

重庆大学本科学生毕业设计（论文）

智能无人配送机器人视觉系统研究设计



学 生：李佳桐

学 号：20152465

指导教师：李敏高级工程师

专 业：测控技术与仪器

重庆大学光电工程学院

2019 年 6 月

Graduation Design(Thesis) of Chongqing University

**Design of visual system for intelligent
delivery robot**



Undergraduate: Li Jiatong

Supervisor: Senior Engineer Li Min

Major: Measuring and Control Technology and Instrument

**College of Optoelectronic Engineering
Chongqing University**

June 2019

摘要

视觉作为人类感知世界的最重要途径之一，在人类驾驶员驾驶车辆过程中发挥的作用无可替代。基于图像传感器视频信号的机器视觉同样为辅助驾驶技术、无人驾驶技术提供不可或缺的驾驶决策依据以及直观监控数据。本文结合智能配送机器人的应用场景，着重为其开发一套完整的、可独立运行的视觉系统，包括具有良好鲁棒性及实时性的道路检测技术、低延时网络多数据综合传输技术、远程监控及人类快速接管技术、实时物体检测及定位技术。

论文的主要工作及成果包括：

首先针对道路检测，本文对主流的检测方法进行了探讨并提出一套具有良好实时性的道路分割方案。本文基于种子区域增长算法创造了一种改进算法，大大增强了此分割方法的稳定性。通过在 BGR 色彩空间和 HSV 色彩空间下特定通道对图像分别处理，再将两种分割结果进行融合并多帧滤波，实验结果表明，本套道路分割方案对光照变化容忍度较高，鲁棒性及实时性满足指导车辆低速行进要求。

对于此类设计为无载人能力的自动驾驶车辆，本文设计了一套通用硬件平台下的远程低延时多数据综合传输系统，由车辆端、服务器端、后台端构成，管理人员可在后台客户端随时收回车辆控制权，保障车辆运行安全。本文利用 TCP 协议以及 JPEG/H.264 两种编码形式分别设计了局域网下和互联网下的两种图像传输模式，以满足不同使用环境要求。基于此系统，客户端利用 Yolo v3 神经网络实现对路面上行人、车辆等物体的实时检测。

关键词：自动驾驶，视觉系统，道路检测，图像传输，物体识别

ABSTRACT

As one of the most important ways for human to perceive the world, vision plays an irreplaceable role in the driving process of human drivers. Machine vision based on video signal also provides indispensable driving decision-making basis and intuitive monitoring data for assistant driving technology and unmanned driving technology. According to the application scenario of intelligent delivery robot, this paper focuses on developing a complete and independent visual system for intelligent distribution robot, including road detection technology with good robustness and real-time performance, low-delay multi-data transmission technology, remote monitoring and human rapid takeover technology, real-time object detection and location technology.

The main work and achievements of this paper include:

Firstly, aiming at road detection, this paper discusses the mainstream detection methods and proposes a robust real-time road segmentation scheme. In this paper, an improved algorithm is created based on the seed region growth algorithm, which greatly enhances the stability of the segmentation method. The image is processed separately in BGR color space and HSV color space, then the two segmentation results are fused and multi-frame filtered. The experimental results show that the proposed road segmentation scheme has high tolerance to illumination changes, and its robustness and real-time performance meet the requirements of guiding low speed vehicles.

For automatic driving vehicles without manned capacity, this paper designs a long-distance, low-delay multi-data transmission system based on general hardware platform, which consists of vehicle, server and background computer. Administrators can take control of the vehicle at any time on the background computer to ensure vehicle operation safety. Using TCP protocol and JPEG/H.264 coding, this paper designs two image transmission modes under LAN and Internet respectively to meet the requirements of different environments. Based on this system, the background end uses Yolo v3 neural network to realize real-time detection of people, vehicles and other objects on the road.

Key words: Autopilot technology, visual system, road detection, video transmission, object recognition

目 录

摘要	I
ABSTRACT	II
1 绪论	1
1.1 研究背景及意义	1
1.2 自动驾驶研究现状分析	2
1.3 自动驾驶视觉系统研究现状分析	4
1.3.1 道路检测研究现状	4
1.3.2 视频传输研究现状	5
1.3.3 物体识别研究现状	5
1.4 课题研究任务	6
1.5 论文组织结构	6
2 系统总体设计	8
2.1 系统总体架构	8
2.2 车辆端软件架构	9
2.3 实验车辆外观	12
2.4 本章小结	12
3 道路检测	13
3.1 道路检测任务与流程	13
3.2 道路检测核心	15
3.2.1 常用图像分割算法介绍	15
3.2.2 标准种子区域增长算法	17
3.2.3 图像预处理	18
3.2.4 种子自动选取	20
3.2.5 改进种子区域增长算法	21
3.2.6 阴影抑制	25
3.3 检测效果改善	28
3.3.1 空洞去除	28
3.3.2 时域滤波	29
3.4 附加检测	29
3.4.1 道路边线检测	29
3.4.2 道路边线筛选	30

3.5 本章小结.....	31
4 远程监管控制系统.....	32
4.1 远程监管控制系统架构.....	32
4.2 网络传输技术.....	33
4.3 视频压缩技术.....	34
4.3.1 JPEG 压缩技术.....	35
4.3.2 H.264 压缩技术.....	36
4.4 数据包结构.....	40
4.4.1 车辆端发送的数据.....	40
4.4.2 后台端发出的数据.....	41
4.5 服务器架构.....	42
4.6 后台端交互设计.....	46
4.7 本章小结.....	47
5 行驶实验.....	48
5.1 光线充足路段测试.....	48
5.1.1 柏油路测试.....	48
5.1.2 瓷砖路测试.....	49
5.2 潮湿水泥路测试.....	50
5.3 光线昏暗路段测试.....	50
5.4 系统耗时.....	52
6 总结与展望.....	53
致 谢.....	54
参 考 文 献.....	55
附录 A：在读期间科研经历与奖项.....	57
附录 B：部分程序源代码.....	58

1 绪论

1.1 研究背景及意义

随着世界范围内城市交通网的快速发展及城市规模的不断扩大，汽车在人们生活中的重要地位越来越无法被替代。然而，与便利一起被汽车带来的还有出行风险。依据世界卫生组织统计，在 2016 年全球范围内约有 135 万人死于马路交通事故，即每 25 秒死亡 1 人。依据我国交通部管理局统计数据，70%以上的道路交通事故与驾驶员的行为失误直接相关，90%以上的事故间接相关。车辆如何减少驾驶行为差错，甚至不依靠人类驾驶成为人们日益关注的焦点以及许多学者的研究对象。除安全角度考虑外，无人驾驶技术也能充分发挥都市交通网的通勤效率，在计算机的精准操控下，拥堵时段的车流混乱度得以降低、平均通行时间得以减小。不仅如此，目前机动车辆平均使用率仅为 1%，无人驾驶技术成熟后，连续运转的模式可以用更少的车量满足现用的运载量，节省社会资源。

所幸的是，随着电子和计算机行业的快速发展，具有完全自主行驶能力的自动驾驶技术距离全面实现越来越近。卡耐基梅隆大学始于 1984 年的 Navlab 试验车、斯坦福大学获得 DARPA (Defense Advanced Research Projects Agency) 挑战赛冠军的 Stanley 试验车等一系列世界范围内各高校的研究成果，以及谷歌公司的 WAYMO 汽车、特斯拉公司的 Autopilot 功能等各大公司牵头的研究项目都展现出了一定程度上的自动驾驶能力。

自动驾驶汽车的环境感知往往涉及到多种传感器的应用。摄像头、激光雷达、GPS、毫米波雷达、超声波雷达等都是世界范围内各个无人驾驶研究项目的常用传感器。由于汽车与人们生命安全息息相关，无人驾驶所用传感器往往要求具有较高的准确性、及时性和鲁棒性，高精度的多线激光雷达成为目前大多研究团队无法绕过的重要对象。然而，激光雷达极其高昂的售价、难以维修的特性使得依赖它的无人驾驶项目难以实现商业化推广。

图像传感器作为一种拥有几十年历史的电子器件，生产成本十分低廉；同时，现代计算机图形计算硬件的不断迭代，深度学习算法不断创新，研究人员得以从视频信号中实时挖掘出更多的信息，发展完全依靠视觉的低成本无人驾驶技术成为了可能。基于视觉的无人驾驶技术以极低的成本、不断提高的鲁棒性、准确性、及时性吸引了特斯拉等一众公司及研究团队。5G 通信的即将到来、通信技术的飞速发展更为基于视觉的无人驾驶提供了无限可能，例如低成本、低性能的小型设备也可依靠廉价的摄像头、强大的移动网络实现异地计算，拥有自动行驶功能。

1.2 自动驾驶研究现状分析

根据国际汽车工程师协会 (SAE International) 在 2014 年建立了一套关于自动驾驶的等级分类标准，并于 2016 年对其标准进行更新，新标准为 J3016_201609。该标准将无人驾驶技术依照所需驾驶员的参与度划分为 Level 0 到 Level 5 一共六个等级。Level 0 为车辆系统向驾驶员发布警告及提醒、但不参与持续的车辆控制，最高等级 Level 5 为车辆自主行驶过程完全不需要驾驶员参与或关注。目前世界上尚未有满足 Level 4 或者 Level 5 等级的自动驾驶车辆出现^[1]。

世界上最早的无人驾驶原型车出现在上世纪 80 年代。在美国国防部 DARPA (Defense Advanced Research Projects Agency) 资助下的 ALV (Autonomous Land Vehicle) 项目于 1985 年实现了在单排单行道上以 31 千米每小时的速度行驶的自动驾驶，该项目于 1986 年加入了躲避障碍功能、于 1987 年加入了白天越野行驶和夜间行驶功能。

卡耐基梅隆大学的 Navlab 项目开始于 1984 年，目前已经发展到 Navlab 11。该项目启动两年后诞生了 Navlab 1，该车改装自雪佛兰厢式货车，并配备了 3 台 Sun 工作站、视频硬件、GPS 接收器、Warp 超级电脑和一系列传感器。不过，该车受限于软件的开发直到上世纪 80 年代末才发挥全部功能，最终该车实现了顶峰速度 32 千米每小时的自动驾驶。在美国国家无人高速系统 (National Automated Highway System) 和 DARPA 的部分资助下，该项目曾于 1995 年完成了横穿美国的自动驾驶实验，试验车行驶了 4584 千米，其中 4501 千米、约占 98% 是在自动驾驶的条件下完成，该记录一直保持到 2015 年。

1987 年到 1995 年，戴姆勒奔驰和慕尼黑联邦国防军大学联合开展 EUREKA Prometheus 项目 (曾接收来自 EUREKA 成员国共计 7.5 亿欧元的资助)，并于 1994 推出了 VaMP 和 VITA-21 两台试验车，安装有具有不同焦距的 4 台摄像机。该车在巴黎的多车道高速路上行车超过 1000 千米，最高速度达到 130 千米每小时。该实验展示了全自动的变道、超车、跟车、组车队等驾驶功能。1995 年，该项目又利用梅赛德斯奔驰 S 级轿车完成往返巴伐利亚的慕尼黑与丹麦的哥本哈根的无人驾驶实验，全程行驶 1600 千米，其中以最高时速 175 千米每小时行驶了 9 千米。

2005 年由美国 DARPA 主办的 Grand Challenge 挑战赛广泛地吸引了大众的注意力。斯坦福大学的 Stanley 实验车以 6 小时 54 分钟的成绩完成了 212 千米的全程自动驾驶比赛并取得冠军。Stanley 实验车在顶部安装了 5 台激光雷达来对周围环境扫描进行 3D 建模，并辅以 GPS 系统来感知地理位置。内部导航系统应用陀螺仪和加速度计来监控车辆的行驶方向，并用来与 GPS 或其他传感器信号进行融合。一台摄像机被用来观测 80 米外的路况 (超出了激光雷达的探测范围) 以确保车辆有足够的加速空间。为了处理传感器数据，该车配备了 6 台装有 1.6 GHz 的英

特尔奔腾 M 处理器的电脑，并运行 Linux 操作系统。Stanley 试验车的成功夺冠也引起了业界对激光雷达的追捧。

谷歌公司的无人驾驶项目始于 2009 年 1 月，起初在谷歌内部秘密实验室 X Lab 进行研发。2014 年该项目推出非量产实验用原型车 Firefly，该车并无方向盘、油门踏板、刹车踏板，设计为完全自动驾驶车辆。2015 年该车完成了世界上第一次无驾驶员陪伴的公共道路无人驾驶测试。2016 年底，该项目更名为 Waymo；截至 2018 年，Waymo 项目累计实现了 1600 万千米的道路测试，测试地点涉及美国 6 个州 25 个城市。该项目在车顶配备一台激光雷达，每秒扫描数百万个点来对周围环境探测；并在车身配备超声波雷达来探测周围物体的距离和运动速度；头顶装备多个高分辨率摄像头来探测信号灯等交通标识；通过对多个传感器数据进行融合，车辆得以感知周围物体的运动信息并做出预测。2019 年 4 月，Waymo 正式公布了其量产装车计划。

特斯拉公司的 Autopilot 系统最早公布于 2014 年 10 月，其公司的 Model S 和 Model X 车型率先支持升级，并与 2015 年 10 月正式通过软件更新让消费者使用。Autopilot 是以摄像头为主要传感器、替代激光雷达的典型自动驾驶系统代表，目前对普通消费者开放了跟随车道自动转角、自适应巡航控速、自动停车入库、自动变道等功能，可在驾驶员监督下实现自动驾驶。该系统总共配备 8 枚摄像头获得 360°、正前方最远 250m 的视野范围，车身四周另装有 12 个最远探测 8 米的超声波雷达对摄像头数据进行辅助，车身正前方另外安装有 1 个最远探测 120 米的普通雷达来在大雨、大雾、沙尘暴等视野受限天气确保车辆行驶安全。2019 年 4 月初，特斯拉推出了全新一代硬件：Full Self Driving 计算机（FSD），该计算机搭载了两颗特斯拉为神经网络计算专门开发的处理器，配合特斯拉新的视觉神经网络、声纳及雷达处理软件，使得数据处理速度较上一代提升 40 倍。

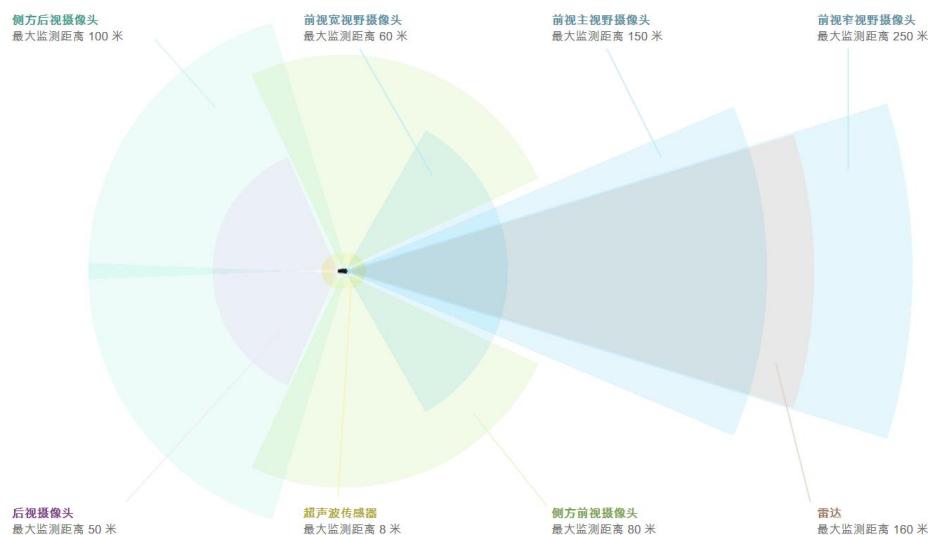


图 1.1 特斯拉 Autopilot 系统传感器示意图

除此之外，全球知名汽车厂商，如通用、奥迪、福特、沃尔沃、日产等众多公司都在积极推进无人驾驶项目研究。国内的清华大学、国防科技大学、吉林大学等高校也都在上世纪末开展过无人驾驶研究项目，百度、长安汽车等厂商也已进行过多次公路自动驾驶测试。

1.3 自动驾驶视觉系统研究现状分析

在现有的城市交通网络下，视觉系统因视觉为机器和人类共同可感知的信号量，成为了无人驾驶研究无法绕开的重要组成部分。唐智威^[2]提出了视觉感知技术在无人驾驶领域的三个应用方向：一是利用视觉为车辆定位，二是利用视觉识别道路和交通标识，三是利用视觉实现避撞。康奈尔大学的一个团队在国际计算机视觉及模式识别会议（Conference on Computer Vision and Pattern Recognition）上发表论文，他们利用一对安装在车辆挡风玻璃后的双目摄像头实现了仿激光雷达的 3D 探测效果，30 米内的 3D 物体识别率被提高到 74%，探测精度可以比拟激光雷达^[3]。随着卷积神经网络（CNN）等模型的快速发展，深度学习算法不断提升机器视觉系统的信息挖掘能力，给机器视觉的未来赋予了无限可能。

1.3.1 道路检测研究现状

基于视觉的道路检测技术是解决机器人行进问题的关键技术，为后续的路径规划等任务提供了重要支撑。目前对于道路检测算法可以分为 2 类：第一类是利用结构化道路上的特征标识，如车道线、斑马线等交通指示标志来匹配预先建立好的道路模型，进而找出道路区域；第二类是通过图像区域分割技术来提取出道路表面，这一类方法的相关研究较多。

清华大学的罗安宁^[4]以超像素作为图像的基本单元，统计了超像素的色彩、纹理细节等特征，并且以双目视觉数据和画面消失点作为提取特征的依据；随后对超像素的选定特征用回归型 Adaboost 分类器进行分类，以分割出道路区域。华南理工大学的黄勇飞^[5]通过求解光照无关角获取了路面图像的光照无关图，增强了检测算法在阴影环境下的鲁棒性，并在此基础上应用了区域增长算法和基于直方图的道路分类器完成道路路面的分割。名古屋大学的 Tomoya Fukukawa 等^[6]在 HSV 色彩空间上应用洪水填充算法以消除阴影的影响并完成路面的分割任务。

除此之外，一些利用单线激光雷达数据辅助摄像头数据的道路分割方法也被提出。Yingna Su 等^[7]先在单目摄像头图像中寻找图像消失点，然后通过激光雷达扫描到图像下方所对应的两个道路边沿点，再从消失点出发在最小代价地图上找到分别到这两个底点的两条路径，路径之间即为分割出的道路区域。另外，近年来深度学习在计算机视觉领域的飞速发展使得语义分割技术大放异彩，GCN、DFN、BiSeNet^[8 9 10]等算法都是极具代表性的成果，其中 ECCV 2018 上公布的 Bilateral

Segmentation Network (BiSeNet) 在 Cityscapes 数据集上实现了 68.4% IoU 的准确度和 105 帧每秒（在 Nvidia Titan XP 上）的速度。

1.3.2 视频传输研究现状

由于网络带宽的限制和高清视频的巨大数据量，将摄像头获取的数据直接通过无线网络进行传输尚无法普及实现。不过，由于视频信号具有极高的空间冗余性和时间冗余性，分析视频信号特性并对其进行压缩成为了许多研究人员关注的对象。其中，国际电信联盟 (ITU) 和国际标准化组织 (ISO) 是制定视频编码标准的两个主要组织。国际电信联盟电信标准分局 ITU-T 于 1988 年推出 H261 标准，该标准通过帧内及帧间预测编码、离散余弦变换 (DCT)、量化、环路滤波、熵编码等处理完成视频信号的压缩，该结构成为后续视频压缩技术的通用结构。MJPEG、MJPEG1/2、MJPEG4(SP/ASP)、H264/AVC 是目前常用视频编码技术^[11]。

搭建专有网络以实现有限距离内的无线图像传输通常可以获得较低的延时和较高的图像分辨率。西安电子科技大学的温凯林^[12]采用帧内 JPEG 图像编解码和 OFDM 调制解调技术，实现了室内环境下 16.67 毫秒延迟的高清图像传输。在产业界，大疆研发的 Lightbridge2 目前代表着业界较高水准，该系统能够在 7 公里范围内实现 1080P 画质的高清图像传输，并且控制延时在 50 毫秒内。

利用现有互联网网络传输视频数据虽然在实时性上不如专门设计的数字图传模块，不过却能使得图像传输不再受传输距离的限制，对于无人驾驶汽车的后台监管意义重大。北京邮电大学的何华丽^[13]在有线网络下提出了基于 H264 的多平台视频监控系统的设计方案。电子科技大学的冷莉洪^[14]利用海思 3518 芯片作为主控芯片对视频进行 H264 编码，再使用 RTP 网络实时视频封装技术实现了以 4G 通信技术为载体的实施视频传输系统。西安电子科技大学的彭湛博^[15]利用现有 4G 网络基础设施，对 H264 编码完成的数据用实时流媒体传输协议 RTSP 推送，实现了延时 400 毫秒左右的高清图像传输。5G 时代的到来将会大大降低网络延时，必将使基于互联网的图像传输技术性能显著提升。

1.3.3 物体识别研究现状

目前来看，在图像的基础上利用统计学进行物体识别的方法主要分为 2 类：1) 提取有效特征并进行分类；2) 建立深度学习模型进行识别分类。在卷积神经网络受限于软硬件性能未取得良好成果之前，通过先验知识建立的特征描述配合分类器进行物体识别是传统的主流方案^[16]。Dalal 和 Triggs^[17]在 2005 年 CVPR 上提出的利用方向梯度直方图 (Histogram of oriented gradient, HOG) 作为表征图像局部纹理的特征描述子的方法取得了很好的表现效果，使得大量研究员在此框架的基础上继续研究。

卷积神经网络的设计灵感本身就来自于生物体的自然视觉感知机制。上世纪

90 年代出现了现代卷积神经网络框架的原始版本，不受限于受限于当时软硬件水平及标记数据集规模，其良好的应用前景并未展现出来。2012 年，Krizhevsky 等^[18]提出了 AlexNet 卷积神经网络，并一举获得当年 ILSVRC 冠军，展现了深度学习在物体识别领域的巨大潜力。2014 年 CVPR 上，Girshick 等提出区域卷积神经网络（Region-Convolutional Neural Network, R-CNN），先获取约 2000 个候选区域并归一化所选区域图像大小，然后利用 CNN 提取固定长度特征，最后线性支持向量机分类器进行分类和回归，该框架引领了后期卷积神经网络框架的发展。目前基于深度学习技术的目标检测框架可以根据检测思路不同分为两类：1) 基于候选窗口的目标检测框架；2) 基于回归的目标检测框架^[19]。前者先选出包含目标概率较大的区域，之后对所选区域进行预测，此类框架检测精度较高，R-CNN、Fast R-CNN、Faster R-CNN 等属于此类。后者不需要候选框，把检测问题看成回归问题，预测一步到位，此类框架检测速度较快，YOLO (You Only Look Once) 系列、SSD (Single Shot Detector) 等属于此类。2018 年 3 月，Joseph Redmon 和 Ali Farhadi 提出了 YOLO v3^[20]，该模型达到了 57.9 AP₅₀ (Average Precision with IoU above 50%)，Titan X 下处理单张图片用时 51 毫秒。

