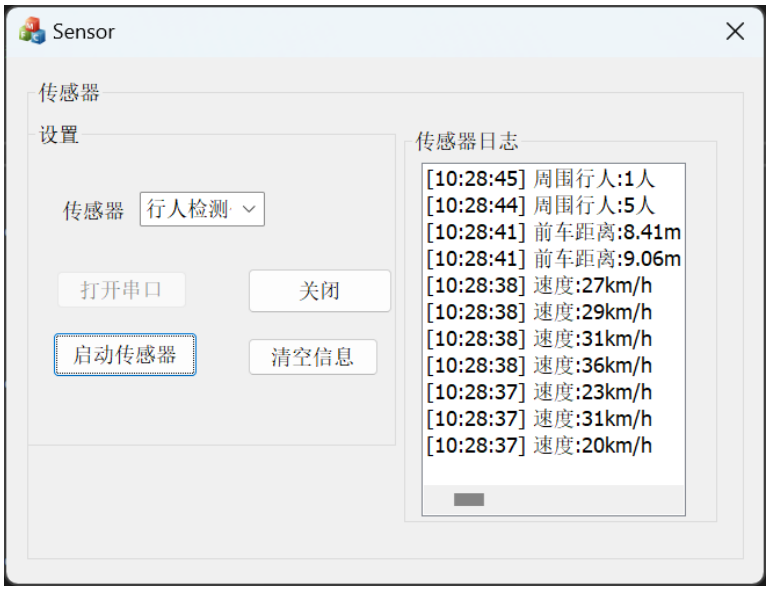

图 4-6 传感器界面设计

第 5 章 数据读写与分析

5.1 数据的写入与保存

数据由传感器产生，通过串口通信进行数据的传输，将收到的数据利用 Vector 容器 m_Data 进行后台存放，之后进行文件的保存，利用迭代器将数据从 m_Data 中传输给文件保存。

```
1.  void CMy02_TCPClientDlg::OnBnClickedButton12()
2.  {
3.      int rank = 1;
4.      CFileDialog* p_savefile = new CFileDialog(FALSE, L".csv", NULL, OFN_READONLY,
    L"CSV(*.csv)|*.csv");//以 csv 格式保存
5.      p_savefile->DoModal();//打开文件保存路径
6.      string FilePath_Save = CT2A(p_savefile->GetPathName().GetString());
7.      delete p_savefile;释放内存
8.      char* old_locale = _strdup(setlocale(LC_CTYPE, NULL));//语言区域设置
9.      if (FilePath_Save.empty()) { // 检查文件路径是否为空
10.         // 如果文件路径为空，可能是用户取消了对话框，这时候应该退出程序
11.         MessageBox( L"保存文件失败，请勿在保存时打开文件", L"错误
    ", MB_OK | MB_ICONERROR);
12.         return ;
13.     }
14.     // 打开文件进行写入
15.     ofstream SaveFile(FilePath_Save, ios::out);
16.     if (!SaveFile) {
17.         MessageBox( L"保存文件失败，请勿在保存时打开文件", L"错误
    ", MB_OK | MB_ICONERROR);
18.         return ;
19.     }
20.     // 写入表头
21.     SaveFile << "标号" << "," << "用户" << "," << "传感器类型" << "," << "时间" << "," << "
    数据" <<"," << "单位" << endl;
22.     CString msg;
23.     for (auto it = m_Data.begin(); it != m_Data.end(); it++) {
24.         // 写入每个 userinfo 对象的数据
25.         SaveFile << rank << "," << it->user << "," << it->Sensor << "," << it->time << "," << it->
    dData << "," << it->unit;
26.         rank++;
27.         SaveFile << endl;
28.     }
29.     // 关闭文件
30.     setlocale(LC_CTYPE, old_locale); //还原语言区域的设置
```

```
    free(old_locale);//还原区域设定
    SaveFile.close();
    MessageBox( L"保存成功", L"提示", MB_OK | MB_ICONINFORMATION);
    poltData();//绘制图像
    return ;
}
```

5.2 数据的读出与显示

由于为数据显示单独设计了一个窗口，所以先点击数据查询，指定文件路径后打开相关文件与并跳转到查询窗口，当跳转到查询窗口时，在 Oninit()函数中的 listinit()函数能够实现数据从文件读出数据并显示在窗口中。

从主窗口跳转到查询窗口的控件函数：

```
void CMy02_TCPClientDlg::OnBnClickedButton10()
{
    // TODO: 在此添加控件通知处理程序代码
    CFileDialog openFileDialog(TRUE, L".csv", NULL, OFN_FILEMUSTEXIST | OFN_HIDE
READONLY, L"CSV Files (*.csv)|*.csv||");
    if (openFileDialog.DoModal() == IDOK) {
        CString filePath = openFileDialog.GetPathName();

        // 创建文件内容窗口并显示文件内容
        DataSearch fliecontent;
        fliecontent.fileContentWindow(filePath);
        fliecontent.DoModal();
    }
}
```

