分类号：U495　　单位代码：10335

密　级：无　　学　　号：21532035

浙江大学

硕士学位论文

中文论文题目：用于嵌入式车载安全预警的交通标志检测若干关键技术研究与验证

英文论文题目：**Research and Verification on Some Key Technologies in Traffic Sign Detection used in Embedded Vehicle-Mounted Early Warning**

申请人姓名：高明飞

指导教师：黄文君　教授

专业名称：控制科学与工程

研究方向：模式识别与智能系统

所在学院：控制科学与工程学院

论文提交日期　二零一八年三月

用于嵌入式车载安全预警的交通标志检测若干关键技术研究与验证

论文作者签名: 

指导教师签名: _____

论文评阅人 1：匿名

评阅人 2：匿名

评阅人 3：匿名

评阅人 4：匿名

评阅人 5：

答辩委员会主席：项基\教授\浙江大学电气工程学院

委员 1：许超\副教授\浙江大学控制科学与工程学院

委员 2：金伟\高级工程师\浙江大学控制科学与工程学院

委员 3：

委员 4：

委员 5：

答辩日期：2018.03.09

Research and Verification on Some Key Technologies in Traffic Sign Detection used in Embedded Vehicle-Mounted Early Warning

Author's signature:_____

Supervisor' s signature:_____

External Reviewers: Anonymous Reviewer

Anonymous Reviewer

Anonymous Reviewer

Anonymous Reviewer

Examining Committee Chairperson:

Ji XIANG \ Prof. \ Zhejiang Univ.

Examining Committee Members:

Chao XU \ A.P. \ Zhejiang Univ.

Wei JIN \ SN ENGR. \ Zhejiang Univ.

Date of oral defence： March 9, 2018

浙江大学研究生学位论文独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得 **浙江大学** 或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文作者签名：　　签字日期：　　年　　月　　日

学位论文版权使用授权书

本学位论文作者完全了解 **浙江大学** 有权保留并向国家有关部门或机构送交本论文的复印件和磁盘，允许论文被查阅和借阅。本人授权 **浙江大学** 可以将学位论文的全部或部分内容编入有关数据库进行检索和传播，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后适用本授权书）

学位论文作者签名：　　导师签名：

签字日期：　　年　　月　　日　　签字日期：　　年　　月　　日

致 谢

光阴似箭，两年多的研究生生活很快就要结束了，回首这段时光，在这一过程中老师、同学、朋友、家人都给予我了诸多帮助，在毕业论文即将完成之际，在此向诸位表示诚挚的感谢。

首先我要向我的导师黄文君老师表示衷心的感谢！在研究生阶段，黄老师无论是在科研方法还是为人处世方面都给了我很多的指导与帮助，黄老师在科研上严谨求是、精益求精的态度，对工作认真负责、一丝不苟的作风，对技术发自内心的热爱，这些都令我受益匪浅。在项目课题进行过程中，黄老师凭借渊博的专业知识和深厚的工程素养总能引导我采用合适的方法去发现问题与解决问题；在职业发展等方面黄老师也为我提供了很多悉心帮助和指导，特别是如何在将来的职场上成为一个优秀的开发人员方面黄老师为我树立了一个很好的学习榜样。在此我向黄老师表示最诚挚的感谢！

其次我要感谢实验室的同学在我读研期间对我的帮助。感谢李德文大师兄对我科研项目以及论文写作上的帮助和指导。感谢王睿、张经纬、曾志伟、周林波、温棋标、刘劲松、祁应梅、常亮、袁松、任旭东、孔繁望、孙泽标、陈梦迟、袁晓军、张阳阳等在平日学习生活中的关怀和帮助。