1.4 课题研究任务

本课题的主要内容是设计一套用于自动驾驶机器人的完整视觉系统，主要工作可以分为如下三个部分：1) 设计了对光照变化不敏感的快速道路检测分割算法及道路边沿筛选算法；2) 设计了一套基于图像的远程实时监管控制系统，保障自动驾驶车辆运行安全，同时依托本系统的后台端对车辆视野范围内的物体进行识别及定位。依靠本套视觉系统，车辆可以有安全保障地完成自动行驶任务。

1.5 论文组织结构

本文内容安排如下：

第一章：绪论。本章综述了无人驾驶技术的发展历史及近况，并介绍了道路检测、视频传输、物体识别这三个研究领域技术现状，同时描述了本课题的研究任务。

第二章：系统总体设计。本章介绍了视觉系统整体结构和车辆端软件架构。

第三章：道路检测。本章重点描述道路检测系统的整体架构，提出了改进的区域增长算法、种子自动选择算法，利用 RGB、HSV 双色彩空间处理消除路面阴影影响，查找轮廓的拓扑结构关系消除路面上的异物影响，实现了鲁棒性较高的实时路面分割，并为车辆控制设计了道路边沿筛选算法。

第四章：远程监管控制系统。本章重点描述了远程监管控制系统的“车辆端-服务器端-后台端”和“车辆端-后台端”两种结构设计及具体实现。利用 TCP 协议

实现互联网/Wi-Fi 下的数据传输,引入心跳包检测实现断网自动重连,利用 FFmpeg 开源库完成对视频信号的 H264 编码压缩或 OpenCV 库完成对视频信号的 JPEG 编码压缩,设计组缓存机制及帧识别算法完成服务器对数据的低延时中转,选用 YOLO 神经网络模型实现对路面物体的定位与识别,同时利用 OpenCV 的 HighGUI 模块完成后台端的交互界面设计。

第五章: 行驶实验。本章展示了将整套自动驾驶视觉系统应用在实验车辆上、使得车辆拥有初步自主行驶能力的行驶实验。

第六章: 总结和展望。本章总结了整个课题的研究,概括了在视觉系统的各方向所做工作,并指出了每项工作中的不足,提出了对可能改进的方向的看法。

2 系统总体设计

本章介绍了智能配送机器人视觉系统的整体结构，阐述了整个视觉系统的工作流程，并重点说明了视觉系统在车辆端的软件架构。

2.1 系统总体架构

为了满足不同使用环境的需求，本文设计了基于互联网和基于 Wi-Fi 的两种系统方案，两种方案都能很好地保障车辆运行安全。基于互联网的设计方案中除车辆端外还增加了服务器端、后台端，由三部分共同组成人类可远程随时收回控制权的基于视觉的自动驾驶系统；基于 Wi-Fi 的设计方案取消了对互联网和云端服务器的需求，不过工作距离受 Wi-Fi 强度限制。

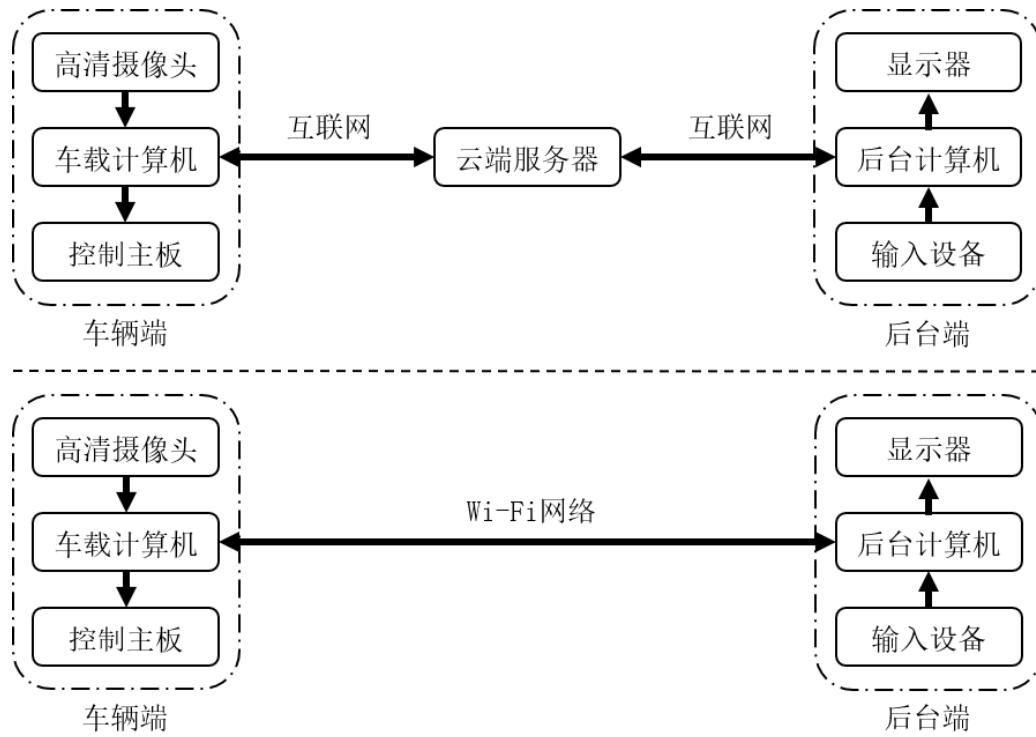


图 2.1 基于互联网和基于 Wi-Fi 的两种自动驾驶视觉系统整体结构图

车载计算机通过 OpenCV 采集高清摄像头数据，一份数据拷贝进入图像处理线程进行道路检测及车道线筛选，主线程继续利用 FFmpeg 开源库完成对图像的 H.264 格式编码或利用 OpenCV 完成对图像的 JPEG 格式编码，之后将编码后的数据、帧类型、图像处理线程的检测结果、帧间标识等数据打包成数据帧，打包好的数据利用 TCP 协议、通过互联网或 Wi-Fi 网络经中转或直接发送给后台计算机，若传输过程出错则关闭网络端口并停车尝试重新连接。

如果利用互联网传输数据，服务器端会自动区分接入计算机是车载计算机还是后台计算机，服务器接收到来自车辆的数据后根据帧类型及编码格式判断是否清空缓存队列、以保证传输实时性，每 15 毫秒对图像缓存堆栈进行检测，若队列非空则弹出一帧图像发送给后台计算机，服务器接收到来自后台计算机的数据后立即转发给车辆，若传输过程出错即关闭相应网络端口、清空缓存。

后台计算机接收到直接自车辆/中转自服务器的数据后通过查找帧分割符及数据入栈缓存完成数据分帧，之后在每帧数据中提取出视频部分并对其进行解码显示，其余部分解析还原成原始数据由用户选择显示，解码后的图像也同时拷贝进入物体识别线程，通过 OpenCV 的 DNN 模块加载 YOLO v3 模型对物体检测定位，为提高运行速度及考虑设备兼容性，检测过程调用英特尔核心显卡。

检测结果及后台操控指令打包通过互联网/Wi-Fi 网络发回车载计算机；车载计算机根据指令内容判断自动行驶还是接受操控，控制程序根据车道线信息及操控指令在 ROS (Robot Operating System) 平台上发布消息、进而通过串口与控制主板进行通信实现对车辆电机及舵机的控制，实现自动驾驶功能。

实验系统的车载计算机配置为英特尔 i3 处理器，Ubuntu 16.04 操作系统，ROS Kinetic 操作平台，摄像头最高支持 1280*720 分辨率下 60 帧每秒的图像采集；云端服务器选用阿里云旗下轻量应用服务器 2.5Ghz 单核 CPU，2G 运行内存，最大 5Gbps 的网络带宽，搭配 Ubuntu 16.04 操作系统；后台计算机为个人笔记本电脑，应用英特尔 i7-5500U 处理器，核心显卡英特尔 HD Graphics 5500，搭配 Windows 10 操作系统。各端代码统一使用 C++ 编写。

2.2 车辆端软件架构

车辆端软件直接参与车辆控制与数据的上行下行传输，为整个无人驾驶视觉系统的核心。为充分保障车辆运行安全，网络不好时人类无法对无人车辆进行监管，车辆设计为自动停止运行并尝试网络重新连接；另受限于低成本小型机器人的有限计算能力，物体识别与定位功能放置在后台端，再通过网络将结果传输回车辆参与行驶决策；基于以上两个原因，本车辆端软件需在后台端或后台端加服务器端配合下正常工作。车辆端详细架构如图 2.2 所示，为尽力保证图片结构清晰，架构图中仅显示关键步骤，一些细节被隐藏。

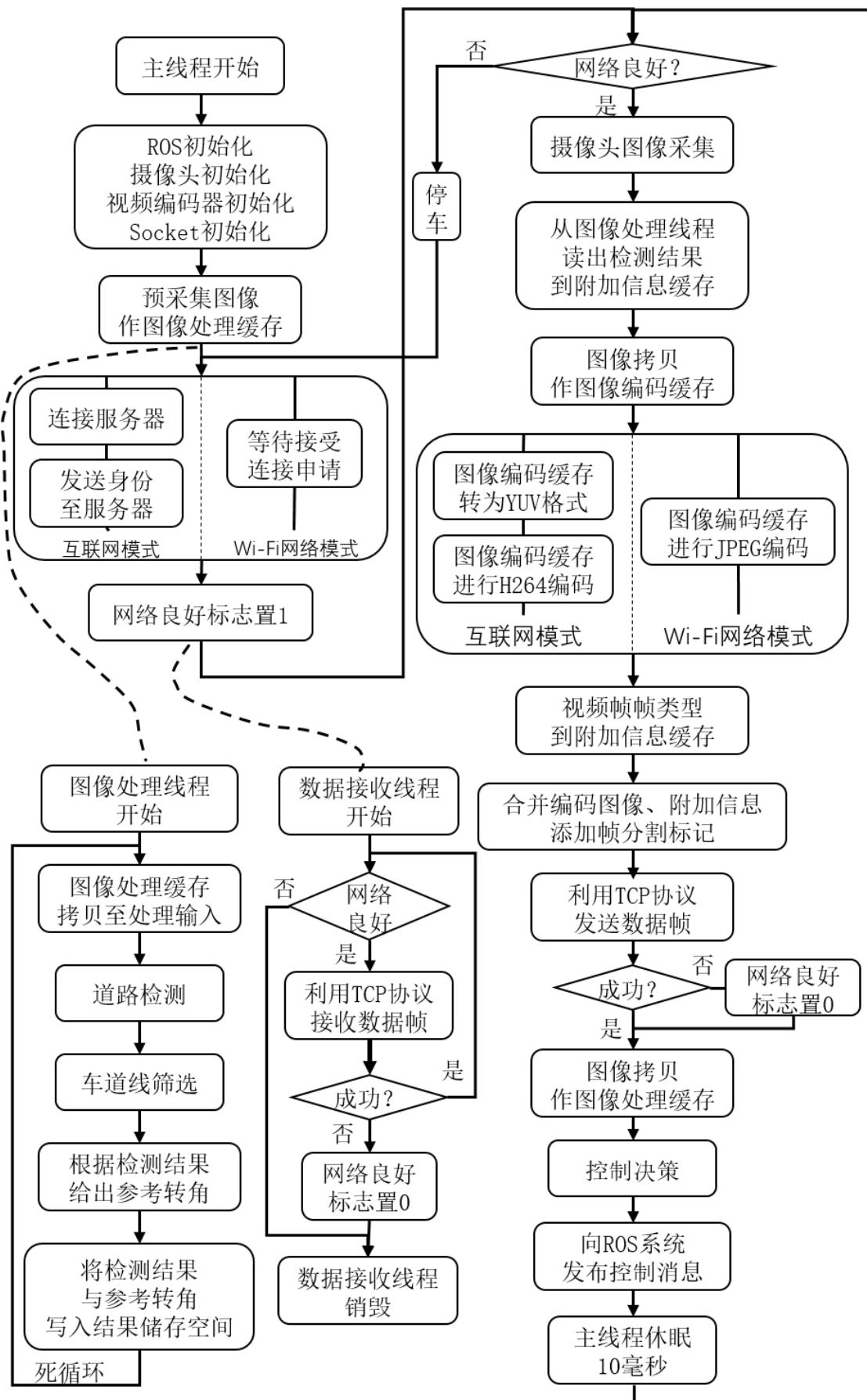


图 2.2 车辆端软件架构图

为了使车辆完成受监管的复杂自动驾驶任务，本程序设计为多线程结构、并使用互斥锁保证线程间数据交换的安全。

程序启动后会先初始化 ROS 节点、定义 ROS 消息主题，为发布消息以控制车辆行进及转弯做准备；调用 OpenCV 打开摄像头并对摄像头参数进行初始化；调用 FFmpeg 完成对 H.264 格式编码器的初始化（仅在互联网模式下）；提示用户输入服务器 IPV4 地址（仅在互联网模式下）并完成对套接字的初始化。整个初始化过程中如有步骤出错，程序会抛出异常并退出、不再继续运行。

接下来，程序会先采集一幅图像并将其送入为图像处理过程准备的处理输入储存空间。之后，单独开启一条图像线程以完成道路检测、车道线筛选、转角建议等任务（第三章详细讲解）；除非主线程结束或运算出错导致主线程结束，本线程不会退出，一直处理输入储存空间内的图像。

然后，程序的主线程在互联网模式下会根据 IPV4 地址主动连接服务器，或者在 Wi-Fi 模式下等待控制端连接。一旦连接建立成功且网络状态保持良好，程序会进入不断采集图像、发送图像、控制车辆的循环；而如果网络连接忽然断开，程序便会退出循环并立即向相应 ROS 消息主题发送停车消息控制车辆停车，然后重新尝试建立连接。

在网络状态良好的循环中，图像每采集完一次便会向专门为视频编码开辟的储存空间和专门为图像处理开辟的储存空间各拷贝一份数据；在图像处理线程中，每一次处理循环的开始会从这个专门为图像处理开辟的储存空间里拷贝出数据。因此，主线程和图像处理线程中都在对这个储存空间操作时加互斥锁以保护数据安全，确保同一时间只有一个线程对其读写：如若一个线程对其访问时另一个线程正在处理其中数据，则后来线程等待直到先占线程处理完毕释放互斥锁。使用这个专门开辟的储存空间作为中转可减小互斥阻塞几率、提高效率。

除了为其他线程拷贝图像之外，主线程中还要对循环中的每一帧进行视频编码及发送。在互联网模式下，程序会利用 FFmpeg 库对图像进行转码及 H264 格式软编码（60 毫秒），为了保证数据传输的及时性、这里采用 I、P 帧编码（无 B 帧）、零滞后等参数设置；在局域网模式下，程序会利用 OpenCV 对图像进行 JPEG 软编码，以更大的数据量（占用更大带宽）换取更短的编码时间（8 毫秒）、更好的及时性。编码完成后，压缩视频数据、该帧类型（I 帧或 P 帧）、图像处理线程的检测结果、帧间分隔符会打包成一个数据帧利用 TCP 协议发送。

每一次循环中，程序都会根据图像处理线程的检测结果、数据接收线程收到的数据做出控制决策，然后再通过向相应 ROS 消息主题发布消息来控制车辆行驶。在每一次循环结束后，主线程会休眠 10 毫秒，之后再检测网络是否良好进入下一次循环。以此来完成对整个车辆的控制。

2.3 实验车辆外观



图 2.3 实验车辆外观

2.4 本章小结

本章介绍了智能配送机器人视觉系统的整体设计，描述了基于互联网和基于 Wi-Fi 网络的 2 种系统架构以及在此架构下各个模块（车辆端-服务器端-后台端或车辆端-后台端）如何协作工作；除此之外，本章还重点说明了整个系统的核心——车辆端软件的架构，按流程介绍了受监管基于视觉自动驾驶的实现过程。

3 道路检测

本章主要讲解自动驾驶视觉系统的道路检测部分。道路检测是图像分割技术在自动驾驶领域的应用，为后续的路径规划、控制决策提供了重要依据。本系统中的道路检测技术专门针对车辆行驶应用场景设计，本章重点介绍该技术的任务、难点、整体实现流程以及其中所用到的特殊设计的算法，并在最后第五节简单介绍附加于此技术的车道边界检测技术。

3.1 道路检测任务与流程

在计算机视觉中，图像分割指把一幅图像分割成多个部分（像素集合，也被称作超像素）的过程。分割的目标是简化或改变原始图像的表征使其更具有实际意义或者更易于分析。更准确的说，图像分割是给数字图像中的每个像素点赋予一个标签，使得具有相同标签的像素点享有相同特征。

道路检测即是在整幅图像中将代表道路路面的像素点挑选出来形成集合的过程。通常来讲，根据车辆行驶环境的不同，道路检测可以分为结构化道路的检测和非结构化道路的检测。结构化道路是指具有车道线等结构性标识的道路，例如城市道路、高速公路等；而非结构化道路通常指缺乏结构性特征的道路，例如乡村土路、沙石路等。总体上，结构化道路的检测由于图像中具有易于视觉识别的结构化信息，难度要小于非结构化道路的检测。

光照变化为基于视觉的道路检测带来了难题，同样的道路环境在不同的天气、不同的时间下会在图像上反映出不同特征，车辆室外不断行驶所带来的道路环境不断变化更使得保持光照一致的条件不可能得到满足。同时，即使在城市结构化道路上，不同路面的材质、颜色、纹理细节等特征也存在较大差异。路面上的异物、标识等也容易造成分割出的路面不连续。这些难题都对道路检测技术的鲁棒性提出了较高要求。另一方面，由于机动车辆行驶速度较快、行驶安全与人身安全息息相关，道路检测技术必须保证较高的实时性。

本系统设计的道路检测技术在结构化道路和非结构化道路上都能以较低的硬件要求较好地完成实时检测任务，并且受光照变化影响较小。同时，在结构化道路上还附加了对车道边界的检测筛选。图 3.1 中的 (a) 和 (b) 分别展示了本系统道路检测技术在结构化道路和非结构化道路上的检测结果。



(a) 重庆大学校园内结构化道路



(b) 山区非结构化道路

图 3.1 本系统道路检测技术检测结果：(a) 中左图为重庆大学校园内结构化道路，右图中绿色部分为道路检测结果、蓝色线段为左右结构化约束边沿筛选结果；(b) 中左图为山区非结构化道路，右图中绿色部分为道路检测结果。

本视觉系统设计的道路检测技术及边界检测技术实现流程如图 3.2 所示。

首先，道路检测算法接收到输入图像后先对图像进行空间尺度归一化，统一变为 640*360 个像素大小，以便配合后续处理步骤的参数。之后，本算法对归一化后的图像以 13*13 的卷积核大小应用高斯空域滤波，对整个数字图像进行平滑处理，抑制由地面上（沙石路、板油路、防滑水泥路等）纹理产生的高频信号。将滤波后的彩色图像转为灰度图像，即只保留亮度信息，在灰度图像上快速求取其近似梯度图，再以 5 为阈值将其二值化作为限定区域增长的蒙版。根据车辆行驶特征，可设计出自动选取种子的算法，并以其输出的结果为种子，对滤波后的 BGR 三通道图像应用特殊设计的改进种子区域增长算法即可得到道路分割结果。

另一方面，由于 HSV 色彩空间下 S 通道（饱和度通道）图像对于路面上影子的不敏感性，本算法也在 S 通道图像上应用一次种子区域增长算法，将该通道的检测结果与 BGR 通道上的检测结果进行有条件的结合，以在路面存在阴影时抑制路面阴影的影响。

然后，在融合完的路面区域上查找所有轮廓及其拓扑关系，仅保留所有最外层轮廓，对其他轮廓全部填充，以此消除路面上杂物或强噪声引起的路面不连续。之后，在此基础上对初步的检测结果进行多帧滤波，以改善检测结果。最后再用基本的种子填充算法选中路面的连通域，消除这两步处理中产生的其他区域。到这里，

本系统道路检测算法就完成了对道路的检测、分割，得到了道路区域。

道路区域信息对于自动驾驶技术意义重大，这里提供一个应用途径：得到了道路区域即可消除大量环境干扰，只要在道路区域的灰度图像上先后应用 Canny 边缘检测、霍夫直线检测、再对这些直线进行筛选就可以得到结构化道路下起到约束车辆行驶作用的道路边界，为车辆进一步控制决策提供信息。

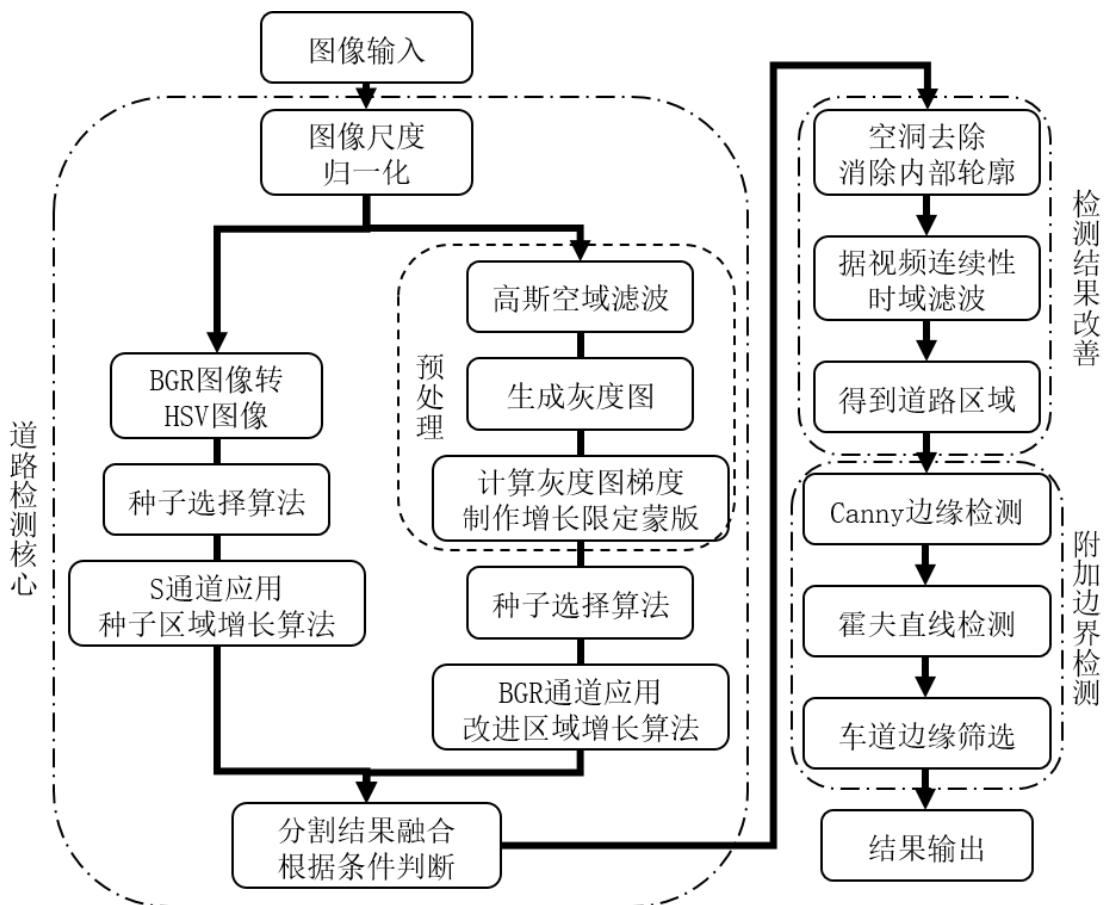


图 3.2 道路检测算法流程图

3.2 道路检测核心

道路检测依赖图像分割技术。图像分割作为计算机视觉领域的一大重要基本任务，目前已经发展出了基于不同理论的各种各样的算法。目前来看，常见的图像分割算法有：基于阈值的分割、基于压缩编码的分割、基于边缘检测的分割、基于聚类算法的分割、基于直方图的分割、基于区域的分割、基于训练的分割等。