从文件中读取数据并显示的函数 listInit()：

函数首先通过 m_FilePath 找到之前数据保存的文件，由于 List Contorl 已经初始化有标题，所以文件的第一行不进行读取，利用 if 判断行数并结合 continue 跳过第一行，之后使用 strinstream 按照逗号分隔，并在循环中利用 getline()写入到 List Contorl 中。代码中的 Switch 语句适用于筛选传感器信息并存储到命名为 SearchDataList 的 Vector 容器中，便于后续在后台的数据查询与检索。

```
void DataSearch::listInit()
{
    // TODO: 在此处添加实现代码.
    std::ifstream file(m_FilePath);
    int rank = 1;
    if (file.is_open()) {
        std::string line;
        bool isFirstLine = true; // 添加一个标志以确定当前是否是第一行
        while (getline(file, line)) {
```

```
10.          // 如果是第一行，则跳过
11.          if (isFirstLine) {
12.              isFirstLine = false;
13.              continue;
14.          }
15.          // 使用 stringstream 按照逗号分割每一行的数据
16.          std::stringstream ss(line);
17.          std::string item;
18.          int columnIndex = 0;
19.          // 插入新的一行
20.          int rowIndex = m_listCtrl.GetItemCount();
21.          m_listCtrl.InsertItem(rowIndex, _T(""));
22.          // 逐个读取每个字段，并插入到对应的列中
23.          while (getline(ss, item, ',')) {
24.              // 将 string 转换为 CString
25.              CString str(item.c_str());
26.              // 插入到对应的列
27.              m_listCtrl.SetItemText(rowIndex, columnIndex, str);
                                  //分析传感器信息
28.              switch (columnIndex) {
29.              case 0: {rankNum = item; break; }//序号
30.              case 1: user = item; break;  //用户名
31.              case 2: sensorType = item; break;  //传感器种类
32.              case 3: time = item; break;  //时间
33.              case 4: data = std::stod(item); break;//数据
34.              case 5: unit = item; break;  //单位
35.              default: break;
36.              }
37.
38.              columnIndex++;
39.          }
40.          DataSearch Data;
41.          Data.SensorData(rankNum, user, sensorType, time, data, unit);//有参构造，存储数据
42.          searchDataList.push_back(Data);//存放到容器中
43.      }
44.      file.close();
45.  }
```

图 5-1 数据的前台显示

标号	用户	传感器类型	时间	数据	单位
1	Car1	毫米波雷达	22:26:37	20	km/h
2	Car1	毫米波雷达	22:26:38	31	km/h
3	Car1	毫米波雷达	22:26:39	23	km/h
4	Car1	毫米波雷达	22:26:40	36	km/h
5	Car1	毫米波雷达	22:26:41	31	km/h
6	Car1	毫米波雷达	22:26:42	29	km/h
7	Car1	毫米波雷达	22:26:43	27	km/h
8	Car1	毫米波雷达	22:26:44	37	km/h
9	Car1	毫米波雷达	22:26:45	36	km/h
10	Car1	激光雷达	22:26:53	7.72	m
11	Car1	激光雷达	22:26:54	2.57	m
12	Car1	激光雷达	22:26:55	8.73	m
13	Car1	激光雷达	22:26:56	7.39	m
14	Car1	激光雷达	22:26:57	5.62	m
15	Car1	激光雷达	22:26:58	3.74	m
16	Car1	激光雷达	22:26:59	1.13	m
17	Car1	激光雷达	22:27:00	1.82	m
18	Car1	行人检测仪	22:27:07	4	人
19	Car1	行人检测仪	22:27:08	2	人
20	Car1	行人检测仪	22:27:09	2	人

图 5-2 数据的 CSV 文件保存

5.3 数据的分析

5.3.1 数据可视化分析

数据的可视化主要分为速度数据可视化、行人数数据可视化以及疲劳程度数据可视化。基于 Python 的 matpoltlib 库进行数据可视化处理，用 py 程序获取保存为 CSV 格式的数据文件，再调用 matpoltlib 库作图[0]。