最后我要特别感谢我的家人和我的女朋友，你们在我背后给予我的关怀、支持与鼓励促使着我不断前进。

高明飞

2018 年 3 月于浙江大学玉泉校区

摘 要

车载安全预警系统可及时为驾驶员提供必要的行车安全预警信息以提高驾驶安全性，其包含若干子系统，如交通标志识别、超速预警等，而交通标志检测则是支撑诸多子系统的重要基础技术之一；本文就针对交通标志检测中基于颜色分割的定位算法及多线程任务调度策略这两项关键技术进行了研究，提出了适用于性能有限嵌入式系统的混合颜色分割策略及混合切换任务调度策略，并通过搭建嵌入式原型样机在实际道路环境中验证了方法的有效性。此外为更好的验证及评估交通标志检测算法的效果，本文建立了中国道路交通标志视频数据集，并将此数据集公开发布以供其他研究人员使用，这也是此领域目前唯一的中国公开数据集。

目前主流成熟的交通标志检测定位方法基本均是基于颜色及几何形状局部特征的，本文在此框架下对用于车载安全预警的交通标志检测中最为重要的红色及黄色分割方法展开了深入研究，针对已有主流颜色分割方法的不足提出了混合颜色分割策略，此策略通过若干线性分类器的组合实现了对红色及黄色准确高效的分割，分割效果优于目前常用的各方法且其算法执行速度与最简单的 RGB 阈值法相似，可保证安全预警算法在性能有限的小型嵌入式车载设备上依然有较好的实时性；在颜色分割基础上本文采用经典的 Hough 变换实现了对红色圆形交通标志的检测定位并在数据集上评估了算法的效果。

本文通过对交通标志检测识别问题进行建模分析提出可用采样间隔时间作为定量衡量此类系统实时性的指标，进而针对目前广泛使用的多核 CPU 提出了理论最优的理想多线程任务调度算法，此算法可显著降低采样间隔时间以提高系统实时性；不过理想任务调度算法实际无法实现，因此本文进一步提出了实际可实现的混合切换任务调度策略及动态更新参数估计策略；通过控制系统模型数值仿真及实际嵌入式原型样机上的测试验证均表明本文提出的方法可有效优化采样间隔时间分布以此提高系统实时性。

本文同时开发了基于 Qt 的算法验证平台软件及基于 Intel Joule 模块的嵌入式原型样机，并在其上验证了上述各方法的有效性，最后在校园环境及城市道路上分别进行了静态及动态系统集成测试；测试结果表明本文提出的方法可在小型嵌入式设备上满足系统实时性要求，在天气光照条件较好时检出率也相对较高，不过算法鲁棒性依然需要加强。

关键词：交通标志检测；车载安全预警；计算机视觉；颜色分割算法；多线程任务调度

Abstract

Vehicle-mounted early warning system can provide some necessary safety warning information for driver to improve driving safety, which includes a number of subsystems, such as traffic signs recognition, overspeed warning, etc. And real-time traffic sign detection is an important fundamental technology in these subsystems. In this paper, we mainly focus on color-based traffic sign location algorithm and hybrid switching multithreading task scheduling strategy, which are two key technologies of the real-time traffic sign detection. Some strategies which are suitable for the limited performance embedded system are proposed. And we also set up an embedded prototype to verify the effectiveness of these methods in the actual road environment. In order to test the effect of the traffic sign detection algorithm, we established a China traffic signs dataset and made it public available for other researchers, which is currently the only Chinese public dataset in this area.

At present, the mainstream traffic sign detection and localization methods are mainly based on the local features of color and geometric shape. In this framework, red and yellow segmentation methods, which is the most important color in traffic sign detection, are studied in depth. In this paper, the basic principle, implementation method and segmentation results of current used methods are studied and compared. The comparison results show that these methods have some limitations, therefore a hybrid color segmentation strategy is proposed, which is based on these existing methods. This strategy achieves accurate and efficient segmentation of red and yellow in traffic signs by using a combination of several linear classifiers. The segmentation result is superior to the commonly used methods and the speed of the algorithm is similarly the same as that of the simplest RGB threshold method, so that we can guarantee the real-time requirements is satisfied in the small vehicle-mounted embedded equipment with limited performance. Based on the color segmentation result, we use the classical Hough circle transform to detect and locate the red circular traffic sign and evaluate the effect of the algorithm on the data set.