3.2.1 常用图像分割算法介绍

基于阈值的分割：阈值分割是最简单的图像分割方法，此方法通过设定不同阈值大小将一幅灰度图像划分为不同部分。此方法的核心在于选取一个合适的阈值，通常使用最大熵方法、均衡直方图方法、方差最大法等来选取阈值。

基于压缩编码的分割：基于压缩的分割方法假定最优的分割方案会使图像数据的编码长度最短。压缩和分割这两个概念之间的联系在于分割任务就是在寻找图像中的特定模式或者其他规律以便用来完成压缩任务。该分割方法用纹理和边界形状来描述分割出的每个部分，并计算依照此次分割来编码图像所需的数据位。因此，在一幅图像的分割方案中找出编码最短的方案即完成目标。

基于边缘检测的分割：边缘检测是图像处理中一个重要、独立的研究领域。区域边界和图像边沿信息密切相关，因为在区域的边界位置像素的亮度通常会有一个跳变，所以边缘检测技术也被用作另一种图像分割技术的基础。通常通过微分算子来确定图像边沿，如寻找一阶微分算子 Prewitt 算子、Sobel 算子、Roberts 算子计算后的极值点或寻找二阶微分算子 Kirsh 算子、Laplace 算子计算后的零点。

基于聚类算法的分割：K-均值算法是一种迭代算法来将图像划分为 K 个簇。此算法的原理是首先随机选取或者用启发式算法选取 K 个簇心，然后把图像中的每一个像素归到就离它最近的那个簇心所代表的簇，接着将对每个簇的中所有的像素求取平均来得到每个簇的新簇心，最后反复归类及计算新簇心直到每个簇都不再变化，完成分割。这里的距离指像素与簇心在颜色、亮度、位置等特征上的差值的平方或绝对值。此算法可保证收敛，不过得到的可能不是最优解，最终分割效果受初始簇心的选择和簇的个数影响。

基于直方图的分割：基于直方图的分割方法相对于其他分割方法来说十分有效率，因为它仅要求对全体像素点例遍一次。在此方法中，先基于整副图像计算出直方图，然后在直方图中找出峰值点和低谷点来对图像中的簇进行定位。色彩或光强信息等可以作为直方图计算标准。通常，处理时可以通过在已经分出的簇内递归应用此直方图查找方法进一步细分簇、直到无新簇产生来改善分割结果。

基于区域的分割：基于区域的分割可以分为以增长为基础和以分裂为基础的两种方式。分裂-融合算法是基于分裂的方式，该方法应用一幅图像的四叉树分割。该算法从代表整副图像的树根开始，如果发现一部分区域中有不同性质，则将其分裂成四个子区域，并不断如此。相反地，如果检测到任意两个相邻区域具有相同的性质，则将这两个融合形成一个新的区域。整个处理过程不断迭代进行，直到不会有新的区域分裂或融合。区域增长方法主要基于图像中相同区域的相邻像素点拥有相近值的假设。通常的做法是将一个像素点与其相邻的点比较，如果满足了相似评判标准，就把该像素点和其相邻点归为一个簇内。在此方法中，相邻评判标准的确定十分重要，并且分割结果易受噪声影响。

基于训练的分割：大多数的分割方法都仅仅基于图像中像素的色彩信息，人类在观察图像进行分割时却用到更多的知识。随着算法的不断提升及硬件性能的不断突破，全卷积神经网络（FCN）通过深度学习训练在像素级的语义分割上表现出

色。现有的通过训练的分割方式需要用到大量人工标记的训练集花费大量时间训练参数，再通过在神经网络中前向传播对输入图像进行分割，属于监督学习。

3.2.2 标准种子区域增长算法

种子区域增长法是一种基于区域的图像分割方法。本方法在图像上选取一个种子作为输入，这个种子标记了要被分割出的物体。在固定值比较下，该种子点周围的像素点与这个种子点的特征值做差得到差值 δ ，如果这个差值 δ 小于事先设定的阈值则将这个像素点归到该区域，通过不断比较区域周围的点与种子点，选中区域不断增长，直到不能增长，完成区域分割。在相对值比较下，每一次增长过程中的比较对象为区域内和区域外的两个相邻点，而非与种子点比较。

同时，本算法在判断相邻点连通性上通常选用四连通或八连通，两种方式填充效果对比如图 3.3 所示。图中的红色点代表种子点，黄色点代表其各自连通意义下的相邻点，黑色点为不满足增长条件的点，白色部分为满足增长条件的点，绿色部分代表满足增长条件并应用种子区域增长法后选中的区域。

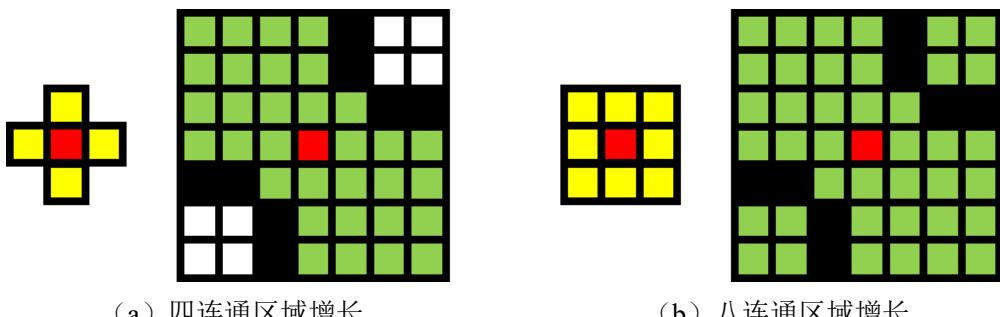


图 3.3 四连通与八连通对比

通常情况下，对于 RGB 三通道彩色图像，处理时会分别选取三通道的值作为分割比较的特征值。当前算法要处理的点的空间坐标为 (x, y) ，该点相邻于已知在选中区域内的空间坐标为 (x', y') 的点，当它们满足公式 3.1、公式 3.2、公式 3.3 的关系时，则 (x, y) 点被归入选中区域。

$$R(x',y') - T_R \leq R(x,y) \leq R(x',y') + T_R \quad (3.1)$$

$$G(x',y') - T_G \leq G(x,y) \leq G(x',y') + T_G \quad (3.2)$$

$$B(x',y') - T_B \leq B(x,y) \leq B(x',y') + T_B \quad (3.3)$$

式子中， $R(x,y)$ 、 $G(x,y)$ 、 $B(x,y)$ 分别代表点 (x, y) 处的 R、G、B 三通道的值。 T_R 、 T_G 、 T_B 分别代表了实现设定的三通道差值阈值。

对于灰度图像来说，通常将亮度值作为分割比较的特征值。类似地，当算法当前处理点与选中区域内的相邻点满足公式 3.4 时，当前处理点被选中。

$$I(x',y') - T_I \leq I(x,y) \leq I(x',y') + T_I \quad (3.4)$$

式中， $I(x,y)$ 代表 (x, y) 处亮度值， T_I 为事先设定的亮度阈值。

3.2.3 图像预处理

种子区域增长算法由于在像素级别上进行比较，对临近像素间的差值变化敏感，区域分割过程容易受噪声的影响。另一方面，由于地面不平整花纹的存在，阳光强烈时、路面图像临近像素间的三通道值变化剧烈。所以，为了提高检测算法的稳定性，在应用种子填充算法之前，对图像进行滤波十分必要。

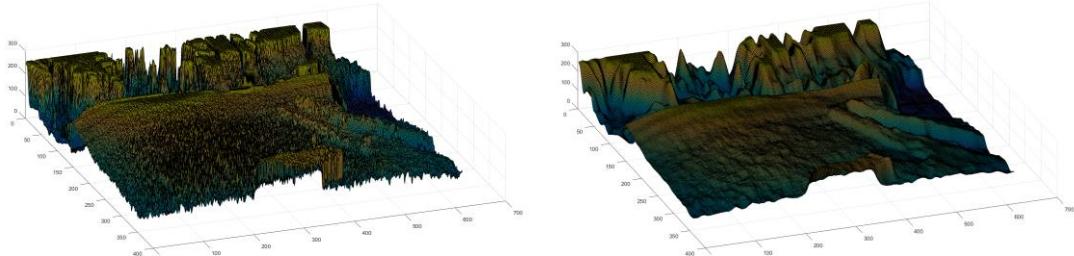
利用连续二维正态分布函数作为加权函数，可以求得任意大小的空间高斯滤波卷积核。式 3.5 中， x 、 y 代表像素距离卷积核中心的距离， σ 代表标准差。

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}} \quad (3.5)$$



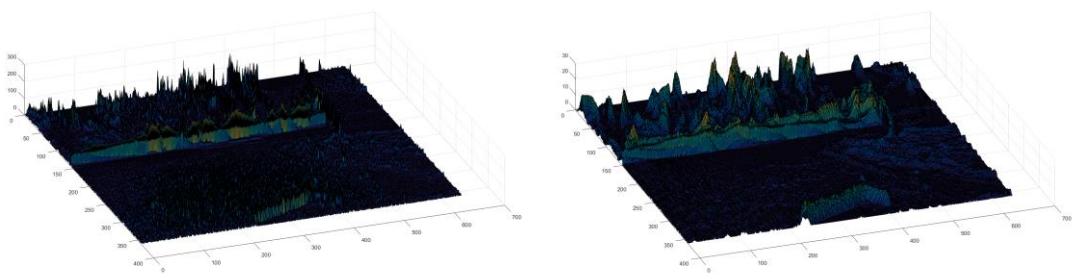
(a) 原图

(b) 高斯滤波后图像



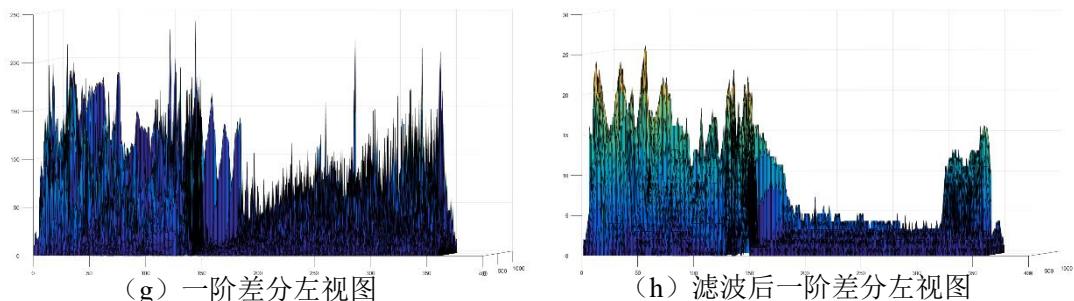
(c) 三维亮度图

(d) 滤波后三维亮度图



(e) 一阶差分图

(f) 滤波后一阶差分图



(g) 一阶差分左视图

(h) 滤波后一阶差分左视图

图 3.4 卷积核大小 13*13，方差 6.3 的高斯滤波处理前后数据

图 3.4 展示了高斯空域滤波的效果。a、b 两幅图分别展示了作者在重庆大学校院内拍摄的路面原始图像和经过高斯滤波处理后的图像。c、d 两幅图分别对应原图与滤波后图像经过彩色图像到灰度图像转换、将亮度映射到 Z 轴所生成的三维图像。从这两幅图中，可以清楚地看出，原图中路面所对应的部分毛刺众多，而滤波后的图像在路面所对应的区域较为平缓。e、f 图像是由灰度图像经“临近行、临近列直接差分再绝对值相加”直接得到的一阶微分图像，而 g、h 则分别是它们的左视图，以更方便看出 Z 轴上的波动差距。

由于种子区域增长算法的原理即为相邻点做差，与上述实验中的求取一阶差分的方法有一定程度上的对照性。可以看出，在未经滤波的图像中，根本无法通过相邻点差值(0~240 范围波动)将路面、或是任何一个感兴趣区域与周围区分开来，而经过高斯滤波的图像，在路面连续区域差值明显(0 到 5 范围波动)减小。这使得种子区域增长算法具有了理论上的可行性。

此外，在进行种子区域增长过程之前，还应该制作全差分掩膜(或称蒙版)作为边界以限制区域增长。否则只要横、纵一个方向满足条件便能连通区域。

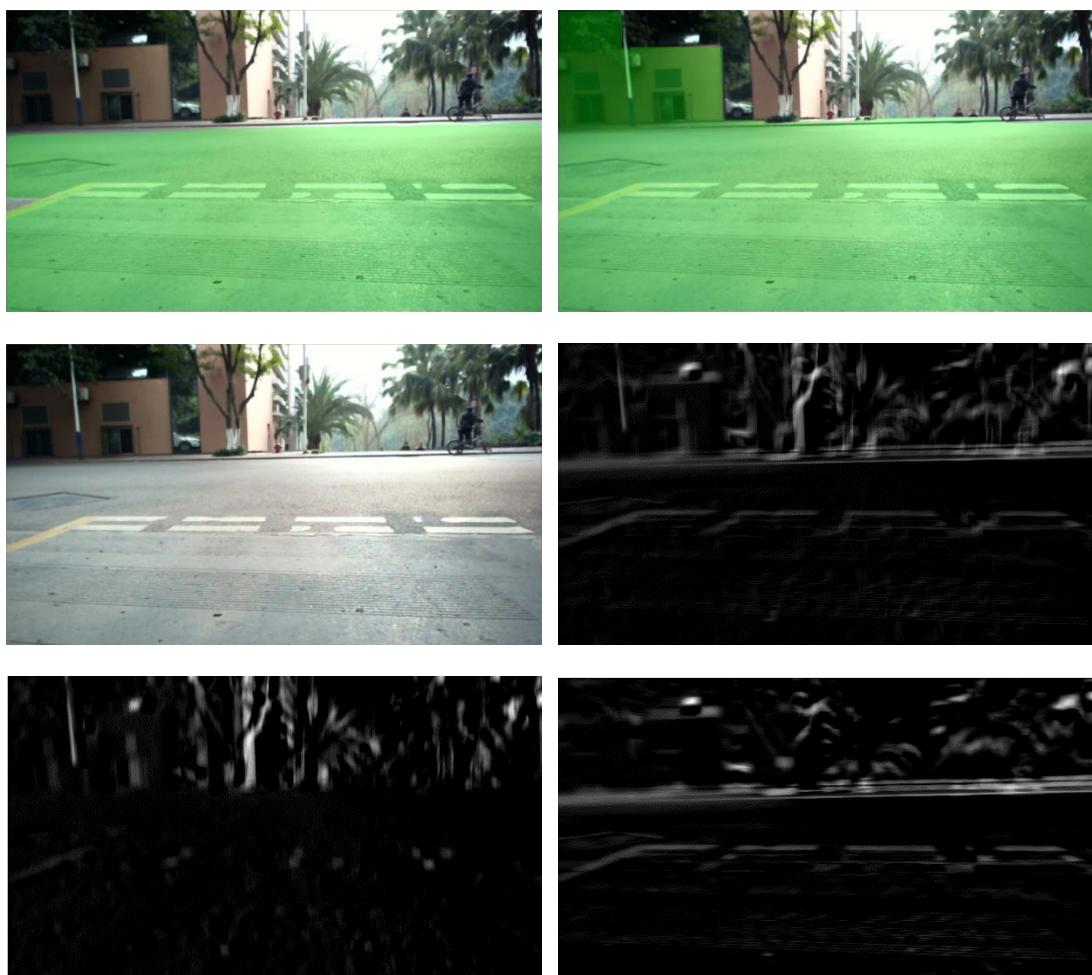


图 3.5 定义种子区域增长算法边沿的必要性：第一行左图为使用掩膜分割、右图为未使用掩膜分割；第二行左图为原始图像；其余依次为一阶全差分、横向差分、纵向差分图像。

3.2.4 种子自动选取

对于自动驾驶技术来说，道路的检测应该由车辆自身全自动完成、而不需要人为干预。不过种子区域增长算法的第一步必须先人为输入一个种子点，这对于自动驾驶过程来说显然是不现实的。幸运的是，现实世界映射到二维图像中的每一部分都是具有其意义的，例如人类就知道从“脚下”延伸出去的区域是路。应用同样的道理，处理时可以选取数字图像中下方的区域作为区域增长算法的种子点。不过，由于本算法对于单个像素敏感的特性，一旦初始种子点选取在了高噪声影响的点、或者地面斑驳中的点等特殊位置，区域增长就会被限制在这一小块特殊区域、导致对整体路面分割失败，如图 3.6 所示（红点为种子位置）。

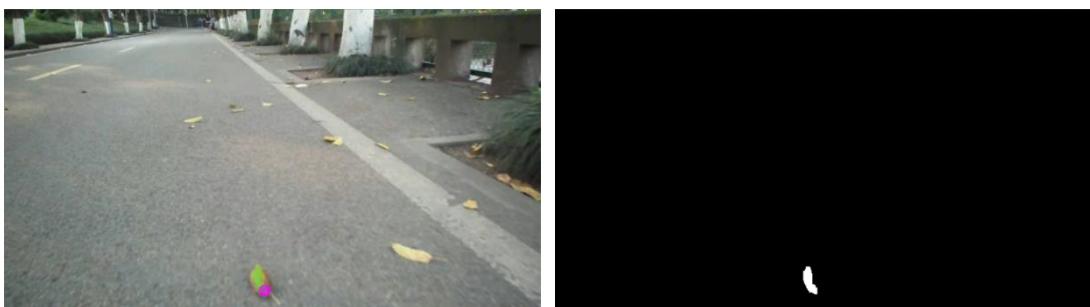


图 3.6 固定种子时路面杂物导致分割失败

为解决这个问题，处理时可以选取画面下方的一片区域作为采样区，并应用其统计信息在其中选取合理的种子点。在校园内道路上的大量实验测试表明，本套种子自动选择算法效果良好，如图 3.7 所示（红点为计算出的种子）。



图 3.7 应用自动种子选择算法后的分割效果

本种子自动选择算法设计如下：首先，在画面中下方，即车辆的“脚下”，选取一块正常情况下不会被遮挡的区域作为种子选择区域（图 3.7 中下方半透明白色矩形区域）。下一步，对这片区域内的图像进行降采样，横纵分别每四个像素点采集一次，在对选择结果影响不大的前提下减小数据量。经过了空间降采样处理后，再将此区域的 BGR 彩色图像转换为灰度图像，仅保留图像亮度信息、降低数据的维度，方便利用直方图直接分析。对这些经过降采样、色彩转换后的像素点进行统

计、计算每个亮度值出现的频率(图 3.7 种子选择区域像素点亮度的直方图如图 3.8 所示)。最后，直方图的峰值说明该亮度出现次数最多，更大概率选择区域的整体特征，所以找出直方图的峰值，并把它所对应的任意一个像素点定位出来，就完成了适用于种子区域增长算法的种子自动选择。

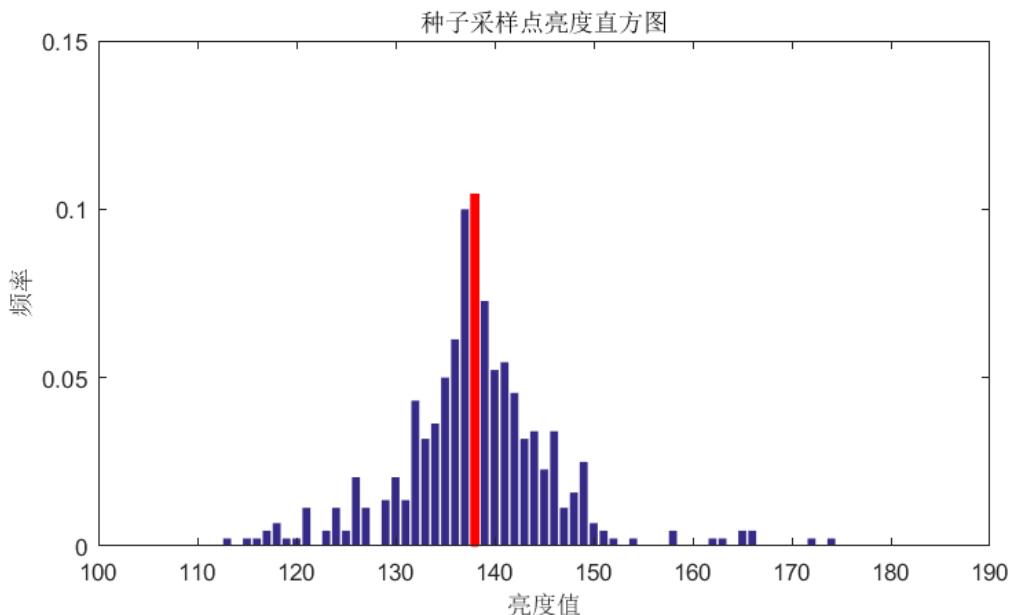


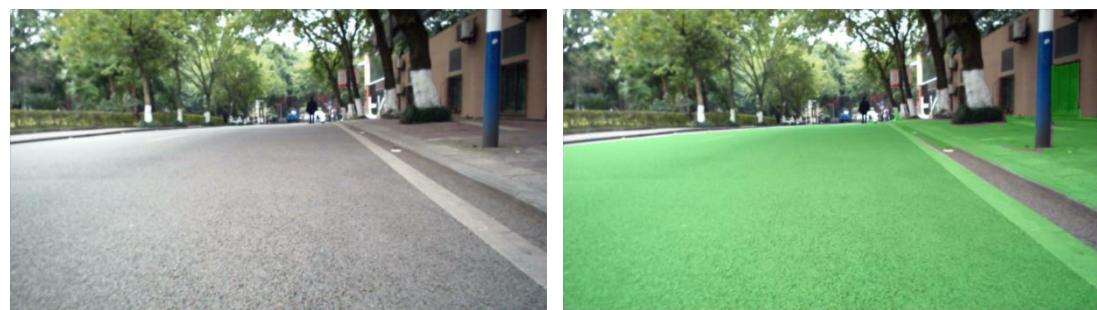
图 3.8 种子选择区域像素亮度直方图

3.2.5 改进种子区域增长算法

传统种子区域增长算法在区域边界清晰时多数情况下可以较好地完成图像分割任务。但是，由于种子填充算法在原理上是在像素级别进行相邻点比较及区域扩张，尽管预处理阶段用全差分掩膜标记了区域增长的边界，但是当边界不连续、或者有缺口的时候，该填充算法也会不可避免地会“溢出”进入并不属于当前区域的相邻区域，从而造成分割失败。