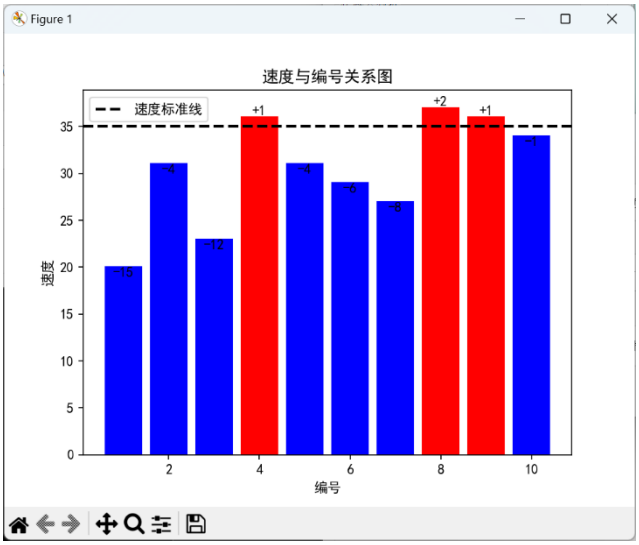

图 5-3 速度数据的可视化

在城市的中心区域，假设速度的标准线设定为 35km/h，当数据所测得的速度超过所设定的标准线时，数据会被标红处理，着用筛选除去了海量正常数据，找到了异常数据点。便捷后续的查询工作。

图 5-4 行车雷达图

行人检测仪会不断扫描周围的环境，佳能周围行人（障碍物）的数目与位置传输至驾驶舱，实现精准安全驾驶。

图像中的紫色区域代表行车时的可能危险区域，红点代表移动的行人或者障碍物，每一个半径不同的圆代表与车辆的距离不同，如图中显示，在行驶过程中，周围总共的行人有 5 人，危险区域中的行人数为 1 人，方位为东南方向，距离约为三个标准单位。有了此数据，驾驶员能够提前了解潜在风险，做好预防准备。

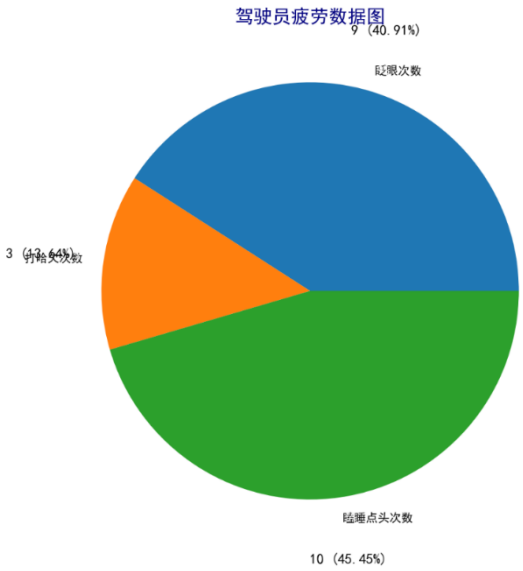

图 5-5 疲劳驾驶状态饼状图

当驾驶员被检测系统判定为疲劳状态时，系统会在后台生成对应的 csv 文件，实现数据的报讯，当驾驶员像读取并观察数据时，可以通过可视化饼状图实现，同时通过对疲劳状态的三个指标（眨眼数、瞌睡点头数与打哈欠数）的占比分析，能够更好的优化疲劳状态的判断模型。

5.3.2 数据的服务器分析

驾驶员在行驶过程中的数据能够通过 TCP 网络通信传输给服务器，当服务器检测到数据异常时，会向驾驶员发出警报。

图 5-6 服务器数据分析预警

5.3.3 数据的检索

当驾驶员得到了服务器的报警或者从可视化图像中发现了异常数据点时，能够利用数据的检索功能实现具体数据内容的查询（以上方异常速度数据为例）。输入数据范围与传感器种类，完成检索。

图 5-7 异常数据检索

通过具体数据的检索，驾驶员可以知晓异常数据的具体编号、产生用户、检测传感器以及产生时间。以图 5-3 为例，图中异常数据编号为 4、8、9，在检索后能够得知时用户 Car1 在 11 点左右检测到速度的异常，便于驾驶员去复盘行车记录。

第 6 章 扩展功能设计

6.1 数据的可视化分析

具体界面与操作流程在章节 5.3.1 已具体说明，不做赘述，下方以雷达图为实例展示代码部分，首先从 CSV 文件中读取第四列人数信息，之后利用 matplotlib 库实现对象的创建，利用读取的数据随机生成代表人的红点，设定距离至少为 1，之后绘制雷达图以及其他标注。