In this paper, by modeling and analyzing traffic sign detection and recognition problems, we propose that sampling interval time can be used as an index to quantitatively measure the real-time performance of such systems. In addition, an optimal ideal multithreading task

scheduling algorithm is proposed, which can significantly reduce the sampling interval time to improve the real-time performance of the system. However, the ideal task scheduling algorithm cannot be realized in practice. Therefore, we propose a practical hybrid switching task scheduling strategy and dynamic update parameter estimation strategy. Numerical simulation based on control system model and the actual test on embedded prototype both show that the methods proposed in this paper is effective to optimize the sampling interval time distribution and can improve the system real-time performance.

We also develop an algorithm verification platform based on Qt and an embedded prototype based on Intel Joule module, and verifies the validity of the methods in this paper. Finally, the integrated test is carried out in the real road environment. The test results show that the methods proposed in this paper can meet the system real-time requirements in small embedded devices, and the detection rate is relatively high when the weather and light conditions are good. However, the robustness of the algorithm still needs to be strengthened.

Keywords: Traffic Sign Detection, Vehicle-Mounted Early Warning, Computer Vision, Color Segmentation Algorithm, Multithreading Task Scheduling

目 录

图目录

表目录

1 绪论

1.1 引言

随着综合国力的增强及人民生活水平的日益提高，我国机动车保有量持续增长，统计数据表明近 5 年来全国每年新增民用汽车超过 1000 万辆，到 2015 年底民用汽车拥有量已达 1.6 亿辆[1]，汽车已成为满足大众出行需求中必不可少的一环。但在享受汽车为我们出行带来便利的同时我们也要看到，交通事故也造成了巨大的人员伤亡及财产损失，仅在 2015 年一年间因交通事故造成的直接财产损失就高达 10 亿元[1]；因此如何更好的保障交通安全已成为一个极为重要和有价值的研究课题。

研究表明在所有的交通事故中超过 90%的事故是由于机动车驾驶员因素导致的，其中又有 30%以上的事故源于超速行驶、疏忽大意等原因[2]。特别是在车流量较少、路况相对较好的情况下，驾驶员经常会自然而然的提高车速而没有意识到此时已经超过了安全限速，而且路况的单调性也会让驾驶员放松警惕，在这种时候若前方道路突然出现异常状况就极有可能导致交通事故。这些情况下若能及时向驾驶员发出预警，提示驾驶员车速过快、路况异常、跟车距离过近等情况就可以在很大程度上避免事故的发生。

此类系统就是所谓的车载安全预警系统，其可为驾驶员提供必要的安全提示信息以此提高行车过程的安全性。车载安全预警系统已有很多商业化产品，然而在学术研究方面尚存在很多需要进一步深入研究的问题，本文就立足于此，对车载安全预警系统中很重要的交通标志检测问题所涉及的几项关键技术进行研究分析，并结合仿真与嵌入式原型样机上的实际测试验证研究结果的有效性。

1.2 研究背景及意义

车载安全预警系统作为一个关系到驾驶安全性的重要课题很早之前就吸引了研究人员的目光，相关研究可以追溯到欧共体（European Community，欧盟前身）于 1986 年底提出的普罗米修斯计划（PROMETHEUS Project），此计划是欧洲尤里卡计划（EUREKA）的一部分，旨在联合研发各种提升欧洲道路交通水平的技术，重点在于为提升驾驶安全性、环境友好程度、交通运输效率及经济效益提供新的解决方案[3]。此计划一共持续了接近 10 年，总投资共计 7.5 亿欧元，是相关领域研究中规模最大的研究项目[4]，高级辅助驾驶系统（Advanced Driver Assistance Systems, ADAS）就是普罗米修斯计划的一个重要