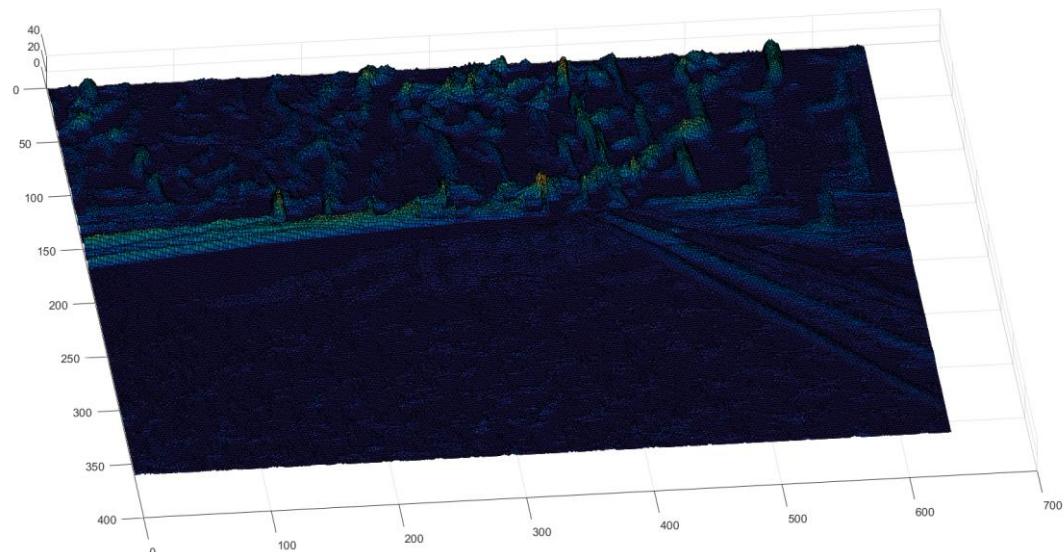
另一方面，现实世界中采集到的图像往往情况复杂。相近色异物的遮挡、阳光在地上反射造成过曝、远处场景模糊等因素都有一定几率在两个本不应该连通的相邻区域内架起一座“桥梁”，跨域区域边界，导致区域分割的失败。哪怕仅有一个像素宽度的“桥梁”也足以将选中区域引入到相邻区域，所以，单纯对原图像进行空间尺度变换、以期减小边界缺口宽度并不能保证区域的增长不会溢出界限。

如图 3.9 所示：(a) 为在重庆大学校园内采集的图像；(b) 为应用标准种子区域增长算法后的检测结果（已对图像进行预处理），从图片可以很明显地看出，标准算法错误地将分割区域漫延到了可行驶车道外；(c) 显示了图像亮度的全差分计算结果并以三维图显示，在图中可以看到道路的尽头处出现了一条通往车道路外的“沟壑”，正是这个“缺口”导致分割失败；(d) 中指出了区域增长溢出路线。

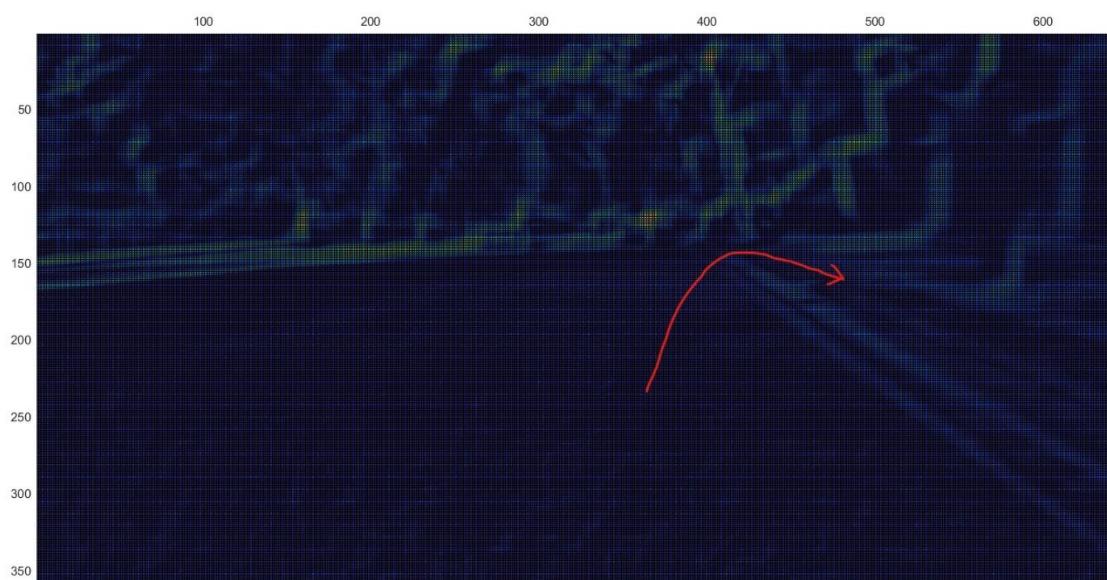


(a) 原图

(b) 标准种子区域增长算法处理图



(c) 亮度全差分三维图



(d) 区域增长溢出路线

图 3.9 标准种子区域增长算法失效说明

由于标准算法存在着上述不稳定性，本系统应用的种子区域增长算法对标准算法进行了改进。改进的中心环节便是由点向前推进改为由边向前推进：标准的种子区域增长算法在向上下左右四个方向推进时都是将下一个像素点与本像素点比较，满足条件则把其归入选区；改进算法在上下左右推进时都是将与前进方向垂直的一段像素和其前面一段像素进行比较，这一段的像素全都满足条件才将当前点归入选区。如图 3.10 所示，红色的为种子点，在标准算法中，若种子下一步要向左行进到左边绿色格，则需要比较左边绿色格与黄色格；在改进算法中（以直径五个像素为例），若种子下一步要左行进一格，则需要将左边五个绿色格与其左侧对应五个黄色格分别比较，五个比较结果都满足条件才能行进。

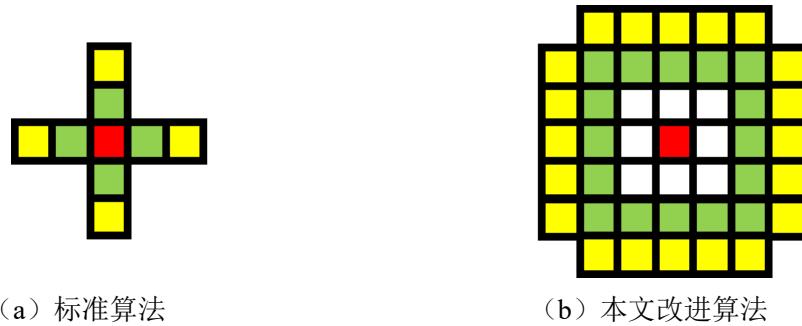


图 3.10 种子区域增长算法感受域对比

可以把标准的种子区域增长算法想象成：从种子点开始注水，水在待分割区域流动，只有在遇到区域边沿的时候才被拦截，水漫之处都是被选中区域；如果边界不完整，水流便会从缺口处溢出。相对应的本文改进种子区域增长算法可以想象成：在种子点处放一个球，球在待分割区域滚动，遇到区域边沿时球的运动会受阻，球能滚动到的地方都是被选中区域；只要边界缺口小于球的半径，球便不会滚出区域。

图 3.11 对二值图像的处理结果简单说明了标准种子区域增长算法与本系统改进算法的区别。注意应用本改进算法后还需膨胀运算，以弥补边缘缺失。

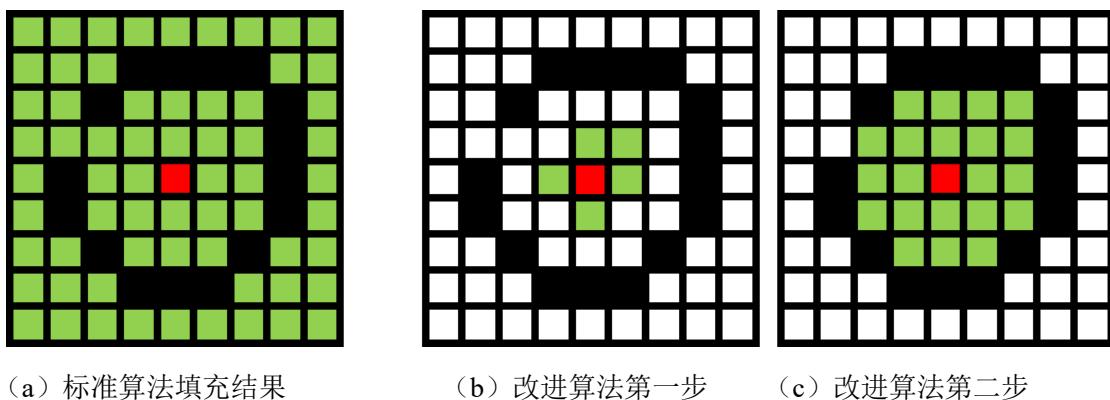


图 3.11 标准算法与改进算法（直径三个像素）填充对比

在编程实现本改进算法时，采用行扫描加堆栈储存起始点的方式完成递归填充过程。滚动直径为五、相对值比较下用 C++ 实现该算法的流程如图 3.12 所示。

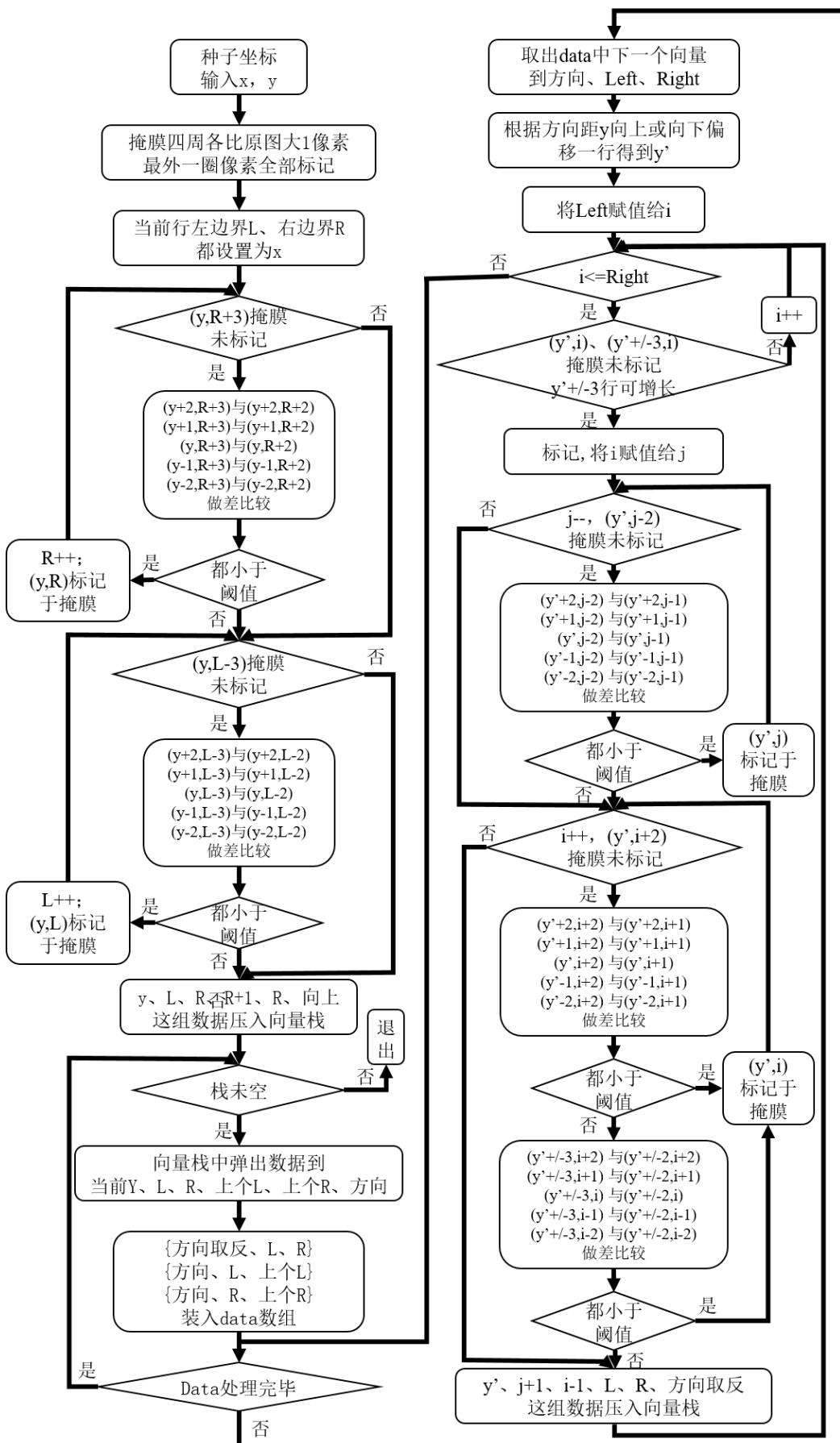


图 3.12 C++编程实现改进种子区域增长算法流程图

通过如上程序流程实现本系统改进种子区域增长算法后，分割出区域的边界位置受本算法“滚球”影响会有“滚球”半径宽度的像素点未填充，因此再对分割出的区域应用具有与“滚球”相同形状内核的形态学膨胀运算，复原边界信息。通过新改进算法进行如上处理后，图 3.9 中原图的重新分割结果如图 3.13 所示。



图 3.13 改进种子区域增长算法（直径为 5 个像素）处理效果

3.2.6 阴影抑制

在室外场景下，光照环境变化是影响计算机视觉算法性能的一大难题。对于无人驾驶视觉系统来说，增强道路检测算法在不同光照环境下的鲁棒性十分重要。

通常来讲，由于电子感光器件都是以 RGB 三色滤光镀膜加光强传感器实现对彩色世界的感知，因此 RGB 色彩空间由于其面向硬件的特性成为数字图像处理领域应用最广的色彩空间。图 3.14 中的 a 为重庆大学校园内采集的带阴影的彩色路面图片；b、c、d 作为 a 的 R 通道、G 通道、B 通道图像，分别展示了图像的红色分量、绿色分量、蓝色分量信息，越亮的地方代表该值更大。



(a) 原始彩色图像

(b) R 通道



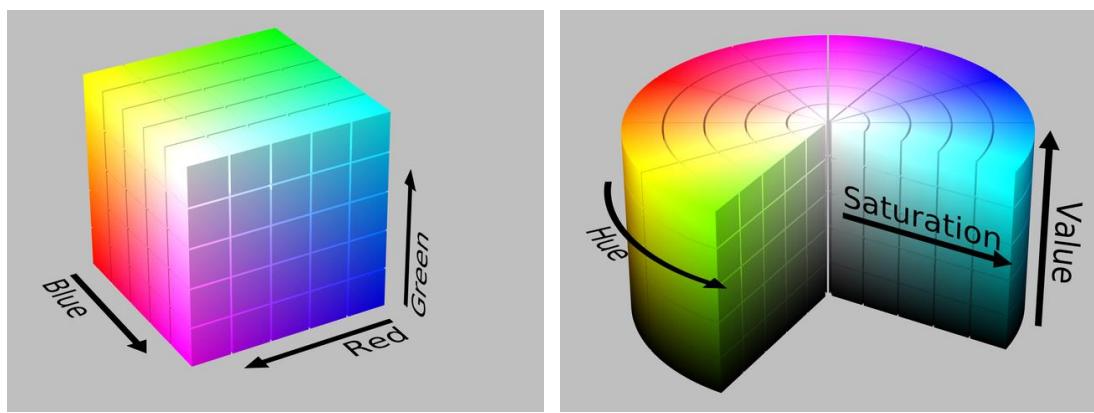
(c) G 通道

(d) B 通道

图 3.14 RGB 色彩空间

HSV（即色调、饱和度、亮度：hue、saturation、value）色彩空间是在 20 世纪 70 年代由计算机图形学研究者们研发出的色彩模型，该色彩模型与人类视觉系统感知颜色的方式密切相关。在这个色彩模型中，色调被安排为一个 0 到 360 度的圆环，其中 0 度的位置表示红、120 度的位置表示绿、240 度的地方表示蓝。饱和度被安排为 0 到 1 之间，用来表示颜色接近光谱色的程度；任何颜色都可以被视为光谱色与白色混合的结果，当白光成分为零时，饱和度为 1，此时颜色最鲜艳，当颜色为完全灰白时，饱和度为 0。亮度也被安排为 0 到 1 之间，用来表示明亮程度；当亮度为 0 时，表示完全黑暗，当亮度为 1 时，表示最亮。

RGB 色彩空间和 HSV 色彩空间在几何体中的表示如图 3.15 所示。在表示 HSV 的圆柱体中，圆柱的中心轴表示灰色，其底端表示黑、顶端表示白；以之对应的是表示 RGB 的立方体中从原点引出的对角线，同样表示灰度。在 HSV 圆柱体的外弧面上的颜色都是纯光谱色，越靠近底部亮度越暗、反之亮度越亮。



(a) RGB 色彩空间

(b) HSV 色彩空间

图 3.15 几何体演示色彩空间

关于 RGB 色彩空间（三通道值都归一化到 0 到 1 之间）向 HSV 色彩空间（H 通道取 0 到 360，S、V 通道取 0 到 1）的转换，在转换时首先需要依照式 3.6，式 3.7，式 3.8，先计算 M，m，C 三个值。

$$M = \max(R, G, B) \quad (3.6)$$

$$m = \min(R, G, B) \quad (3.7)$$

$$C = M - m \quad (3.8)$$

之后，就可以先依照公式 3.9，3.10 计算 H 通道的值。

$$H' = \begin{cases} 0 & , if \ C = 0 \\ \frac{G-B}{C} + 0 & , if \ M = R \\ \frac{B-R}{C} + 2 & , if \ M = G \\ \frac{R-G}{C} + 4 & , if \ M = B \end{cases} \quad (3.9)$$

$$H = 60^\circ \times H' \quad (3.10)$$

最后，先后依照公式 3.11 和公式 3.12 计算 V 通道以及 S 通道的值。

$$V = M \quad (3.11)$$

$$S = \begin{cases} 0 & , if \ V = 0 \\ \frac{C}{V} & , otherwise \end{cases} \quad (3.12)$$

根据以上公式将图 3.14 中 a 图从 RGB 色彩空间转换到 HSV 色彩空间，其结果显示在图 3.16 中。为了方便显示，已将 H 通道从 0 到 360 线性变换为 0 到 180，S、V 通道从 0 到 1 线性变换到 0 到 255。

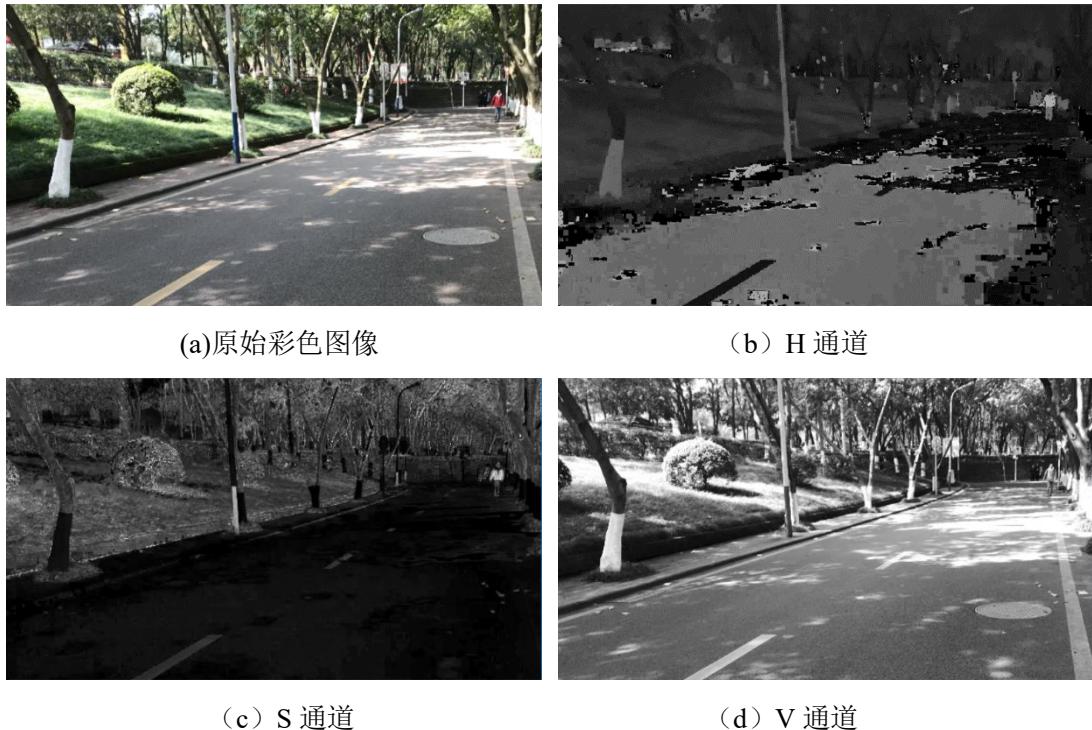


图 3.16 HSV 色彩空间

通过大量实验后，结果清晰地显示，在 HSV 色彩空间内，S 通道对于阳光下的阴影并不敏感，也就是说有阴影的路面区域和无阴影的相邻路面区域具有相近的饱和度，因此可以应用此特性实现算法对阴影影响的抑制。对图 3.14 中的 a 图像进行检测，其在 RGB 三通道下和在 S 通道下的结果对比如图 3.17 所示。（这里为直观对比两种检测的区别，两个检测结果都未经过去空洞处理）



图 3.17 左图为 RGB 三通道下的检测结果，右图为 S 通道下的检测结果

3.3 检测效果改善

按前述算法处理图像在大多数情况下可以顺利完成对路面区域的检测。不过，有些时候，由于地面上存在大块异物（如：树叶、井盖）或强噪声（高斯空域滤波未消除掉）等元素，路面的分割结果会存在“空洞”以及种子区域放置错误导致的彻底选错等现象。路面存在“空洞”的情况较为常见，彻底选错概率很小。

3.3.1 空洞去除

地面标志、异物、或阳光直射下的暗纹等存在使得区域增长算法分割出的连通域中间存在一些空洞，而这些空洞有时会对后续处理带来灾难性的影响，比如，后续希望在选出的区域进行边缘检测时，则这些空洞有可能相对于被放大的噪声。针对这种现象，在处理过程中可以采取轮廓查找算法，找出图像中的所有轮廓以及它们之间的拓扑学关系。然后依据轮廓的继承关系，对所有轮廓的最外层父级轮廓进行向内填充，便可以消除这些空洞，如图 3.18 所示。