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = ['SimHei'] # 显示中文
plt.rcParams['axes.unicode_minus'] = False # 正常显示负号
# 读取 CSV 文件
csv_file = r'D:\HuaweiMoveData\Users\HUAWEI\Desktop\excel\工作簿 3.csv'
try:
    # 读取数据
    data = pd.read_csv(csv_file)
    # 输出列名以供调试
    print("列名：", data.columns)
    # 检查数据是否为空
    if data.empty:
        print("错误：CSV 文件为空。")
    else:
        # 获取总人数：取第四列数据的最后一个值
        num_people = data.iloc[-1, 3] # 第四列是人数数据
        # 随机生成方位
        num_directions = 9 # 将方向数量设置为 9，与数据点数量匹配
        directions = np.linspace(0, 2 * np.pi, num_directions, endpoint=False) # 平均分布方向
        # 随机生成每个方向上的人数
        num_people_per_direction = np.random.randint(0, num_people, num_directions)
        # 生成每个方向上的人的具体位置
        people_positions = np.random.uniform(0, 2 * np.pi, num_people)
        # 绘制雷达图
        fig, ax = plt.subplots(figsize=(8, 8), subplot_kw=dict(polar=True))
        ax.fill(directions, num_people_per_direction, color='blue', alpha=0.25)
        # 计算并标注每个红点到中心的距离
        for i in range(len(people_positions)):
            angle = people_positions[i]
            radius = np.random.uniform(1, num_people, 1) # 随机生成距离，至少为 1
            ax.scatter(angle, radius, color='red', s=50) # 绘制红点
```

```
            plt.text(angle, radius, f'{radius[0]:.2f}', fontsize=8, ha='center', va='center', color='black'
)
        # 添加标签
        ax.set_yticklabels([])
        ax.set_xticks(directions)
        ax.set_xticklabels(['Direction {}'.format(i + 1) for i in range(num_directions)], fontsize=1
2)
        # 添加标题
        plt.title('周围障碍分布雷达图', size=20, color='black', y=1.1)
        # 添加图例
        legend_labels = ['危险区域']
        ax.legend(legend_labels, loc='upper right')
        # 添加雷达图周围的装饰性元素
        ax.set_frame_on(False)  # 关闭雷达图的边框
        # 在图表中添加文本框显示行人的数量
        plt.text(0, 0, f'总行人数量: {num_people}', fontsize=12, ha='center', va='center')
        # 显示雷达图
        plt.show()
except pd.errors.EmptyDataError:
    print("错误：在 CSV 文件中找不到任何数据。")
except Exception as e:
    print("发生错误：", e)
```

6.2 MDS 疲劳检测分析

随着智能汽车的发展，DMS 的需求将越来越强烈。不仅是为了驾驶安全，也是为了提供更好的驾驶体验。DMS 不仅可以帮助驾驶员及时发现疲劳的迹象，从而避免可能的交通事故。而且还能与其他车载系统结合，提供更为智能和人性化的驾驶体验。未来汽车完善的 DMS 系统，能够很大程度上确保驾驶的安全和舒适。

通过相关资料查找，现有的 MDS 系统主要分为疲劳检测与分心检测两种，判断疲劳的标志为眨眼数、瞌睡点头以及打哈欠。分心检测的主要手段时检测人眼的注视方向以及人脸的移动速度等。

程序主体为开源的软件代码，该程序采用了多种数据，包括人脸朝向、位置、瞳孔朝向、眼睛开合度、眨眼频率、瞳孔收缩率等。通过这些数据的实时计算，可以评估驾驶员的注意力水平，进而分析是否存在疲劳驾驶的情况，并及时作出安全提示。

同时，项目还增加了语音提示模块[5]，当系统检测到驾驶员疲劳时，系统自动播放语音提醒驾驶员注意休息，此外，为了完善驾驶疲劳检测的判断模型，建立了疲劳检测数据饼状图，分级各项数据的占比比重。

图 6-1 MDS 检测分析

代码主体为网络开源项目，添加了语音提示模块与疲劳数据饼状图分析模块，对于项目的检测不做过多赘述。

6.3 V2X 车车通信

V2X 通信技术由福特公司在 2014 年 6 月 3 日发布，在现场展示的是福特两辆经过特殊改造过的车辆，演示了这项 V2X 通信技术是如何防止碰撞事故发生的，街上的每辆汽车都能够互相自由交流，如果有司机对一些潜在的危险没有察觉的话，其他司机可以通过该系统向他们做出预警。

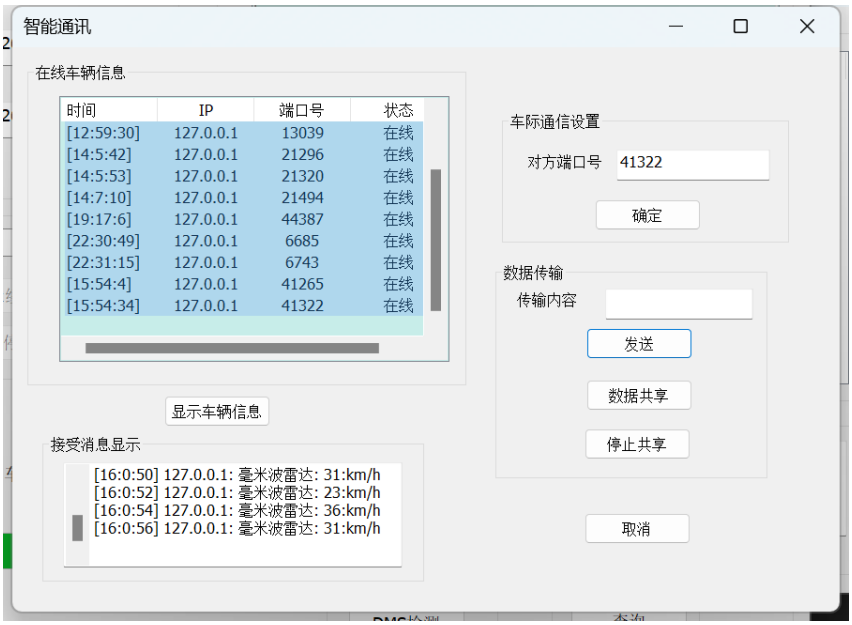

图 6-2 车车通信数据共享实现

本项目中，驾驶员通过接收服务器发送的已连接用户端口号信息，选择自己要连接的用户，之后通过发送端口号请求与对方链接，连接成功后，能够实现与其他驾驶员的交流与数据共享。

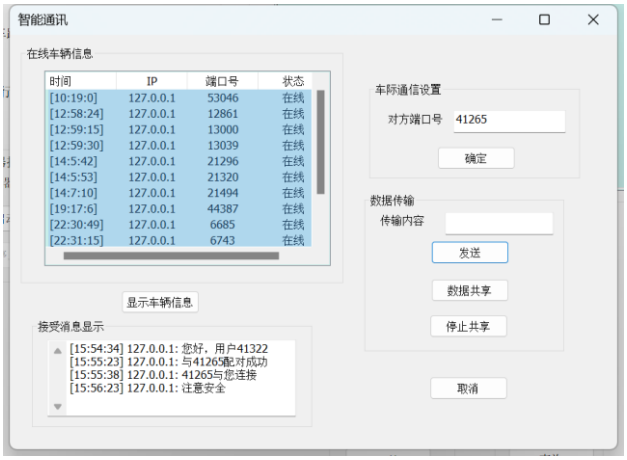

图 6-3 车车通信交流实现（41322 的界面）

图 6-4 车车通信交流实现（41265 的界面）

6.3.1 服务器转发实现

服务器将所有连接的客户端信息储存在 List 列表中，之后接收客户端 A 发送的端口号，创建一个 List 列表，该列表的模板是指向服务器类的指针（list<MyServerSocket*> m_clientList），利用迭代器在 List 列表中进行检索配对，配对成功后，利用相应指针指向所保存的客户端类，完成转发。

客户端信息的录入：

```
1.  void MyServerSocket::OnAccept(int nErrorCode) {
2.      pClientSocket = new MyServerSocket();//指向服务器类
3.      CString a;
4.      int port;
```

```
5.
6.      if (Accept(*pClientSocket)) {
7.          // 获取客户端的地址信息
8.          SOCKADDR_IN clientAddr;
9.          int addrLen = sizeof(clientAddr);
10.         pClientSocket->GetPeerName((SOCKADDR*)&clientAddr, &addrLen);
11.         // 获取客户端端口号
12.         UINT clientPort = ntohs(clientAddr.sin_port);
13.         pClientSocket->m_clientPort = clientPort;
14.         port= clientPort;
15.         char buff1[256] = { 0 };//存放发送的数据
16.         sprintf(buff1, "您好，用户%u", pClientSocket->m_clientPort);
17.         int flag= pClientSocket->Send(buff1,20);
18.         if (!flag)
19.         {
20.             AfxMessageBox(_T("error"));
21.         }
22.         m_clientList.push_back(pClientSocket); // 将客户端指针添加到列表
23.         if (m_clientList.empty())
24.         {
25.             AfxMessageBox(_T("客户端列表为空"));
26.         }
27.         pClientSocket->AsyncSelect(FD_READ | FD_CLOSE);
28.     }
29.     else {
30.         TRACE("Failed to accept client connection. Error code: %d\n", GetLastError());
31.     }
```

客户端信息的转发：

客户端信息的转发，关键在于找到储存对应客户端信息的服务器指针，由于指针保存在 List 列表中，先用服务器接收发送过来的端口号，采用遍历方式匹配配端口号，找到对应指针，再将类中定义的服务器类指针 SelectPort 指向迭代器中匹配的服务器指针，实现消息的转发，利用 this 指针实现向发送方回复成功消息。