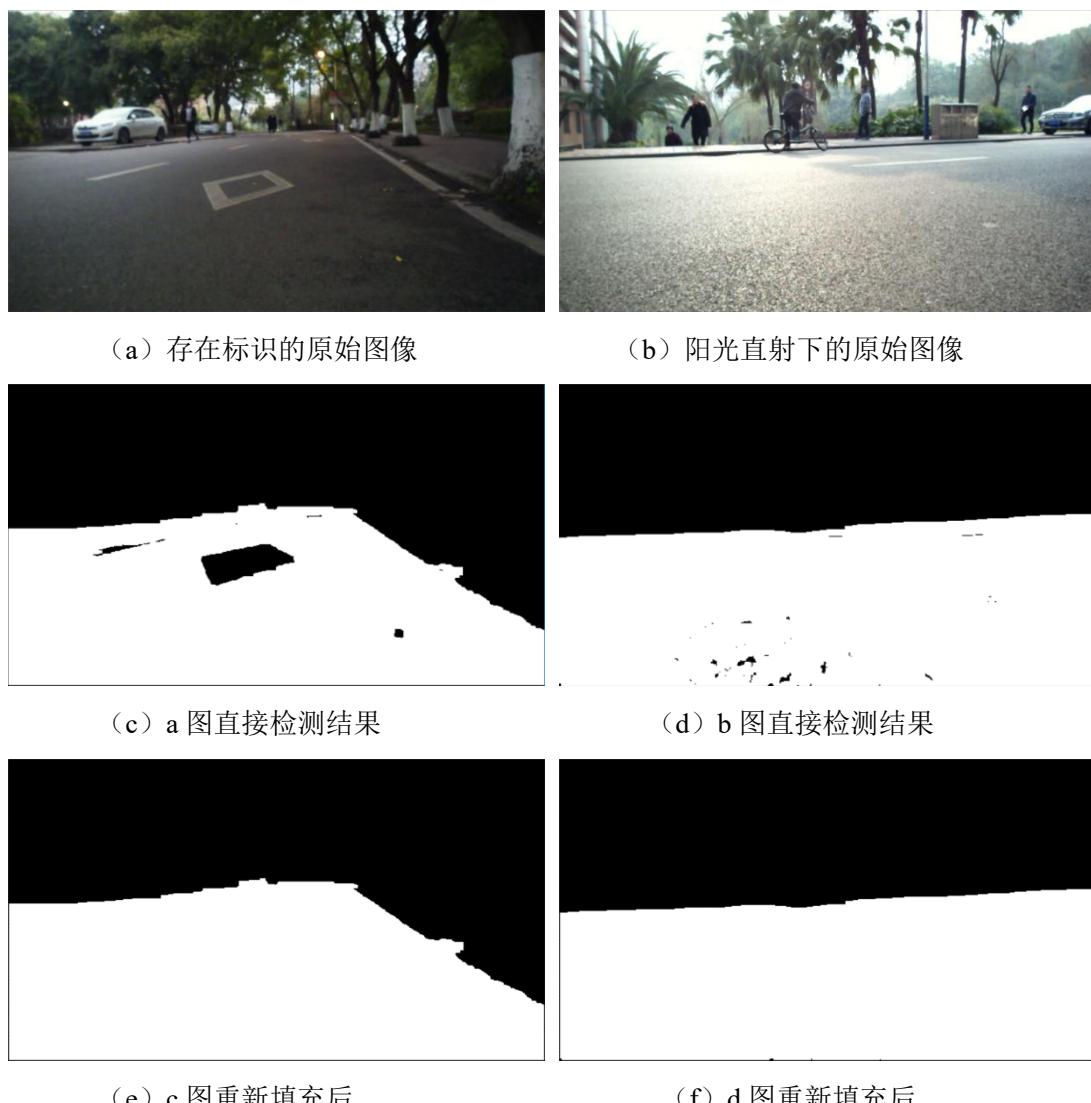


图 3.18 空洞去除处理

3.3.2 时域滤波

由于本检测算法是自动驾驶视觉系统的一个组成部分，输入的信号是连续视频信号，通常相邻前后帧之间变化不大。利用这种时间冗余度、图像相似性，可以对每一帧的检测结果进行时域滤波，再输出最终结果，可以使检测结果更稳定。图 3.19 展示了某一帧出错后由时域滤波修正的过程。**a** 图为当前帧原始图像，**b** 图为对原始图像直接检测结果；**c** 图为将当前帧送入时域滤波器后输出的结果，**d** 图为对经过滤波的结果再二值化后当前帧最终检测结果。

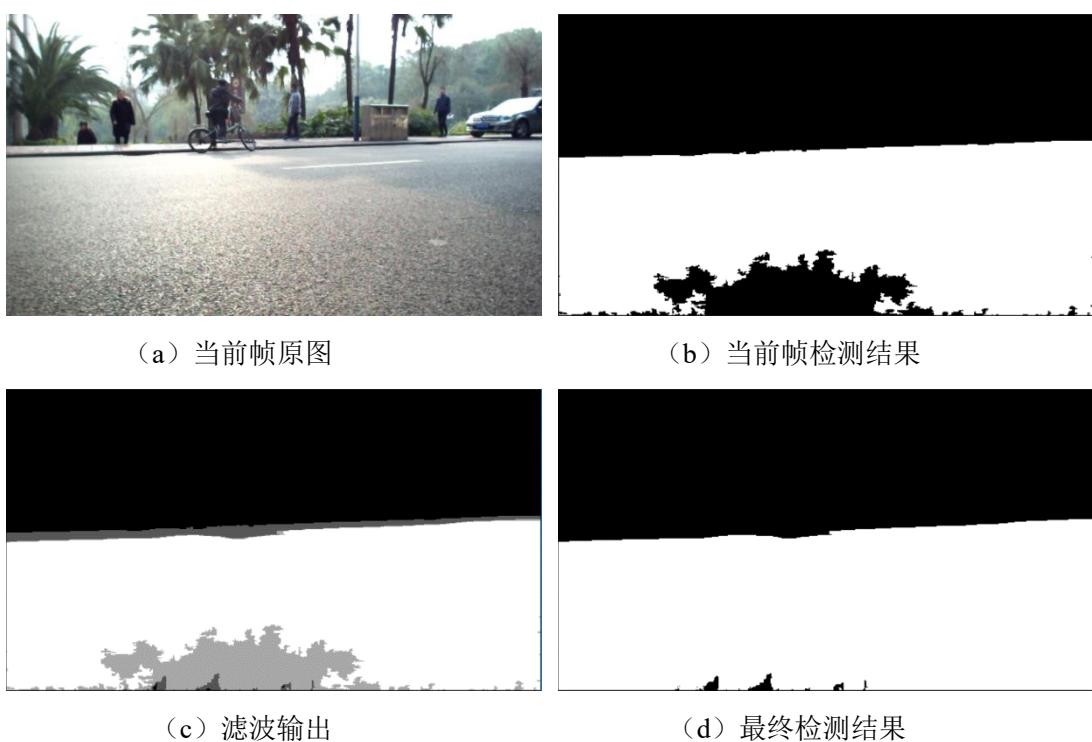


图 3.19 时域滤波效果图

3.4 附加检测

有了本系统设计的稳定道路检测结果，可以较准确地将后续算法关注区域设定在车辆可行驶道路区域。比如，在此道路检测技术的支持下，本系统可以快速将限制车辆行驶的区域边沿信息检测并筛选出来，以帮助车辆行驶决策。

3.4.1 道路边线检测

边缘检测技术作为数字图像处理的重点研究领域，目前已经发展出了很多成熟的算法。其中，比较常用且有效的算法便是 Canny 边缘检测。其检测步骤可归纳为五步：1) 应用高斯滤波器来平滑图像；2) 计算图像的亮度梯度；3) 应用非最大抑制法来消除边缘检测的杂散响应；4) 应用双阈值来找出潜在边沿；5) 用滞后效应来追踪边沿，即抑制所有未与强边沿相连的弱边沿。

在之前已经检测出来的路面区域应用 Canny 边缘检测即可探测出所以可能为

路面边沿的像素点，同时避免杂乱场景对探测的干扰。在这之后，接着在这些边沿上应用累计概率霍夫变换，即可识别出潜在的直线线段。本道路边线检测算法结果如图 3.20 所示。其中 a 图为车辆在重庆大学校园内测试时采集的真实图片，b 图为道路检测算法检测出的道路区域的灰度图，c 图为在道路区域绘图图像上应用 Canny 边沿检测的检测出的边界效果，d 图上的粉红色线段即为应用概率霍夫变换所识别出的直线线段。

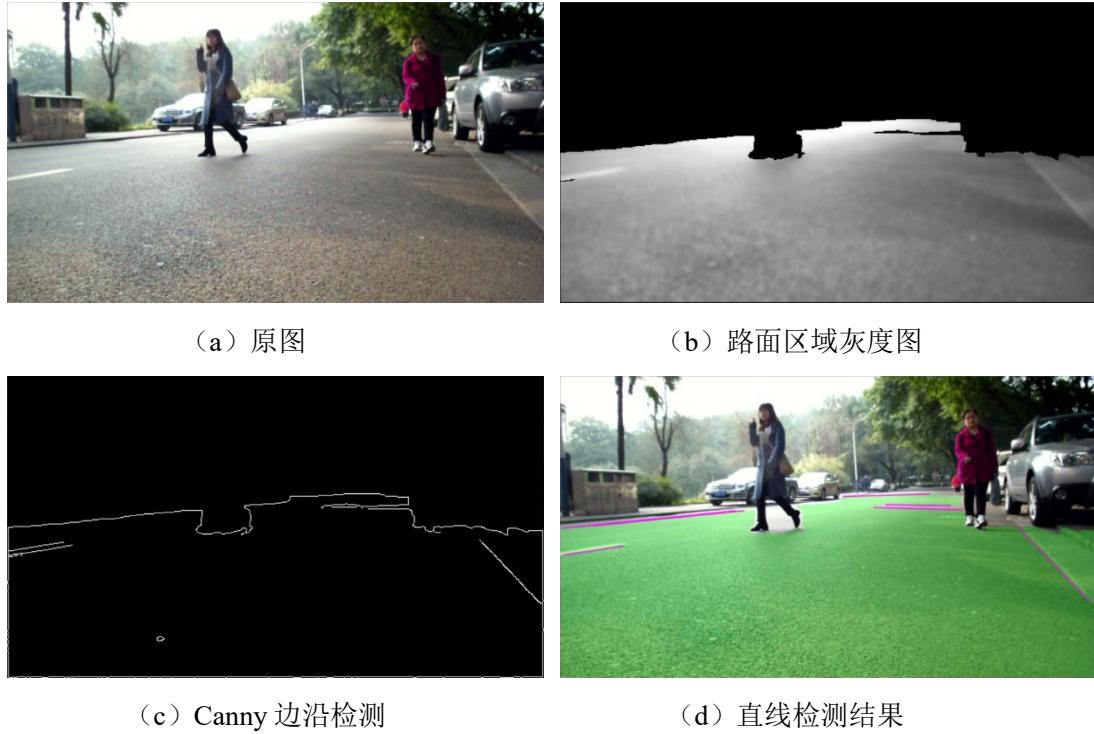


图 3.20 可能边线检测结果

3.4.2 道路边线筛选

单纯的在道路区域进行直线识别仅仅能实现将所有的结构化道路直线指示信息都找出来，不过并没有办法让车辆得知那条直线才是真正能够指示其行驶的标志线。而且，由于路上情况复杂，一些路面上的行人、车辆遮挡的边沿也有可能被直线识别算法识别为直线，而这些直线段的位置、大小都是随机出现的。为了能够给车辆明确的控制依据，本系统设计了一套边线筛选算法。

算法核心为用椭圆从画面中下方开始逼近待选线段，最先触碰到的线即为约束边线，其实现过程为：先将所有检测出来的以端点像素坐标表示的直线线段做坐标变换，将图像中下方设置为新坐标系的原点；然后求取这些线段所在直线的一般式方程，并对参数归一化（防止后面计算过程数据溢出），利用椭圆方程与直线方程求出与直线相交的最小椭圆轴长（长轴与短轴比事先设定）与相交点的坐标。之后，对每一条线段判断出其两个端点在其所在直线上是位于与椭圆相交点同侧还是异侧，若为异侧，则记录下椭圆长轴，若为同侧，则计算过距相交点最近端点的

椭圆，并记录下这个新计算椭圆的长轴。至此，就已经得出与每个线段相交的最小椭圆。将这些椭圆按轴长从小到大排序，删除掉轴长过小椭圆，剩下的轴长最小的椭圆对应线段即为约束边界，再根据该线段的位置、斜率来判断是左边界还是右边界，同侧边界只记录其对应椭圆轴长最小的那个，两侧边界都找到即停止查找。

图 3.21 展示了一些随机选取图片的行驶约束边界查找结果（短轴与长轴比为 0.8）。左边一列是车辆在道路上采集到的原始图片，右边一列为本算法的检测结果。图片中的绿色区域为检测出的道路区域，粉红色线段为识别出的可能行驶约束边界，蓝色线段为筛选算法处理后识别出的真正约束车辆行驶的边界。



图 3.21 道路检测及行驶约束识别结果

3.5 本章小结

本章详细介绍了本项目无人驾驶视觉系统的道路检测部分。道路检测部分核心是应用了为车辆行驶场景特殊设计的改进种子区域增长算法来实现图像分割，同时应用了多色彩空间处理、去空洞处理、时域滤波等手段来增强算法的鲁棒性。针对结构化道路区域上具有约束意义的信息，本系统设计了其识别算法，帮助车辆做出行驶决策。经实验车辆上路测试，本部分系统检测效果良好。

4 远程监管控制系统

视觉作为人类与生俱来、最为熟悉的数据感受途径，不仅仅能够被自主行驶车辆的计算机所利用、指导驾驶，更可以为人们提供一个直观监管途径。本章将介绍无人驾驶视觉系统中的远程监管控制系统，着重说明其依靠的网络传输技术、视频压缩技术、数据包组成设计、网络异常处理，并给出专门为互联网模式设计的服务器软件架构，以及后台端交互设计。

4.1 远程监管控制系统架构

本系统设计的基于视觉的远程监管控制系统有互联网模式和 Wi-Fi 网络模式两种模式，其中基于互联网模式的架构采用：车辆端、服务器端、后台端，三端架构；基于 Wi-Fi 的模式的架构采用：车辆端、后台端双端架构。本远程监管控制系统主要应用到网络传输技术、视频压缩技术，依托本系统定义的数据包格式、服务器多线程软件、后台端软件，完成稳定可靠的对行驶车辆监管控制功能。由于基于互联网的开发版本除了多出中转服务器的设计以外，其余设计与基于 Wi-Fi 的开发版本完全一致。这部分系统的简要架构使用基于互联网版本展示，如图 4.1 所示。

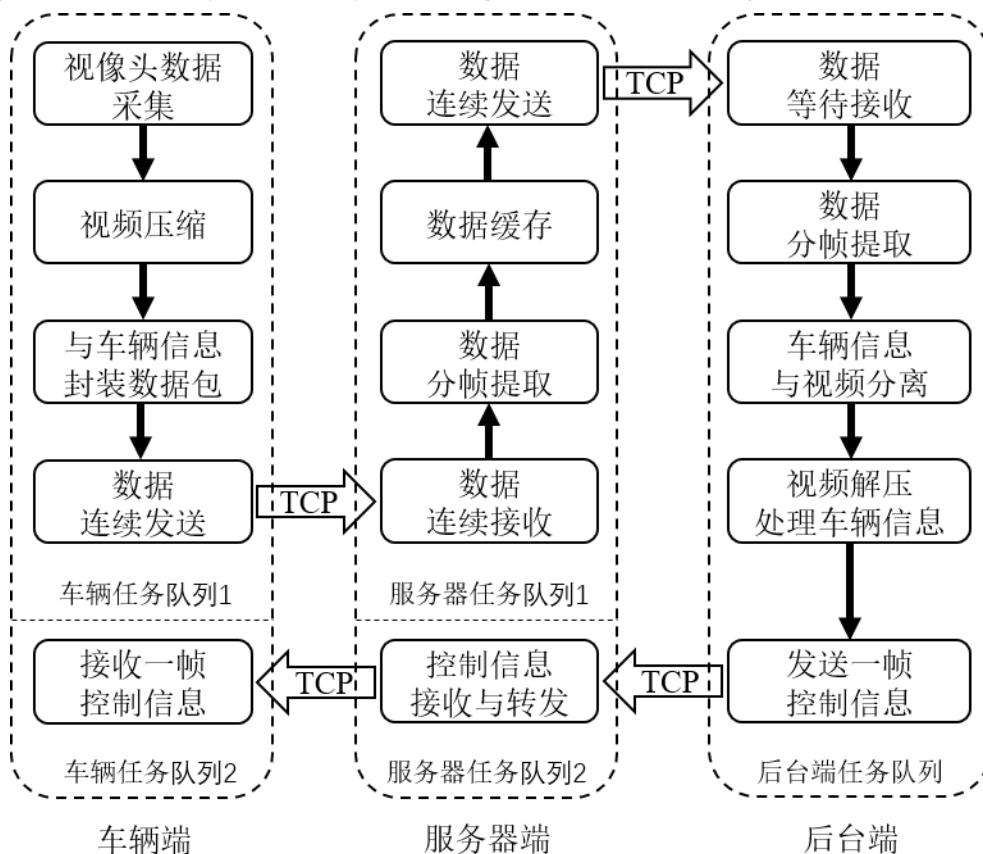


图 4.1 互联网模式下远程监管控制系统架构图

本智能无人配送机器人的视觉系统的车辆端、服务器端、后台端软件设计全部采用多线程设计，图 4.1 中仅简化地展示了与远程监管控制部分相关的流程，关于车辆端的软件详细设计可参见第二章第二小节中的图 2.2，关于服务器端的软件详细设计可参见本章的第五小节。

本章的后续章节将依次详细介绍本系统所利用的网络传输技术、视频压缩技术、多信息数据包封装格式、服务器详细设计、后台端交互设计。

4.2 网络传输技术

TCP/IP（传输控制协议/互联网协议，Transmission Control Protocol/Internet Protocol）协议族是一套应用在互联网和相似计算机网络中的概念模型以及通信协议，是互联网的基础通信架构。其得名自该协议家族最早通过的两个协议，传输控制协议（TCP）以及互联网协议(IP)。TCP/IP 协议族支持了端到端的数据通信，并指明了数据应该如何被包装、分配地址、传输、路由和接收。该协议族将所有网络相关协议按照组网程度划分成了四层，从最下层到最上层分别为：链接层、网络层、传输层、应用层。

链接层被用来在两个连接到一起的主机间交换数据包。根据以太网协议，传输数据的基本单元为数据包，每一个数据包由标头和数据两部分组成：其中标头包含了发送者与接收者信息、数据类型，固定为 18 字节；数据最长为 1500 字节。同时，以太网协议规定联网设备必须具有网卡接口，而网卡的地址就是数据包的接收地址、发送地址，每台设备独一无二，被称为 MAC 地址，长度 48 位。

网络层被用来负责跨域子网络的数据包交换。各区域网络之间的主机通信需要将数据从原地址所在网络发送到目的地址所在网络，这个过程被称为路由。在网络层之后，每个联网设备除了 MAC 地址还有一个网络地址，网络地址由管理员分配，用来帮助确定计算机所在的子网络。IP 协议就是用来规定网络地址的协议，在其之上发送的数据包被称为 IP 数据包，同样分为标头与数据两部分，整体长度在 65535 个字节内。通常整个 IP 数据包被放入以太网数据包的数据部分，若长度超过 1500 个字节，则会被拆开发送。

传输层被用来建立基本数据通道以供应用程序区分不同任务的数据交换。为了计算机各进程能够区分出供其使用的数据包，传输层建立了网络端口的概念，其取值为 0 到 65535 之间的整数。在传输层进行的端到端数据传输可以被分为两类：传输控制协议（TCP）中使用的面向连接的传输和数据报协议（UDP）使用的面向无连接的传输。UDP 提供的服务不需要双方建立连接，同时不需要接收方确认，属于不可靠的传输，可能出现丢包、乱序等现象，其数据包也分为标头和数据，标头含有发送接收端口，整个数据包总长度不超过 65535 个字节。TCP 提供的服务

需要双方三次握手建立连接，是一种可靠的数据量服务，提供流量控制、数据校验、按序传输、丢包重传等服务，其数据包长度理论无上限。

应用层包含了大多数应用向用户提供服务所需的协议。这一类的协议包括超文本传输协议（Hypertext Transfer Protocol, HTTP）、文件传输协议（File Transfer Protocol, FTP）、基础邮件传输协议（Simple Mail Transfer Protocol, SMTP）等。流媒体技术中常用的实时传输（RTP）协议和 RTP 控制（RTCP）协议也在这层。

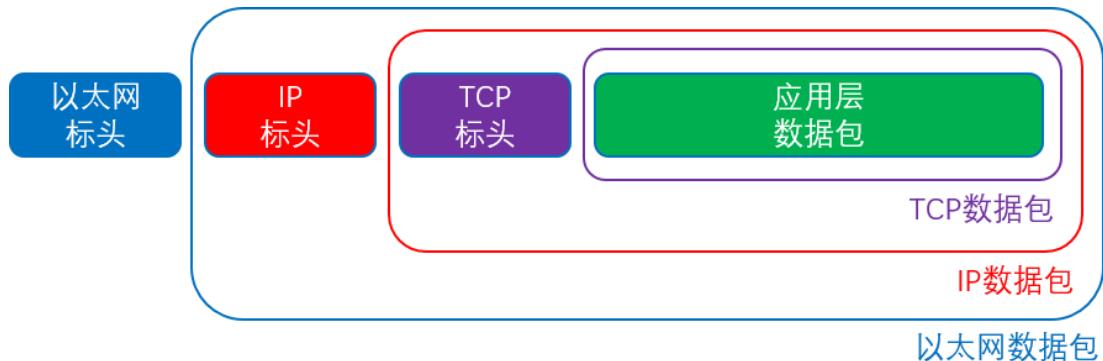


图 4.2 TCP 传输数据结构

本系统为了满足远程控制系统的高度可定制性、平台无关性，在 Windows 系统和 Linux 系统下都可以采用套接字（Socket）编程，将所有的数据传输都直接建立在 TCP\IP 协议族的传输层中 TCP 协议上，数据包封装、推送数据流、网络异常处理等任务都针对本系统特殊开发设计。TCP 协议本身提供的各种数据纠错、按序到达等特性可以保证压缩视频信号的准确传输、控制指令的无误到达；同时依靠底层协议自行设计服务器、直接推送裸流数据也有助于减少延时，这对于无人驾驶车辆的监管来说十分重要。组网方式如第二章中图 2.1 所示。

4.3 视频压缩技术

为了实现对视觉信息的充分利用，借助网络传输行程车辆的第一视角图像十分重要。通常来讲，原始的视频信号中的每一帧都由行、列、色彩三维像素矩阵组成，如果直接统计原始图像，一幅长 640 个像素点、宽 360 个像素点、色彩深度 24 位的图像就需要占据 691.2K 字节，按照 30 帧每秒的速率采集视频，一分钟就会产生 1.24G（字节）的数据，根本无法利用现用网络实时传输。值得庆幸的是，视频信号在时间和空间两个维度都具有极大的冗余性，视频传输前可以利用视频信号的这个特性对其进行压缩，以使借助网络实时传输图像数据成为可能。本系统从提升传输及时性的角度考虑，为两种组网模式分别选用了两种不同的编码格式：在 Wi-Fi 直接组网模式下，本系统采用 JPEG 编码格式，以达到缩短压缩时间、任意丢弃高延时帧的目的；在互联网组网模式下，本系统采用 H.264 编码格式，以达

到节约带宽、降低网络延时时间的目的。

4.3.1 JPEG 压缩技术

JPEG (Joint Photographic Experts Group) 是数字图像有损压缩的常用技术，其压缩程度自由可调、以便满足对图像质量和占用空间的不同要求，第一版标准发布于 1992 年。JPEG 编码方式有多种，其中基本的编码过程由以下几个步骤组成：1) 将一幅图像的色彩表现形式从 RGB 转换到 Y'CbCr (其中 Y' 表示储存信息，Cb 和 Cr 储存色彩信息)；2) 对色彩数据进行下采样，通常使 Cb 和 Cr 通道的空间分辨率降低两倍或三倍，因为人眼视觉对亮度信息比对色彩信息更加敏感；3) 将整张图片按照 8×8 的块进行分割(若边缘不足 8 个像素点，则对边界像素复制补齐)，每一个块内分别对 Y'、Cb、Cr 通道进行离散余弦变换 (DCT) 获得频域信息；4) 对频域内的信号幅值进行量化，对低频信号分量以更高精度量化、对高频信号的使用更低精度量化 (因为人眼对大范围区域的亮度微小变化比对高频亮度变化更敏感)，同时总体量化精度可调以便在质量和储存空间之间取舍；5) 对于所有的 8×8 区块采用无损熵编码，进一步压缩数据量。

OpenCV 开源计算机视觉库内置了 JPEG 编码模块，本系统利用其完成 JPEG 格式图片压缩，以供 Wi-Fi 网络下传输图像使用。JPEG 压缩效果如图 4.3 所示：压缩系数为 100 时，肉眼分辨无差别；压缩系数为 10 时，画质下降清晰可以。