```
1.  void MyServerSocket::HandleReceivedData(int nErrorCode) {
2.      char buffer[1024] = { 0 };
3.      char buffers[1024] = { 0 };
4.      char buffers2[1024] = { 0 };
5.  //储存发送消息的字符数组
6.  int bytesRead = Receive(buffer, sizeof(buffer));
7.
8.      CString a(buffer);
9.      char buff[20] = { 0 }; // 声明一个足够大的字符数组来存储转换后的字符串
10.     // 使用 sprintf 将 UINT 转换为字符串
```

```
11.     sprintf(buff, "%u", this->m_clientPort);
12.     // 将字符串输出到控制台
13.     if (bytesRead > 0) {
14.         if (m_clientList.empty()) {
15.             AfxMessageBox(_T("empty"));
16.         }
17.         UINT recevicePort = _ttoi(a);//注意转换，不然乱码
18.        //SelectPort 是服务器指针，配对成功之前，SelectPort 是空指针
19.         if (SelectPort != NULL) {
20.             int flag1 = SelectPort->Send(buffer, bytesRead);
21.         }
22.         //遍历实现配对
23.         for (auto it = m_clientList.begin(); it != m_clientList.end(); ++it) {
24.             if ((*it)->m_clientPort == recevicePort) {
25.                 SelectPort = *it;//配对，获取对应指针
26.                 strcat(buff, "与您连接");
27.                 sprintf(buffers2, "与%u 配对成功", SelectPort->m_clientPort);
28.                 this->Send(buffers2, 30);//提醒发送方配对成功
29.                 SelectPort->Send(buff, 20);//提醒接收方有新的连接请求
30.                 break;
31.             }
32.         }
33.
34.     }
35.     else {
36.         TRACE("Failed to receive data from client on port %d. Error code: %d\n", m_clientPort,
    GetLastError());
37.         Close();
38.     }
39. }
```

6.3.2 客户端的接收与发送

在 V2X 车车通信模块中，主要分为三个线程实现，分别为数据的持续共享，数据的持续接收以及通信界面的主线程。

数据的持续接收线程：

```
1.  DWORD WINAPI process(LPVOID pParam)
2.  {
3.      CString msg;
4.      while (true)
5.      {
6.          int nLength = p_Mainprocess->m_rcver.GetWindowTextLengthW();
7.          // 将光标移动到文本末尾
8.          p_Mainprocess->m_rcver.SetSel(nLength, nLength);
```

```
        // 将新文本追加到编辑框中，并换行
        if (p_Mainprocess->message->messager != msg)
        {
            p_Mainprocess->m_rcver.ReplaceSel(p_Mainprocess->message->messager);
            p_Mainprocess->m_rcver.ReplaceSel(_T("\r\n"));//换行
        }
        p_Mainprocess->message->messager = msg;
        p_Mainprocess->m_rcver.SetWindowTextW(p_Mainprocess->message->messager);
        p_Mainprocess->m_rcver.SetWindowTextW(_T("\r\n"));*/
    }
}
```

代码中：p_Mainprocess 是指向主窗口类的指针，message 是指向主窗口中客户端的指针，messager 是客户端所接收的消息，m_rcver 是 Edit Contorl 的控件变量。

数据的持续共享线程：

该线程将数据从文件中读取所需要发送的数据，之后通过主程序界面的指针实现向服务器发送消息，之后服务器通过该用户连接的端口号实现转发。

```
DWORD WINAPI process3(LPVOID pParam)
{
    CarCommu* p_Mainprocess = (CarCommu*)pParam;
    std::ifstream file("D:\\HuaweiMoveData\\Users\\HUAWEI\\Desktop\\工作簿1.csv", std::ios::in);
    if (file.is_open()) {
        std::string line;
        bool isFirstLine = true; // 添加一个标志以确定当前是否是第一行
        while (getline(file, line)) {
            // 如果是第一行，则跳过
            if (isFirstLine) {
                isFirstLine = false;
                continue;
            }
            // 使用 stringstream 按照逗号分割每一行的数据
            std::stringstream ss(line);
            std::string item;
            std::vector<std::string> columns;
            // 逐个读取每个字段
            int colNum = 1; // 记录当前所在列数
            std::string rowData; // 用于存储同一行的第 3、5、6 列数据
            while (getline(ss, item, ',')) {
                // 如果是第 3、5、6 列，则将数据添加到 rowData 中
                if (colNum == 3 || colNum == 5 || colNum == 6) {
                    rowData += item;
                    // 如果不是最后一列，添加冒号分隔符
```

```
                if (colNum < 6) {
                    rowData += ':';
                }
            }
            colNum++;
        }
        // 发送整行数据给客户端
        send(p_Mainprocess->clientSocket, rowData.c_str(), rowData.length(), 0);
        Sleep(2000);
    }

    file.close();
  }
  return 0;
}
```

第 7 章 系统集成与调试

7.1 系统集成调试

本项目将所有功能集成于三个模块，分别是传感器模块、服务器模块与驾驶舱模块，下方进行各模块的集成调试。

7.1.1 传感器控制调试

利用主驾驶舱的控制区域对传感器数据传输进行启动、暂停继续与停止的控制。

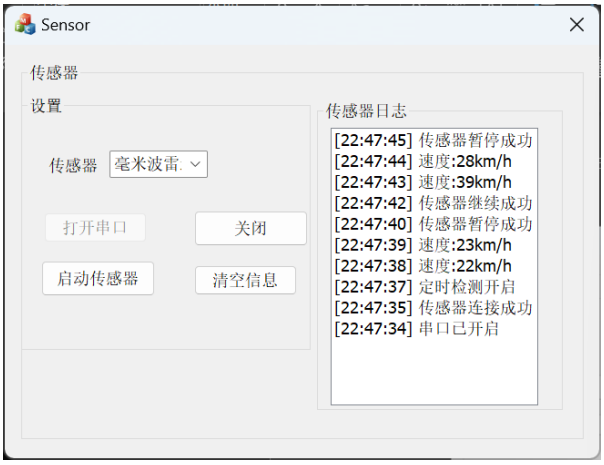

图 7-1 驾驶舱控制传感器启动、暂停继续与停止

在传感器传输数据的同时，传感器能够将数据传输给中心数据库，当中心数据库监测到驾驶舱数据异常，会实现报警，提醒用户注意车辆运行状况。

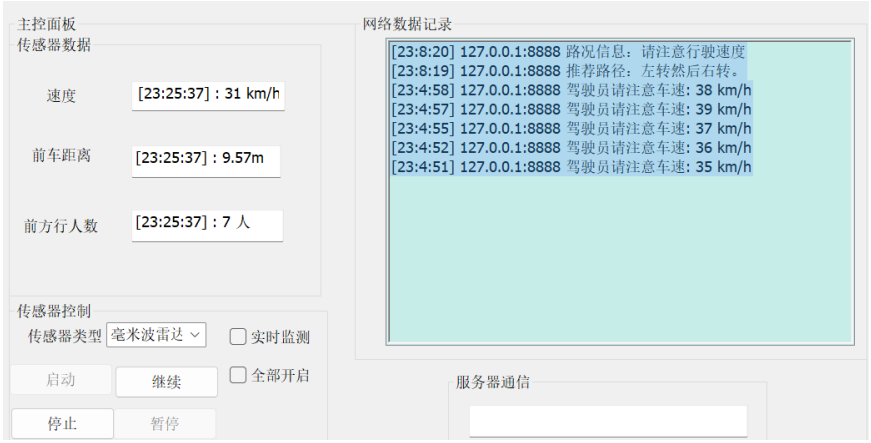

图 7-2 接收中心服务器预警与路径推荐

7.1.2 查询功能调试

当接受传感器传输数据后，通过点击保存文件实现文件保存后，点击查询输入相关条件实现查询，其余功能在前面的章节已经实现，不做重复演示(具体请看第 5 章、第 6 章、第 7 章。

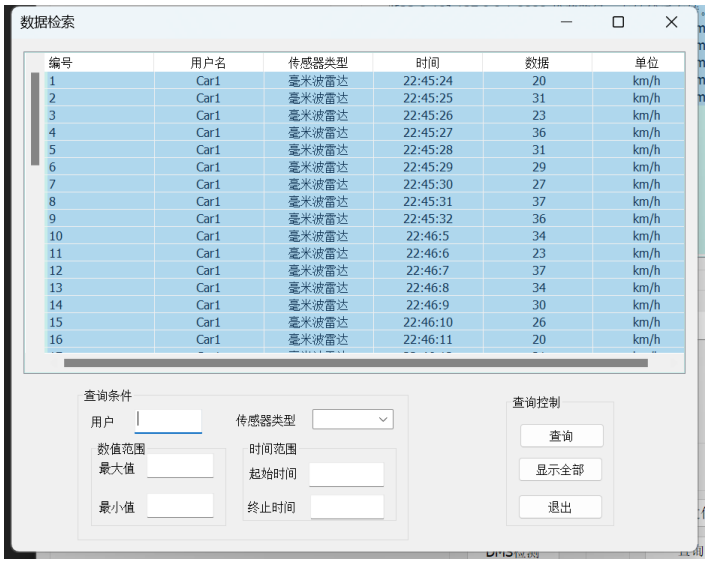

图 7-3 数据筛选前

图 7-4 数据筛选后

7.2 系统调试方法

在系统设计与代码编写的实际过程中，遇到了各种形式的错误，在编写过程中，本项目主要使用了以下三种调试方法：

7.2.1 断点调试

在软件开发中，断点调试是一种常用的技术，它允许开发人员在程序执行过程中暂停执行，以便仔细观察程序的状态、变量的值以及执行流程。这种调试技术通过在代码中设置断点，也就是指定程序在某个特定位置停止执行，从而方便开发人员检查程序的状态和执行路径。一旦程序执行到设置的断点处，它会自动暂停执行，开发人员可以使用调试器提供的功能来查看程序的当前状态，例如检查变量的值、单步执行代码、查看函数调用栈以及检查内存状态等。这样，开发人员就可以更加高效地发现和解决程序中的问题，提高软件的质量和稳定性。