图 4.3 OpenCV 库中 JPEG 编码效果：(a) 为重庆大学校园内采集的分辨率为 640*360 的钟塔原始图片；(b)、(c)、(d) 分别为质量系数 100、50、10 时的压缩效果。

表 4.1 JPEG 压缩后图像大小(单位: 字节)

原始尺寸	系数 100	系数 70	系数 50	系数 30	系数 10
1280*720*3=2764800	711479	151978	108482	76409	36076
640*360*3=691200	204106	44941	32100	22504	10526

表 4.2 JPEG 编码、解码耗时 (单位: 毫秒)

分辨率	1280*720	640*360
编码耗时	34 ms	8 ms
解码耗时	15 ms	4 ms
合计	49 ms	12 ms

对图 4.3(a)进行多质量系数下 JPEG 压缩实验,结果显示在表 4.1、表 4.2 中。实验计算机处理器为英特尔 i7-5500U、操作系统为 Windows 10。在 640*360 像素的原图输入下,采用质量系数 50 的 JPEG 格式压缩可以获得接近 22:1 的压缩比,压缩后单幅图像 32100 字节,采用此编码形式的视频完全可以通过 Wi-Fi 局域网络流畅发送;同时质量系数 50 的 JPEG 压缩图像在人眼视觉感受上与原图差距不大,编码解码共耗时仅 12 毫秒;通过上述数据比较,选用质量 50 的 JPEG 压缩满足本系统使用要求。

4.3.2 H.264 压缩技术

H.264 或 MPEG-4 第十部分是一种基于分块、基于运动补偿的视频编码标准,以其高压缩比、良好的压缩质量被广泛应用在 DVD、视频会议、视频监控等各种视频服务中。其主要依靠将整副图像划分成 16×16 的宏块及更小的子块进而实现:1) 帧间预测编码(运动估计与补偿)去除时域上的冗余;2) 帧内预测编码(对宏块进行 9 种模式匹配)去除空域上的冗余性;3) 离散余弦变换及量化;4) 无损熵编码。H.264 编码具有很高压缩比,其格式下视频按组编码,并主要定义了三种帧类型,他们分别是:1) I 帧(关键帧内编码帧),每组最前面的一帧;2) P 帧(前向预测编码帧);3) B 帧(双向预测编码帧)。其中 I 帧包含一幅图像的所有信息,无需依靠其它帧便能复原图像;P 帧包含着当前帧与前一帧之间的差别信息,需要依靠同组内的前面帧的信息才能复原本帧图像;B 帧包含着与前后帧之间的差别信息,需要依靠其前后帧信息才能复原本帧图像。

FFmpeg 是一款免费开源的多媒体处理库,最早在 Linux 平台开发,现已支持 Windows、Linux、macOS 等操作系统。其音视频编解码库 libavcodec 支持市面上绝大多数音视频编码标准,被众多商业软件公司所采用。本系统利用其中 libavcodec、libavutil、libsSCALE 三个库以及外部 libx264 库完成对摄像头采集来的视频裸流数据的 h.264 格式高效压缩及解压。使用 C++ 对其中函数、数据格式进行封装,实现编码的流程如图 4.4 所示、解码的流程如图 4.5 所示。

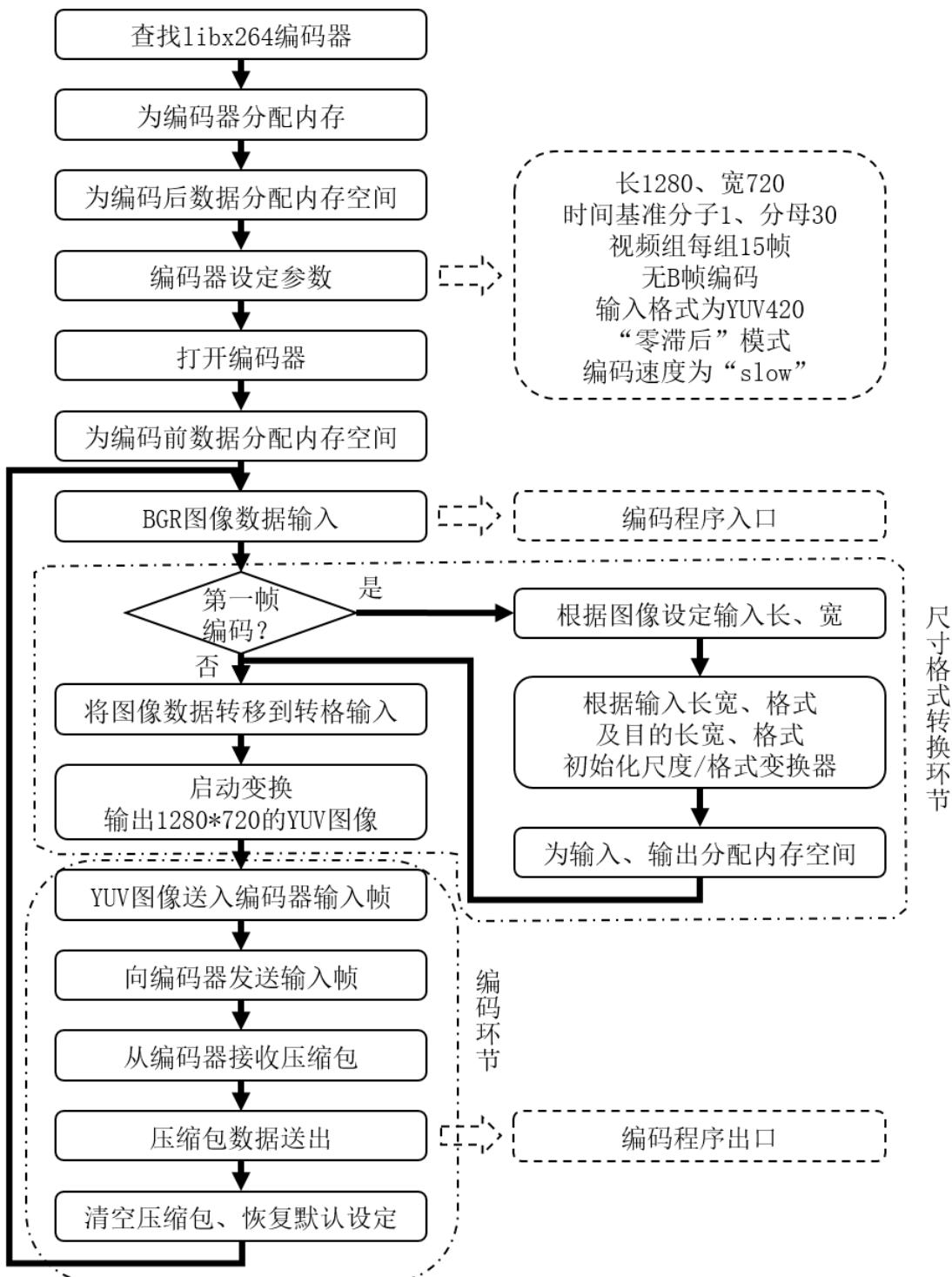


图 4.4 借助 FFmpeg 开源库实现 H264 编码流程

本系统在设定编码器参数时出于低延时考虑、为 H.264 编码选用“zerolatency”模式，每向编码器发送一帧数据、编码器就立即向外届输出这一帧的编码压缩包，而非在内部缓存。同时编码组中取消 B 帧（双向预测编码），保证接收端能在接收到每一帧时及时解码，而不需要等待下一帧到来才能解码。在编码入口处，每次送入的图像都会先被尺寸、格式规范化，保证编码及发送过程不出错。

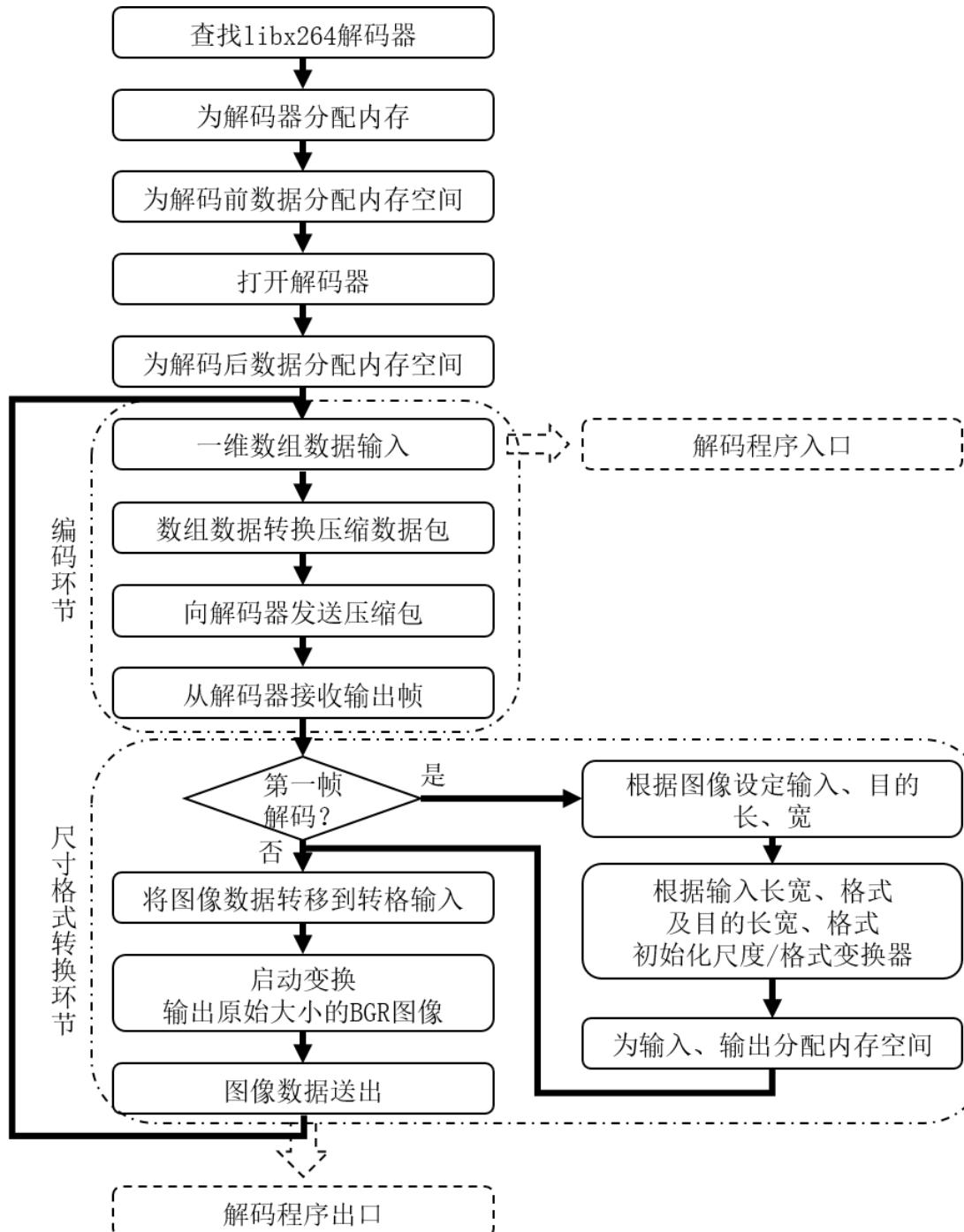


图 4.5 借助 FFmpeg 开源库实现 H264 解码流程

解码环节整体上是编码环节的对称结构，但解码器不需要对参数进行设定，解码器会通过其接收到第一帧图像获取相关信息、完成参数的初始化。本系统接收端送入解码环节的是通过网络接收、处理过的一维数组（即车辆端编码完成后压缩包的裸流数据），首先调用 FFmpeg 中 `av_packet_from_data` 函数对其复原为压缩包格式，再送入解码器进行解码。解码完成后的数据再经过格式变换为 BGR 格式，方便此函数后面的程序对图像进一步处理及显示。

由于 H.264 是基于连续视频信号的编码技术，本文采用了实时采集、发送端接收端同时截图的方式查看其压缩质量。图 4.6 为实验采集图像（由于数据来自实时采集，原图及压缩图并不完全同步），展示了本系统 H.264 技术的压缩质量：其中（a）和（c）图为车辆端摄像头以 1280*720 分辨率采集图像后直接截图；（b）图为车辆端采集图像后原尺寸经由 H264 格式压缩、再通过互联网发送至后台接收端（图中笔记本电脑）、在接收端解码后的图像；（d）图与（b）图仅差别在其是在车辆采集图像后经过尺度变换为 640*360 后再编码发送的图像。从实验效果来看，H264 编码后的图像与原图对比无肉眼观察下的明显差别。



图 4.6 H264 压缩编码图像质量

通过对多段实时采集视频压缩结果分析，本系统 H.264 编码参数设定下（图 4.4 中所示），编码后的图像大小统计结果如表 4.3 所示，耗时如表 4.4 所示。

表 4.3 H.264 压缩后图像大小（单位：字节，15 帧每组）

分辨率	1280*720	640*360
I 帧极大值	95000	25000
I 帧典型值	55000	15000
I 帧极小值	50000	12000
P 帧极大值	18000	2500
P 帧典型值	13500	1400
P 帧极小值	12000	2500
组内每帧平均值	16000	2400

表 4.4 H.264 编码、解码耗时（单位：毫秒）

分辨率	1280*720	640*360
编码耗时	94 ms	63 ms
解码耗时	10 ms	6 ms
合计	104 ms	69 ms

上述结果表明，本系统设定参数下、1280*720 分辨率下 H.264 格式的视频压缩可以获得 172: 1 的压缩比；平均每幅图 16000 字节的数据量完全可以通过互联网流畅传输；尽管由于 H.264 格式编码复杂，软编码的庞大计算量使得计算机需要更长的时间完成编码任务，但是其在码率上的优势大大减少了互联网传输的延时，因此本系统中在互联网模式下的视频数据部分采取 H.264 编码。

4.4 数据包结构

4.4.1 车辆端发送的数据

考虑到无人驾驶技术的应用场景，本系统将车辆发出的数据分为两个部分：视频数据、车辆状态。其中视频数据为通过 JPEG 或 H.264 压缩的视频数据包以及该帧类型（I/P 帧），包含无人驾驶车辆视觉系统采集到的第一视角信息，方便后台人员直接通过图像观察车辆周围环境、或借助图像完成对车辆的直接接管；车辆状态即包括车辆的自动驾驶系统检测出的道路信息、车速、转角等车辆的自身状态，方便后台人员分析车辆的运行情况。

由于压缩视频数据的长度在不断变化，本系统将车辆端发送的数据组合成数据帧（其组成结构如图 4.7 所示）。每一个数据帧之间通过预先定义的帧分隔符划分，车辆的视觉系统每采集完一幅图像（定长度）并对其压缩后（变长度），便和车辆状态（定长度）一起打包成一个数据帧发出。这样，即使每次数据的长度不定、接收方不一定一次全部接收完毕，接收程序在本文设计的分段函数配合下也能保证数据的传输准确无误。另外，尽管 TCP 协议在理论上对数据包长度并无限制，不过为了保证发送设备/接收设备的网卡有足够的发送缓存/接收缓存区，本系统的每一个数据包在发送的时候将长度限制在 10000 字节，超出长度的数据包则按照每次 10000 字节发送、若剩余字节不到 10000 字节，则只发送剩余部分长度（每次发送统计成功发送字节数、剩余发送字节数，防止数据访问越界）。



图 4.7 本系统车辆端发送数据包结构

为配合车辆端发送的数据包格式，本文设计了数据分段接收算法，以保证每次调用此函数都会从发送端接收数据或从缓存读取剩余数据，再准确分割、输出一帧完整数据供接收端后续处理，其实现流程如图 4.8 所示。“接收数据”、“数据缓存”、“输出数据”为三个 unsigned char 型向量。

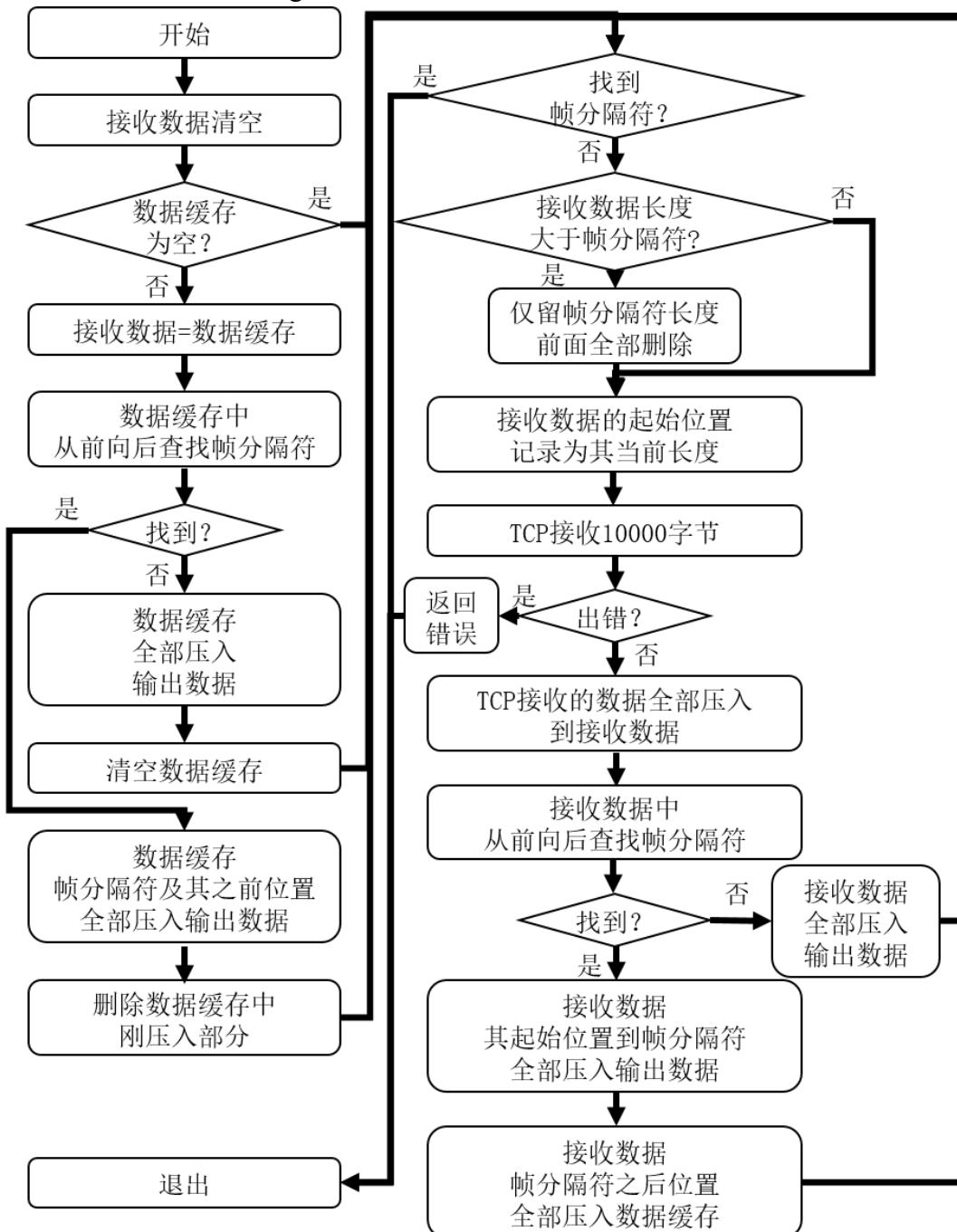


图 4.8 数据分段/分帧接收函数

4.4.2 后台端发出的数据

本自动驾驶视觉系统设计的后台端仅向车辆发送指令（速度、是否自动驾驶、手动模式下转角角度）所以设置为固定 10 字节长度的数据包，无帧间分隔符。

4.5 服务器架构

为了实现借助现有互联网传输数据、完成远程视频监管控制系统，需要在车辆和后台端之间架设一台具有公网静态 IP 地址的服务器，起到数据中转的作用。本系统开发、实验采用阿里云轻量应用服务器，搭载 Ubuntu 16.04 操作系统。

服务器程序使用 C++语言编写，具有自动分辨接入者身份（车辆端还是后台端）、网络状况检测（心跳包检测）、抢占式登入（车辆/后台端若未掉线则拒绝新接入车辆/后台端）、JPEG/H.264 双格式支持、低延时中转数据等功能。

由于服务器需要支持随时接入连接，并且同时服务车辆、后台，对他们分别有着收发任务，本服务器软件采用 C++11 标准库中 `thread` 库实现多线程技术，同时利用 `mutex` 库中的互斥锁功能保证线程间数据的安全交换。

服务器在正常工作情况下会有 6 个线程启动：主线程、等待服务器连接线程、心跳检测线程、从车辆端接收数据线程、从后台端接收数据线程、向后台端发送数据线程。其间关系及各自流程如图 4.9、图 4.10 和图 4.11 所示。

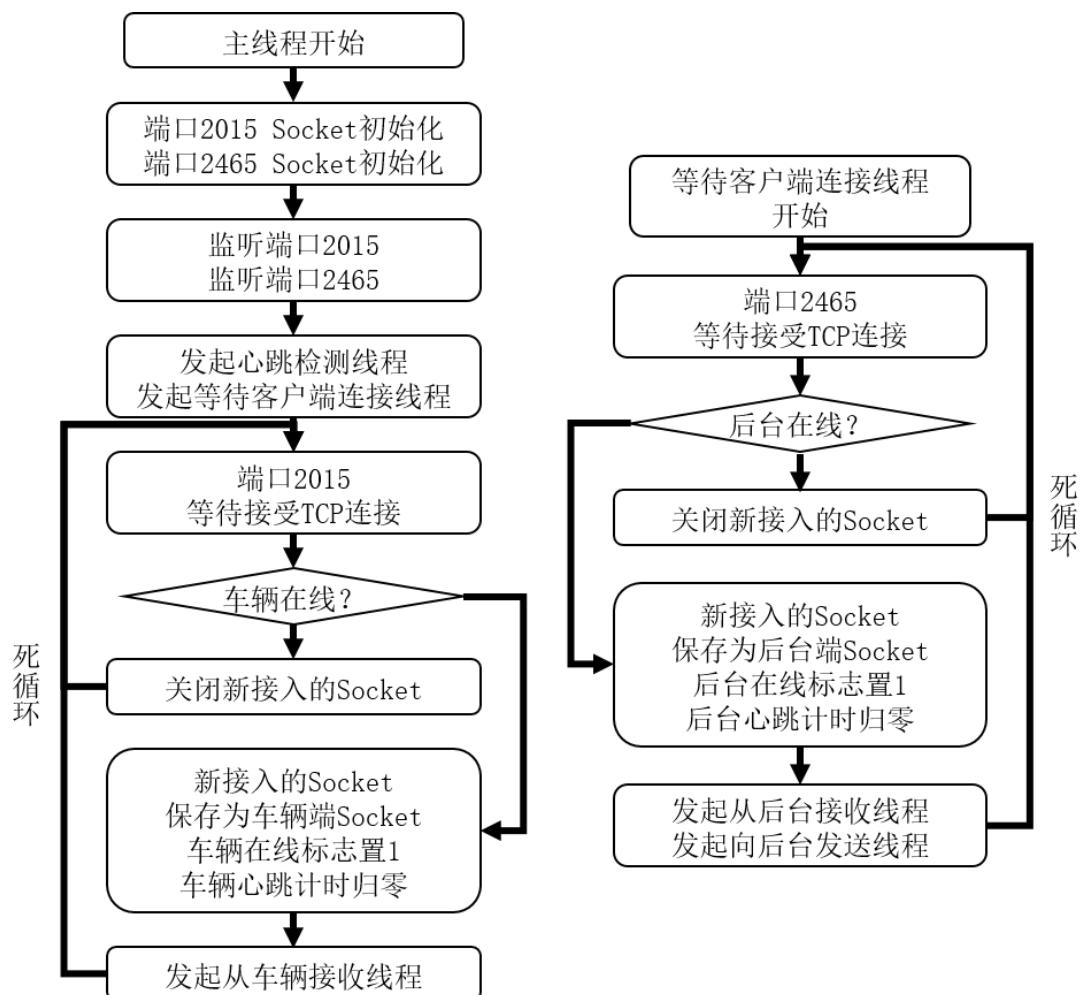


图 4.9 服务器主线程及等待客户端连接线程流程图

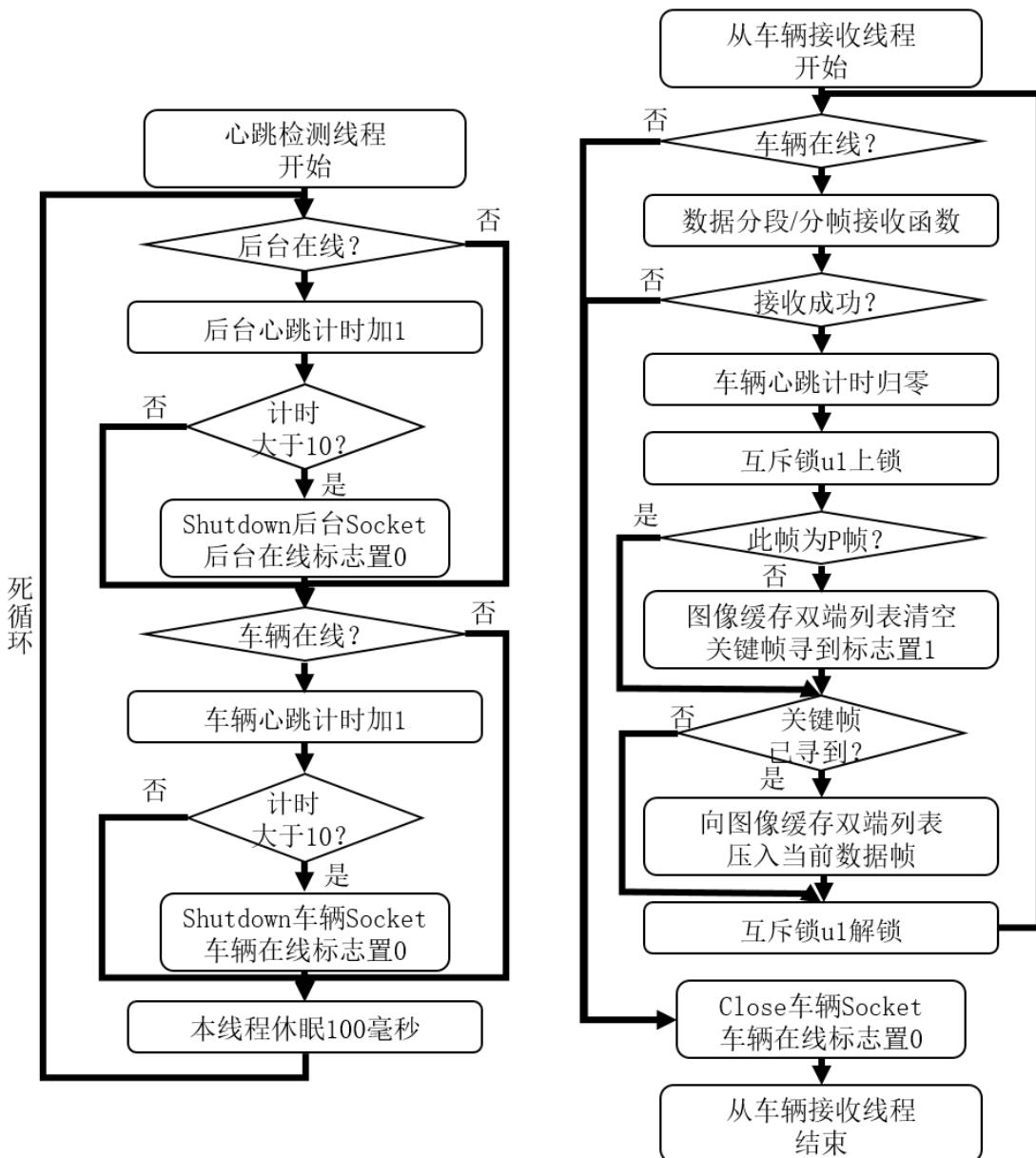


图 4.10 心跳检测线程及从车辆端接收数据线程流程图

由于 TCP 协议一旦三次握手完成、除非双方主动断开连接或服务器方主机重启，则 Linux 下的 TCP 连接状态会一直保持。因此，在网络状况不好、或接入服务器端软件异常时，TCP 连接出错并无法被主动获知。本系统引入心跳包检测机制，去主动探测 TCP 连接或者网络状态是否良好。当超过心跳计时周期，服务器依然未收到来自连接方的数据，则服务器主动断开该 Socket 连接，回收资源。心跳计时在每次收到连接方数据时清空。

服务器每次收到来自车辆端的数据包时，会提取其中的视频帧类型信息。当当前帧是 P 帧（前向预测编码帧）时，会将帧压入为其准备的双端列表中，以防止每组图像中（15 帧一组）丢失中间帧造成 H.264 解码失败。

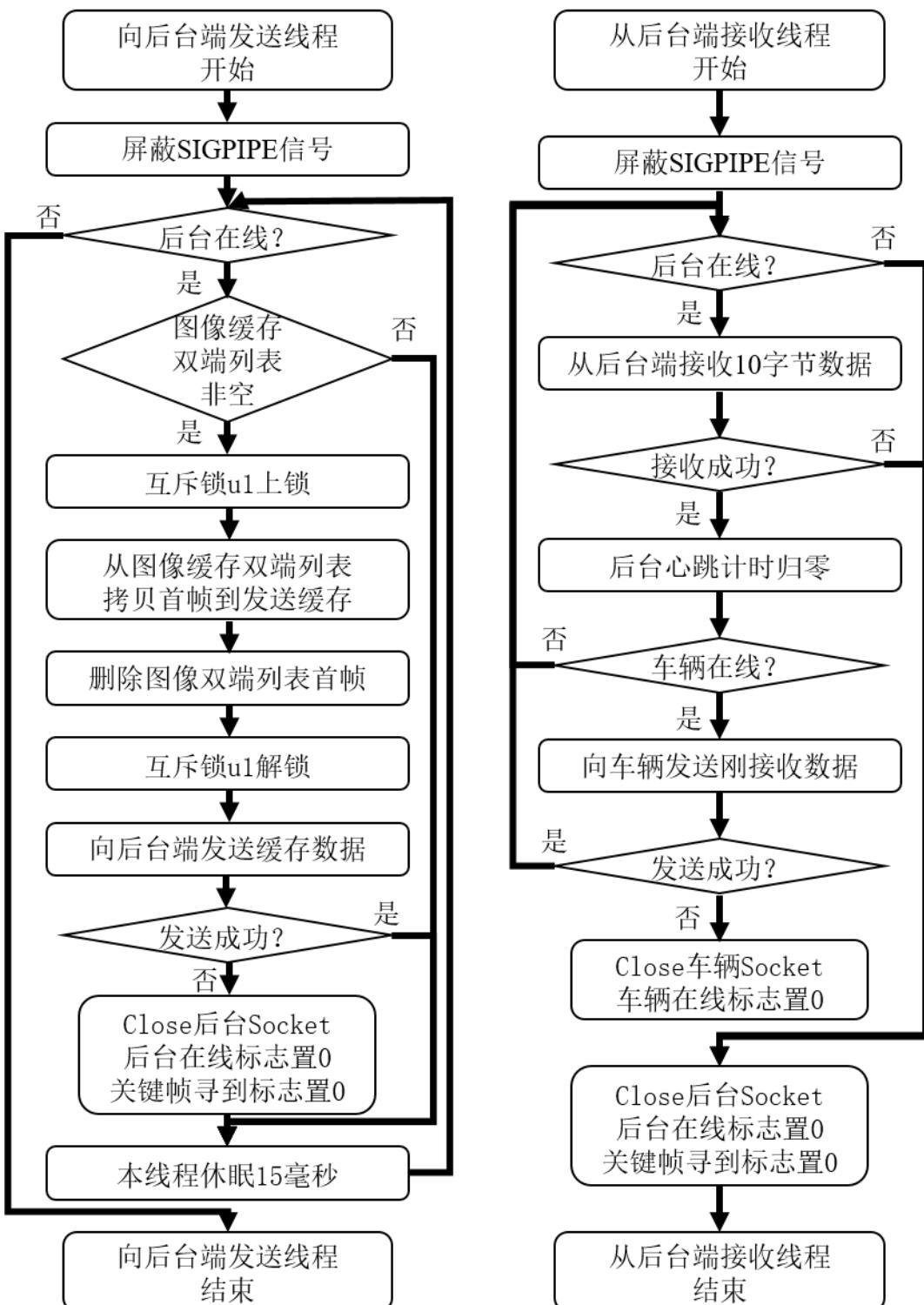


图 4.11 与后台端收、发数据线程流程图

由于心跳检测的存在，网络信号不好时程序会断开 TCP 连接、关闭 Socket，在这之后若是有向该 Socket 的发送任务，系统会抛出 SIGPIPE 异常，所以先对其屏蔽，防止程序退出。向后台端转发数据时，数据来自双端列表缓存，发送时从前弹出数据（装入数据是从后压入），保证同组图像不丢帧、帧续不错乱。

在本服务器的支持下，对互联网模式下的远程监管控制系统的及时性进行测试，实验时图像采集发送端处理器为英特尔 i5-7300U、图像接收显示端处理器为英特尔 i7-5500U，都安装 Windows 10 操作系统；云端服务器为阿里云轻量应用服务器学生版。测试的网络路径为：图像发送端计算机-手机移动热点-4G 基站-网络供应商服务器-自行搭建阿里云服务器-网络供应商服务器-校园网 Wi-Fi 热点-图像接收端计算机。图 4.12 为实验场景图。

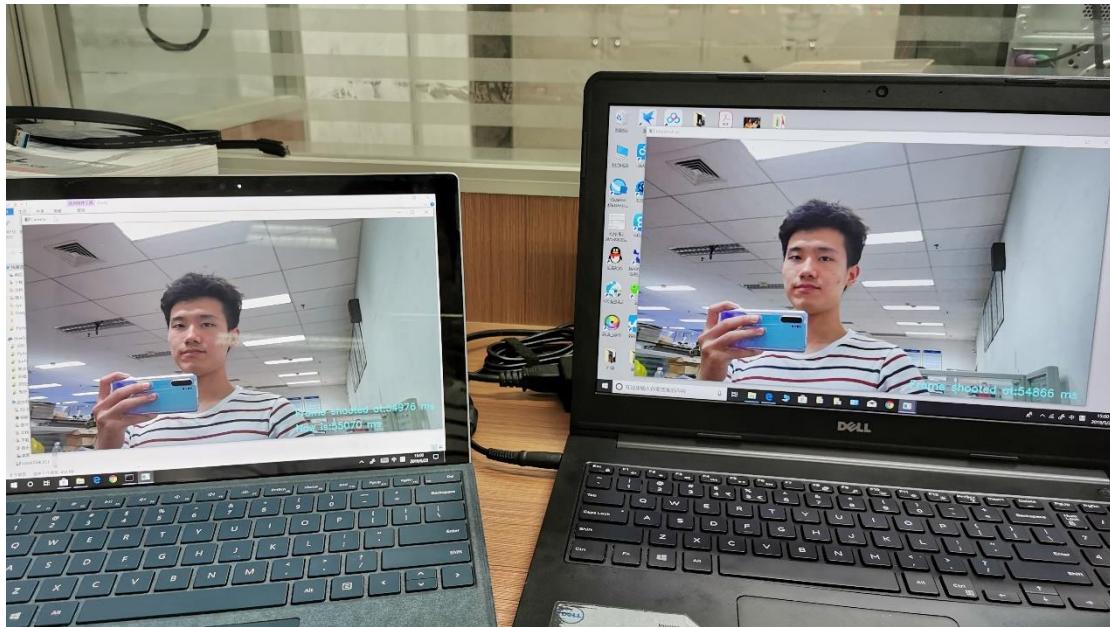


图 4.12 网络延时测试图（左边计算机负责采集图像并编码发送至服务器、右侧计算机负责从服务器接收数据并解码显示；屏幕下加有时间戳、以便测试延时）

实验结果表明，在实验网络环境下，本远程监管控制系统单向链路传输 1280*720 分辨率的图像从采集到远程显示总延时为 204 毫秒，其中 h.264 格式视频编码耗时 94 毫秒、网络传输及解码耗时 110 毫秒。

同时，本服务器需采取远程 SSH 登陆进行开发、调试，因此为其设计了命令行交互界面，以方便管理人员随时监督车辆、后台端的连接情况以及服务器本身工作运行情况。图 4.13 为其一个示例。

```
*****
-----Linux Vision-based Control Cloud Server ( By Li Jiatong )-----
-----视觉控制云端服务器 作者：李佳桐-----

listening... Please link car and controller!
(请连接行驶车辆和操控端！)

New User Links in! Socket:5
Car links successfully! (车辆接入成功！)

ReFromCar Error! (从车辆接收过程出错！)
----Please relink car! Waiting ... (请重新连接车辆！等待中……) ----
```

图 4.13 服务器指示界面

4.6 后台端交互设计

自动驾驶车辆的远程监管控制系统是面向人的一套系统，因此设计一套交互感良好、功能齐全的后台端软件既可提升后台人员管理便捷性、又能使得在路上行驶的车辆更加安全。本系统的后台端软件专门为提升车辆管理体验、充分发挥视觉系统能力而设计。主要具有如下几个功能：

- 1) 第一视角显示图像：依托前面章节开发的网络传输及视频压缩技术，本后台端可以让管理人员以低延时、高分辨率的第一视角图像看到车周环境。
- 2) 显示车辆运行状态：当车辆运行在自动驾驶模式时，画面的左上角会显示出车辆自行判断的并以此控制的转角信息；当车辆运行在手动控制模式时，画面左上角会显示当前对车辆的操控角度和操控速度，键盘可方便更改速度、角度。
- 3) 显示车辆检测结果：本系统定义的车辆发出数据包中含有行驶约束线检测信息，后台端软件可以根据用户意愿从数据包中解析出这部分数据显示在图像上（按 L 键显示、Shift+L 键取消显示）。
- 4) 物体识别：当用户按下 O 键时，程序会新弹出一个窗口，并在窗口中开始实时物体检测（检测速度较慢、单独显示），按下 Shift+O 键即可关闭弹窗。
- 5) 模式切换迅速：当用户按下左右方向键时，车辆控制权立刻归用户。按 Shift+W/S 可改变车速，画面上有上/下/左/右箭头指示，其透明度随车速改变。
- 6) 随时录像：用户只需按下 R 键即可录像（右上角红圈指示），当完成一段录像后、用户只需按 Shift+R 就能停止录像；再按 R 会自动更名保存新文件。
- 7) 断网自动重连：当网络信号出现问题时，后台端软件会自动尝试重连。

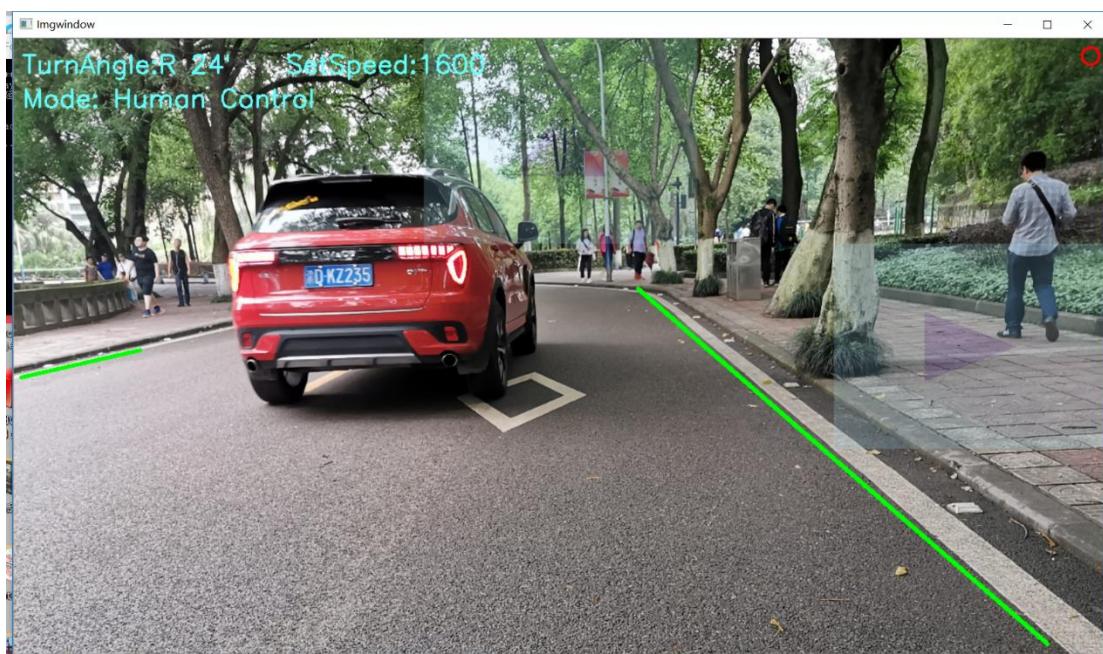


图 4.13 手动收回车辆控制权界面（画面右侧和上侧的半透明方框箭头指示当前按下了键盘前进和右转键，左上角指示当前控制的转角和速度，画面中绿线为车辆自行检测的边线）



图 4.14 车辆自动驾驶界面（画面右上角实心红点表示正在录像；左上角指示车辆状态）

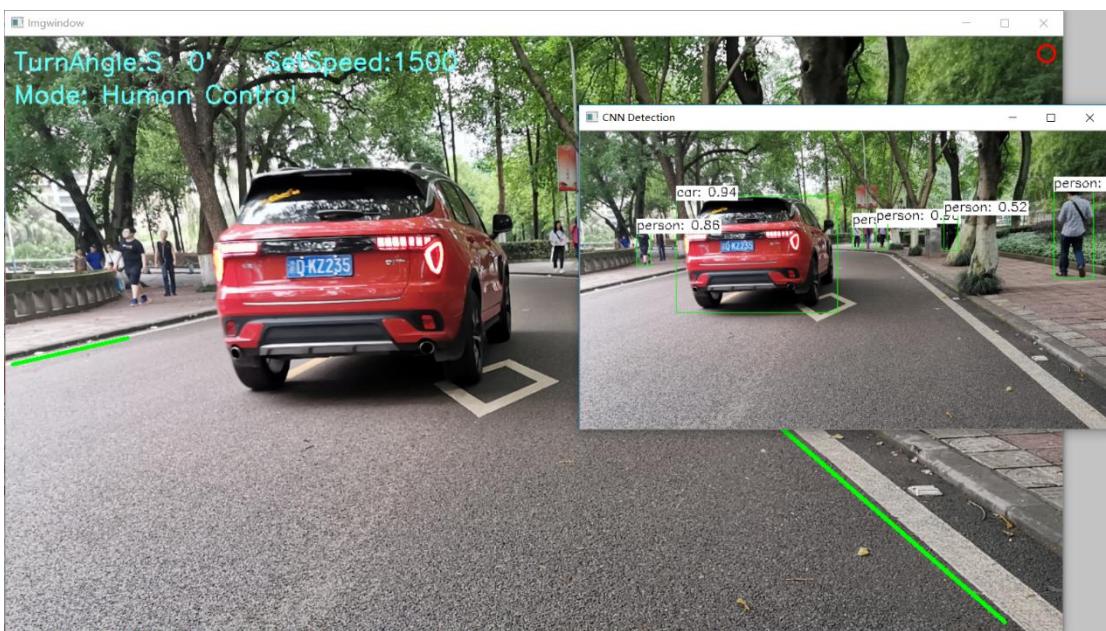


图 4.15 物体识别弹窗（由于耗时较长、刷新率较大窗口低）

4.7 本章小结

本章专门讲解了无人驾驶视觉系统中的远程监管控制部分。认真介绍了从网络传输搭建、视频编解码流程、数据包格式定义、服务器的设计到后台端软件功能实现的全过程。本远程监控控制部分借助现用互联网络及基础网络协议，实现了低延时、高硬件通用性的双向信息传输；互联网连接下 204 毫秒延时的性能，满足了低速自动驾驶车辆的无线图像传输要求。同时，建立在互联网公网连接上的整套系统使得后台监管端与车辆端之间的及时大数据通信不再受距离的限制。

5 行驶实验

为了测试本套智能无人配送机器人视觉系统的性能，本文选取了光线充足下柏油马路、瓷砖路，潮湿的水泥地以及昏暗光线下柏油马路共四种测试场景对基于本视觉系统的机器人自动行驶能力进行测试。实验车辆如第二章图 2.3 所示。

5.1 光线充足路段测试

5.1.1 柏油路测试

首先测试车辆在光照充足环境下的运行情况。柏油马路是一般车辆行驶的主要环境，针对柏油马路的路面行驶测试能够校验多数情况下系统功能是否正常。实验的测试环境如图 5.1 中 a 所示，b 展示了在远程后台端接收到的车辆信息，c 为后台端弹窗显示的物体识别的结果。