```
232     for (auto it = m_clientList.begin(); it != m_clientList.end(); ++it) {
233         if ((*it)->m_clientPort == recevicePort) {
234
235             //AfxMessageBox(_T("配对成功"));
236             SelectPort = *it;
237             strcat(buff, "与您连接");
238             sprintf(buffers2, "与%u配对成功", SelectPort->m_clientPort);
239             this->Send(buffers2, 30);
240             SelectPort->Send(buff, 20);
241             break;
242         }
243     }
```

图 7-1 断点调试图

7.2.2 弹窗调试

弹窗调试通过在代码中插入弹窗语句来输出变量的值、程序的状态以及执行流程信息，以便于开发人员理解程序的运行情况和排查问题。

```
if (bytesRead > 0) {
    if (m_clientList.empty()) {
        AfxMessageBox(_T("empty"));
    }
```

图 7-2 弹窗调试

7.2.3 集成调试

集成调试是一种在整个软件系统中调试问题的方法，通常用于调试多个模块、组件或服务之间的交互问题。这种调试方法用来查找模块之间的通信问题、接口不匹配、数据传递错误等。

在写车车通信模块时，客户端一开始无法准确接收服务器转发的消息，我利用一个完好的服务器先去与待测客户端连接，并发送，之后用一个完好客户端与待测服务器连接，经过信息发送测试，能够判断出服务器能够实现转发，但是客户端无法实现接收，之后针对客户端接收功能进行针对改进。

第 8 章 全文总结

完成课程设计的时候感受到了一阵轻松，在寒假的课程设计代码编写过程中，我首先想到的是搜寻相关背景资料，了解现有智能驾驶舱的功能与设计，查找到了 MDS 驾驶员检测系统与 V2X 车车通信技术，拓宽了我的视野。

在阅读过相关文献后，我认为驾驶舱的智能化主要体现在数据的处理上，如何获取数据并将有用的数据留下、无用的数据筛出，应该是一个智能驾驶舱的基本要求，利用数据构建合理的数据模型，并利用模型对未来发生的事件进行预测，实现推测预警。

在写车车通信的时候的主要思路是客户端 A 发送给服务器端口号、服务器筛选锁定 B、客户端 A 发送消息、服务器转发消息、客户端 B 接收消息。在这简单的逻辑思路下、通信功能从无到有再到进一步完善，总共花费了两天一夜。

由于是 Python 环境，在配置 MDS 疲劳检测所需要的 dlib 库的时候一直报错安装失败，在经历过数次文件夹路径转移后，终于实现了 pip 的成功，之后集成导出 exe 程序的时候也是一直闪退，在尝试过电脑命令行、Py installer 与 pyCharm 命令行后，实现了 exe 的生成并成功运行。

本项目设计的 V2X 车车通信存在以下问题：1.被接受方目前无法拒绝接收消息，2.所接受的数据无法导入模型，判断对方的驾驶习惯。3.V2X 通信存在容易被攻击泄露数据的风险,有较大的改进空间[6]。

通过本次课程设计，进一步强化了对 C++的动手能力，了解了现有智能驾驶舱的先进技术，拓宽了我的视野，增长了编程能力、提升了思维方式。

参考文献

[1]何灿群,殷晴,徐杰新.汽车座舱人机交互智能化设计的研究综述[J].人类工效学,2023,29(02):70-75.DOI:10.13837/j.issn.1006-8309.2023.02.0012.

[2]洪丽华，周卫红，黄琼慧．基于 Python 的数据可视化研究［J］.《科技创新与应用》，2022(33) ：36-40.

[3] 宋佳尧,尉斌,安姝洁,等. 基于 Dlib 的面部疲劳检测模型[J]. 软件工程,2023,26(12):38-40,58. DOI:10.19644/j.cnki.issn2096-1472.2023.012.008.

[4] 计算机软件开发规范:GB 8566-1988[S].

[5]张贝宁,郜健铭,王靖涵等.基于机器视觉的驾驶员危险驾驶检测系统的设计与实验研究[J].科学技术创新,2024(03):48-51.

[6]林煜.V2X 与车路协同技术的深度融合[J].内燃机与配件,2024,(01):73-75.DOI:10.19475/j.cnki.issn1674-957x.2024.01.002.