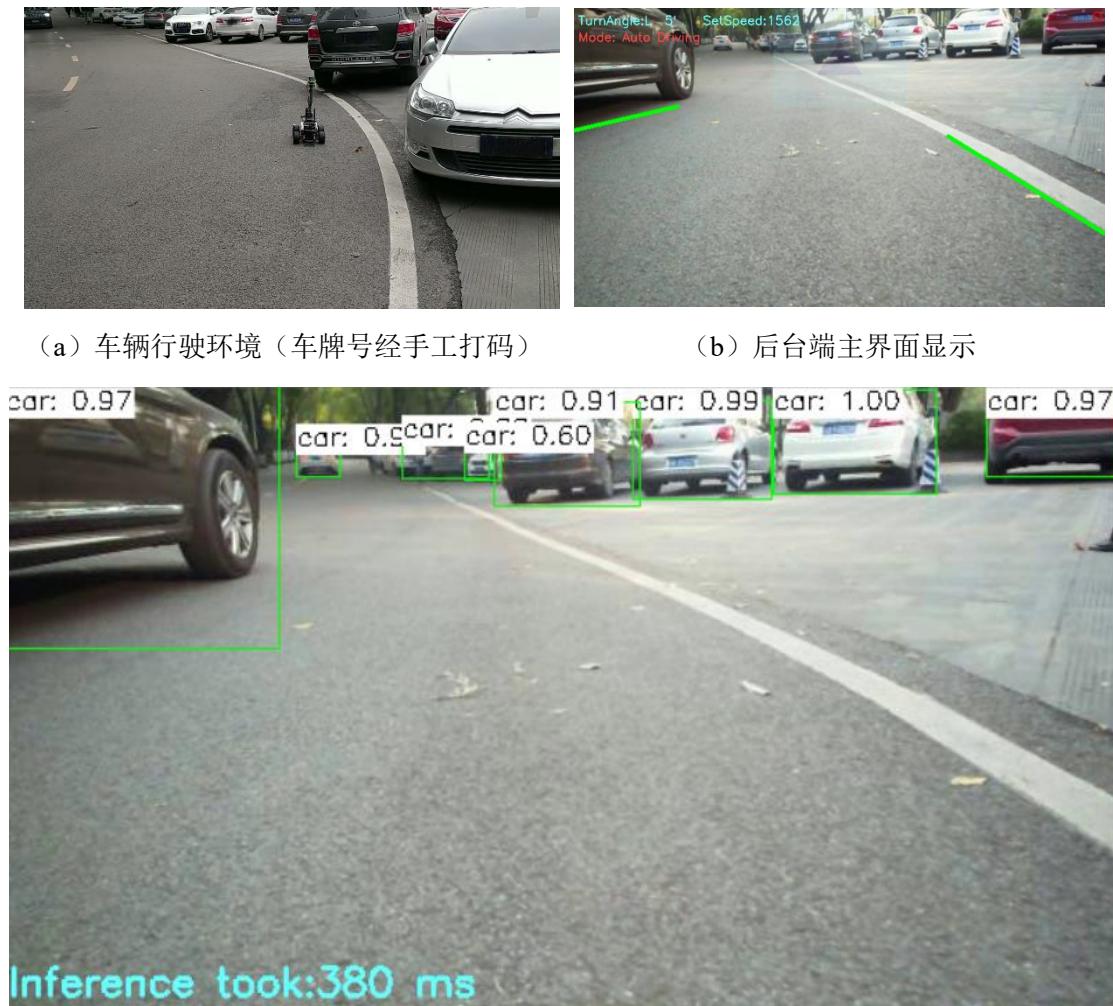


图 5.1 光线充足的柏油马路测试情况

实验表明，在柏油马路上，路面往往标有较为明显的结构性指示信息，方便车辆识别与利用。车辆能够在柏油马路上稳定运行。

5.1.2 瓷砖路测试

对于智能无人配送机器人来说，有时会在大型建筑物内部行驶。而室内的光照环境通常来讲比较稳定、结构化信息（地边沿）较为明显，不过公共场合室内地面通常铺放地砖，其反光会对视觉识别带来极大干扰。在室内瓷砖地上的测试结果如图 5.2 所示，其中 a 展示了测试时的室内环境，b 图展示了与 a 图同一时刻后台端的界面显示（即 a 中的电脑屏幕），c 图为手机拍摄的车辆周围环境与后台端实时界面（带有物体识别弹窗，物体识别弹窗较主页面滞后，因为其耗时更长）。注意 c 图拍摄时如后台端主界面显示，为手动夺取控制权模式。



图 5.2 光线充足的瓷砖路面测试情况

实验表明，在地面反光的瓷砖地面上，车辆行驶不稳定，偶尔会出现数帧乱摆方向、车身抖动、晃动的情况。且经过对车辆状态监控发现，车辆自动控制不稳定的原因正是基于视觉的行驶约束边界检测不准。

5.2 潮湿水泥路测试

在车辆室外行驶的情境下，可能会遇到下雨导致的地面潮湿情况，检验车辆在此环境下的运行情况能够一定程度上检验视觉系统的抗天气干扰能力。图 5.3 中的 a 展示了实验时车辆周围环境，此时刚刚下过雨、地面明显可以看出较为潮湿，部分区域反光， b 展示了后台端显示的主界面， c 展示物体识别结果。



图 5.3 潮湿的水泥路面测试情况

实验结果表明，在雨后的潮湿水泥地面（且道路边界仅有路沿与道路的自然过渡，没有人为标注的结构化指示信息），车辆的视觉系统依然能够正常、稳定工作，车辆行驶过程稳定。

5.3 光线昏暗路段测试

光线昏暗的情况下，视觉系统往往会收到较大干扰，甚至完全失灵。本系统在太阳彻底落山后，摄像头无法采集到足够亮度、足够清晰的画面，因此测试时采用

太阳即将落山的黄昏时段。图 5.4 展示了光线不足情况下的车辆实验结果，其中 a 展示了后台端的主界面显示情况，此即为接收到的车辆发送的第一视角图像，可以看出，此时整体环境较为昏暗；b 展示了后台端物体识别弹窗界面。



(a) 后台端主页面显示



(b) 后台端物体识别弹窗

图 5.4 潮湿的水泥路面测试情况

实验结果表明，尽管在光线不充足的环境下，本视觉系统设计的道路检测技术和行驶约束边界识别技术依然能够正常稳定工作，准确识别出行驶约束线。车辆在测试时的行驶较为稳定。

5.4 系统耗时

道路检测及约束边界识别耗用时间约 20 毫秒。

互联网模式下，640*360 图像分辨率的情况下，从图像采集到后台端显示实时画面延时约 200 毫秒（其中 h.264 格式编码约占 60 毫秒，具体统计见表 4.4）。

Wi-Fi 连接模式下，640*360 图像分辨率的情况下，从图像采集到后台端显示实时画面延时约 50 毫秒（其中 jpeg 格式编码约占 8 毫秒，具体统计见表 4.2）。

远程监管控制系统显示画面偶尔卡顿，多数情况下流畅。

物体识别算法处理一帧图像耗时在 300 毫秒到 500 毫秒（后台端应用 Intel i5-7300U 处理器，算法调用核心显卡 Intel HD Graphic 620）。

6 总结与展望

本文着重研究了自动驾驶技术中的视觉系统，从计算机视觉、流媒体传输两个角度入手，分别完成了对无人驾驶车辆的行驶能力开发和监管途径构建。

在计算机视觉方面，本文主要研究了道路检测技术。通过分析大量路面图像空域特征以及其在不同色彩空间下的特点，本文提出了一种改进种子区域增长算法，以该算法为中心，并辅以一系列数字图像处理手段，设计出了具有较高及时性和鲁棒性的道路检测技术。同时，在稳定的道路检测技术基础上，本文还设计了行驶约束边界的识别及筛选算法，给道路检测技术一个应用示范。通过在重庆大学校园内实验，本套道路检测技术及行驶边界检测技术可有效指导车辆行驶，在较小的计算量下时间车辆基于视觉形式。另外，本文还在后台端应用了 Yolo v3 神经网络模型实现了对路面物体的识别与定位，辅助车辆的驾驶。

在流媒体传输角度，本文独立开发了包含车辆端、服务器端、后台端在内的一整套无人车辆远程监控控制系统。通过利用开源库实现视频的软编码以及利用基础 TCP 协议传输数据消除系统对专有视频芯片、专业图传设备的依赖，并依靠推送自定义数据包、自搭建服务器的方式降低了系统传输数据的延时，同时精心设计网络状态检测机制及各端断网停车重连功能保障车辆行驶安全。

不过，随着硬件水平的不断提高，卷积神经网络的庞大计算量未来必将不是制约它的因素，依靠卷积神经网络的、更精准的语义分割技术在图像分割领域还有较大发展空间。通过利用深度学习算法来增强对路面的检测效果十分可行。

在远程监管控制系统上，本文只设计了基于 Windows 和 Linux 的客户端。为迎合技术发展趋势，将整套系统打通移动端的阻隔也十分具有应用意义。同时，H.264 等先进视频编码技术需要较大计算量，若采用视频编码芯片来替代对图像数据的软编码会大大降低整个系统的延时。

致 谢

经过一个多学期的努力，本人完成了毕业设计的任务。从最初梦想做出一辆拥有自动驾驶能力的车辆，到一步一步完成关键技术，最终完成了基础功能搭建，这对我来说是一个完成梦想的过程。

感谢我的指导教师李敏老师，正是在李老师的大力鼓励下、我才有信心选此题目、做此题目。同时，李敏老师在整个毕设期间的悉心指导和大力支持是使得我不断学习、不断进步的源动力。

感谢大学四年来教授过我的所有教师，正是这四年的学识积累才使得我能够有足够夯实的基础去接触、去挑战新的前沿技术，才让我有能力选择未来的方向。

感谢光电工程学院智能汽车创新实验室的同学们，我们一起奋斗、一起欢乐的日子才是我们科研道路的基石，我们一同成长。

感谢审阅此论文的各位老师，您的审阅才使得我的毕设付出有了关注，感谢您的辛勤付出！

参 考 文 献

- [1] 殷凡青,李传友,姜良超,程吉鹏.自动驾驶汽车的发展现状和展望[J].摩托车技术,2018(11):33-36.
- [2] 唐智威.基于视觉的无人驾驶汽车研究综述[J].制造业自动化,2016,38(08):134-136+140.
- [3] Wang, Y., Chao, W.-L., Garg, D., Hariharan, B., Campbell, M., Weinberger, K.~Q.\ 2018.\ Pseudo-LiDAR from Visual Depth Estimation: Bridging the Gap in 3D Object Detection for Autonomous Driving.\ arXiv e-prints arXiv:1812.07179.
- [4] 黄勇飞. 基于视觉的道路检测技术研究[D].华南理工大学,2015.
- [5] 罗安宁. 基于机器视觉的道路检测算法[D].清华大学,2014.
- [6] T. Fukukawa, Y. Maeda, K. Sekiyama and T. Fukuda, "Road Detection Method Corresponded to Multi Road Types with Flood Fill and Vehicle Control," 2013 Second International Conference on Robot, Vision and Signal Processing, Kitakyushu, 2013, pp. 274-277.
- [7] Su, Yingna & Zhang, Yigong & Alvarez, Jose M. & Kong, Hui. (2017). An illumination-invariant nonparametric model for urban road detection using monocular camera and single-line lidar. 68-73. 10.1109/ROBIO.2017.8324396.
- [8] Peng, C., Zhang, X., Yu, G., Luo, G., Sun, J.\ 2017.\ Large Kernel Matters -- Improve Semantic Segmentation by Global Convolutional Network.\ arXiv e-prints arXiv:1703.02719.
- [9] Yu, C., Wang, J., Peng, C., Gao, C., Yu, G., Sang, N.\ 2018.\ BiSeNet: Bilateral Segmentation Network for Real-time Semantic Segmentation.\ arXiv e-prints arXiv:1808.00897.
- [10] Yu, C., Wang, J., Peng, C., Gao, C., Yu, G., Sang, N.\ 2018.\ Learning a Discriminative Feature Network for Semantic Segmentation.\ arXiv e-prints arXiv:1804.09337.
- [11] 曾友林,冯志华,陈裕鸿,陈挺武,邓国锋,李智财.无人机音视频数据 4G 网络传输装置的研究与应用[J].科技经济市场,2018(12):6-7.
- [12] 温凯林. 低延迟高清数字图传的实现[D].西安电子科技大学,2018.
- [13] 何华丽. 基于 H264 的多平台视频监控系统的研究与实现[D].北京邮电大学,2009.
- [14] 冷莉洪. 基于 H264 的视频传输系统的设计与实现[D].电子科技大学,2017.
- [15] 彭湛博. 无人机实时高清图传系统的设计与实现[D].西安电子科技大学,2018.
- [16] 张新钰,高洪波,赵建辉,周沫.基于深度学习的自动驾驶技术综述[J].清华大学学报(自然科学版),2018,58(04):438-444.
- [17] Dalal, N. and Triggs, B. (2005) Histograms of Oriented Gradients for Human Detection. IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1, 886-893.

- [18] Krizhevsky A., Sutskever I. and Hinton, G E. (2012) ImageNet Classification with Deep Convolutional Neural Networks. NIPS. Lake Tahoe, Nevada.
- [19] 寇大磊,权冀川,张仲伟.基于深度学习的目标检测框架进展研究[J/OL].计算机工程与应用:1-12[2019-05-25].
- [20] Redmon, J., Farhadi, A.\ 2018.\ YOLOv3: An Incremental Improvement.\ arXiv e-prints arXiv:1804.02767.
- [21] 文浩彬,张国辉.基于 YOLO 的驾驶视频目标检测方法[J].汽车科技,2019(01):73-76+72.
- [22] 王文豪,高利.一种基于 OpenCV 的车道线检测方法[J].激光杂志,2019,40(01):44-47.
- [23] Michelle Valente, Bogdan Stanciulescu, Real-time Method for General Road Segmentation DOI.10.1109/IVS.2017.7995758
- [24] Dong Li,YingNan Zhang,XinChao Li,Ke Niu,XiaoYuan Yang,YuJuan Sun. Two-dimensional histogram modification based reversible data hiding using motion vector for H.264[J]. Multimedia Tools and Applications,2019,78(7).
- [25] Yiming Xue,Jie Zhou,Hao Zeng,Ping Zhong,Juan Wen. An adaptive steganographic scheme for H.264/AVC video with distortion optimization[J]. Signal Processing: Image Communication,2019,76.

附录 A：在读期间科研经历与奖项

- [1] 2016 年 6 月到 2017 年 6 月，参加重庆大学大学生科研训练计划（SRTP），制作基于 GPS 定位的太阳能防盗跟踪系统，结题成绩良好。
- [2] 参加第六届全国大学生光电设计竞赛西南赛区赛，获得一等奖。
- [3] 参加第六届全国大学生光电设计竞赛全国赛，获得三等奖。
- [4] 参加第十三届全国大学生“恩智浦”杯智能汽车竞赛西部赛区赛，获得一等奖。
- [5] 参加第十三届全国大学生“恩智浦”杯智能汽车竞赛全国总决赛，获得一等奖。

附录 B：部分程序源代码

道路边界椭圆筛选算法

```

void linesort(Mat& Outputimage, vector<Vec4i>& lines)
{
    vector<long int> lengthtimeilength;
    int lineNum_L, lineNum_R;
    int havefoundL_flag = 0, havefoundR_flag = 0;
    double ellipse_a_L = 100, ellipse_a_R = 100;
    vector<float> slope;
    float slope_limit_perameter;
    vector<int> point1_L, point2_L;
    /*-----坐标系平移-----*/
    for (size_t i = 0; i < lines.size(); i++)
    {
        lines[i][0] -= Outputimage.cols / 2;
        lines[i][1] -= Outputimage.rows;
        lines[i][2] -= Outputimage.cols / 2;
        lines[i][3] -= Outputimage.rows;
    }
    /*-----检测与椭圆相交时最小椭圆-----*/
    float ellipse_b_to_a = 0.7;
    double ellipse_a; //ellipse_b_to_a 小于 1 时是扁的椭圆
    vector<double> ellipse_a_s;
    double AtimesA, BtimesB, CtimesC;
    float crosspoint_x;
    vector<float> crosspoint_x_s;
    for (size_t i = 0; i < lines.size(); i++)
    {
        lengthtimeilength.push_back((lines[i][3] - lines[i][1]) *
            (lines[i][3] - lines[i][1]) + (lines[i][2] - lines[i][0]) * (lines[i][2] - lines[i][0]));
        if ((lines[i][2] - lines[i][0]) != 0)
            slope.push_back(((float)(lines[i][3] - lines[i][1]) /

```

```

(float)(lines[i][2] - lines[i][0]));
else
    slope.push_back(99999);
point1_L.push_back(((lines[i][0] - Outputimage.cols / 2) > 0) ? 0 : 1);
point2_L.push_back(((lines[i][2] - Outputimage.cols / 2) > 0) ? 0 : 1);
AtimesA = lines[i][1] - lines[i][3];
BtimesB = lines[i][2] - lines[i][0];
CtimesC = lines[i][0] * lines[i][3] - lines[i][2] * lines[i][1];
if (AtimesA != 0)
{
    BtimesB = BtimesB / AtimesA;
    BtimesB *= BtimesB;
    CtimesC = CtimesC / AtimesA;
    CtimesC *= CtimesC;
    AtimesA = 1;
    ellipse_a = (CtimesC / AtimesA);
    ellipse_a -= (BtimesB*CtimesC*ellipse_b_to_a /
        (ellipse_b_to_a*BtimesB*AtimesA + AtimesA*AtimesA));
    crosspoint_x = (float)(-1 * (lines[i][0] * lines[i][3] - lines[i][2] * lines[i][1])
        /(lines[i][1] - lines[i][3]) / (AtimesA + ellipse_b_to_a*BtimesB));
    //注意计算时应用的参数都是归一化后的
}
else //A=0 时
{
    CtimesC = CtimesC / BtimesB;
    CtimesC *= CtimesC;
    BtimesB = 1;
    ellipse_a = (CtimesC / (ellipse_b_to_a*BtimesB));
    ellipse_a -= (AtimesA*CtimesC / (ellipse_b_to_a*AtimesA*BtimesB +
        ellipse_b_to_a*ellipse_b_to_a*BtimesB*BtimesB));
    crosspoint_x = 0;
}
if ((lines[i][0] >= crosspoint_x && lines[i][2] <= crosspoint_x) ||
    (lines[i][0] <= crosspoint_x && lines[i][2] >= crosspoint_x))

```

```

ellipse_a_s.push_back(ellipse_a);
else
{
    if (abs(lines[i][0] - crosspoint_x) < abs(lines[i][2] - crosspoint_x))
    {
        ellipse_a_s.push_back((lines[i][0] * lines[i][0] +
                               lines[i][1] * lines[i][1] / ellipse_b_to_a));
        crosspoint_x = (float)(lines[i][0]);
    }
    else
    {
        ellipse_a_s.push_back((lines[i][2] * lines[i][2] +
                               lines[i][3] * lines[i][3] / ellipse_b_to_a));
        crosspoint_x = (float)(lines[i][2]);
    }
}
crosspoint_x_s.push_back(crosspoint_x);
}

/*-----找出左右边界-----*/
//先将所有数组从新排序,
double tempnum1;
Vec4i tempnum2;
float tempnum3;
int tempnum4;
long int tempnum5;
for (size_t i = 0; i < lines.size(); i++)
{
    size_t testposition = i;
    double temp;
    temp = ellipse_a_s[i];
    for (size_t t = i; t < lines.size(); t++)
    {
        if (ellipse_a_s[t] < temp)
        {

```

```

        testposition = t;
        temp = ellipse_a_s[t];
    }
}

tempnum1 = ellipse_a_s[testposition];
ellipse_a_s[testposition] = ellipse_a_s[i];
ellipse_a_s[i] = tempnum1;
tempnum2 = lines[testposition];
lines[testposition] = lines[i];
lines[i] = tempnum2;
tempnum3 = slope[testposition];
slope[testposition] = slope[i];
slope[i] = tempnum3;
tempnum4 = point1_L[testposition];
point1_L[testposition] = point1_L[i];
point1_L[i] = tempnum4;
tempnum4 = point2_L[testposition];
point2_L[testposition] = point2_L[i];
point2_L[i] = tempnum4;
tempnum5 = lengthtimeilength[testposition];
lengthtimeilength[testposition] = lengthtimeilength[i];
lengthtimeilength[i] = tempnum5;
tempnum3 = crosspoint_x_s[testposition];
crosspoint_x_s[testposition] = crosspoint_x_s[i];
crosspoint_x_s[i] = tempnum3;
}

//坐标系反变换
for (size_t i = 0; i < lines.size(); i++)
{
    lines[i][0] += Outputimage.cols / 2;
    lines[i][1] += Outputimage.rows;
    lines[i][2] += Outputimage.cols / 2;
    lines[i][3] += Outputimage.rows;
    crosspoint_x_s[i] += Outputimage.cols / 2;
}

```

```

}

//开始顺序筛选

for (size_t i = 0; i < lines.size(); i++)
{
    slope_limit_parameter = (float)(10000 / ellipse_a_s[i]);
    if (!havefoundL_flag && ellipse_a_s[i] > 100 * 100 &&
        slope[i] < -slope_limit_parameter + 0.05 &&
        crosspoint_x_s[i] < Outputimage.cols / 2)
    {
        havefoundL_flag = 1;
        lineNum_L = i;
        ellipse_a_L = ellipse_a_s[i];
    }

    if (!havefoundR_flag && ellipse_a_s[i] > 100 * 100
        && slope[i] > slope_limit_parameter - 0.05 && slope[i] != 99999
        && crosspoint_x_s[i] > Outputimage.cols / 2)
    {
        havefoundR_flag = 1;
        lineNum_R = i;
        ellipse_a_R = ellipse_a_s[i];
    }
}

/*-----绘制图像-----*/
if (havefoundL_flag)
{
    line(Outputimage, Point(lines[lineNum_L][0], lines[lineNum_L][1]),
        Point(lines[lineNum_L][2], lines[lineNum_L][3]), Scalar(220, 80, 115),
        3, CV_AA);
    ellipse(Outputimage, Point(Outputimage.cols / 2, Outputimage.rows),
        Size((int)(sqrt(ellipse_a_L)), (int)(sqrt(ellipse_a_L * ellipse_b_to_a))),
        0.0, 181, 359, Scalar(0, 0, 255), 1, 8, 0);
}

if (havefoundR_flag)

```

```

{
    line(Outputimage, Point(lines[lineNum_R][0], lines[lineNum_R][1]),
        Point(lines[lineNum_R][2], lines[lineNum_R][3]), Scalar(220, 80, 115),
        3, CV_AA);
    ellipse(Outputimage, Point(Outputimage.cols / 2, Outputimage.rows),
        Size((int)(sqrt(ellipse_a_R)), (int)(sqrt(ellipse_a_R*ellipse_b_to_a))),
        0.0, 181, 359, Scalar(255, 0, 0), 1, 8, 0);
}
}

```

数据包分帧接收函数

```

vector<uchar> restbuff;
int ReFromCar(int sockServer, vector<uchar>* jpgbuff)
{
    int num = 0;
    vector<uchar> samllbuff;
    size_t smallbuffbeginpos;
    int endlastpos = -1;
    char unitbuff[framelen];

    if (!restbuff.empty())
    {
        samllbuff = restbuff;
        endlastpos = tellend(restbuff, Frameend);
        if (endlastpos < 0)
        {
            for (vector<uchar>::iterator it = restbuff.begin(); it < restbuff.end(); it++)
                jpgbuff->push_back(*it);
            restbuff.clear();
        }
        else
        {
            for (vector<uchar>::iterator it = restbuff.begin();
                it < restbuff.begin() + endlastpos + 1; it++)

```

```

        jpgbuff->push_back(*it);

        restbuff.erase(restbuff.begin(), restbuff.begin() + endlastpos + 1);

        return 0;
    }

}

while (endlastpos < 0)

{
    if (samllbuff.size() > endlen)

        samllbuff.erase(samllbuff.begin(), samllbuff.end() - endlen);

    smallbuffbeginpos = samllbuff.size();

    num = recv(sockServer, (char *) (unitbuff), framelen, 0);

    if (num < 1)

        return -1;

    for (int i = 0; i < num; i++)

        samllbuff.push_back((uchar)(unitbuff[i]));

    endlastpos = tellend(samllbuff, Frameend);

    if (endlastpos < 0)

    {

        for (vector<uchar>::iterator it = samllbuff.begin() + smallbuffbeginpos;
             it < samllbuff.end(); it++)

            jpgbuff->push_back(*it);

    }

    else

    {

        for (vector<uchar>::iterator it = samllbuff.begin() + smallbuffbeginpos;
             it < samllbuff.begin() + endlastpos + 1; it++)

            jpgbuff->push_back(*it);

        for (vector<uchar>::iterator it = samllbuff.begin() + endlastpos + 1;
             it < samllbuff.end(); it++)

            restbuff.push_back(*it);

    }

}

return 0;
}

```