研究成果。ADAS 是一系列用以辅助驾驶员进行汽车驾驶以提高驾驶安全性及舒适度的子系统集合，自 1986 年至今 30 余年来各大汽车厂商对此都投入了很多人力物力进行研究，其范围相当广泛，目前已商用的主要 ADAS 子系统及其代表车型见表 1.1，从中可以看到车载安全预警系统是 ADAS 系统的一个子集，通常包括其中与驾驶安全性关系较为紧密的几个子系统，如交通标志识别、智能超速预警、前方碰撞预警、车道偏离预警等；而本文研究的交通标志检测则是交通标志识别、智能超速预警等多个子系统的重要基础技术之一。

表 1.1 主要已商用的 ADAS 系统

系统名称	对应英文名称	代表车型
自适应定速巡航(ACC)	Adaptive Cruise Control	长安福特新蒙迪欧
自适应前照灯(AFS)	Adaptive Front-lighting System	奔驰 E 级
自动泊车	Automatic parking	奔驰 GLC
夜视行人检测	Automotive night vision	宝马 7 系
盲区监测	Blind spot monitor	本田雅阁
变道盲区提示	Lane change assistance	奔驰 E 级
前方碰撞预警(FCW)	Forward Collision Warning	沃尔沃 S60L
车道偏离预警(LDW)	Lane Departure Warning	大众 CC
疲劳驾驶预警	BAWS	比亚迪 S6
十字路口辅助功能	Intersection assistant	奥迪 Q7
全景摄像头环绕视图	Surround view	凯迪拉克 CT
智能限速控制(ISA)	Intelligent Speed Adaptation	东风标致 4008
交通标识识别(TSR)	Traffic Sign Recognition	宝马 7 系
堵车辅助跟车	Traffic Jam Assistant	奥迪 Q7

宝马 7 系作为宝马公司最高端的系列，其最新车型 BMW 740i 及 BMW 750i 上搭载了最为先进的车载安全辅助系统，基本上囊括了表 1.1 中所有的功能，这作为行业标杆代表了汽车行业的最高水平，故本节中就先以其为例介绍车载安全预警系统的最新研究进展，以下涉及到的相关技术资料来源于宝马公司技术报告[5]。

宝马 7 系中安全预警与辅助驾驶主要是依靠安装在汽车各个部位上的诸多传感器进行环境感知来实现的，主要使用的传感器见图 1.1 及表 1.2。其中 KAFAS 立体摄像机负责采集进行图像采集，用以支持各种基于计算机视觉方法的安全预警功能，它是由两个安装于同一壳体内的高分辨率、高帧率 CMOS 摄像头及其处理单元构成的，其结构见图 1.2。出于篇幅考虑此处将重点关注 KAFAS 立体摄像机所实现的功能，其余一些通过雷达等实现的功能，如 ACC、盲区预警、夜视检测等就不多做说明，如有兴趣可进一步阅读文献[5]中的详细说明。

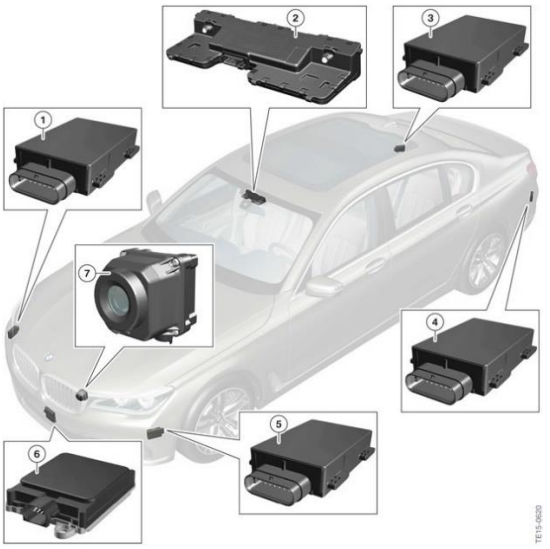

图 1.1 宝马 7 系辅助驾驶传感器分布图

图 1.2 KAFAS 立体摄像机结构

表 1.2 宝马 7 系辅助驾驶传感器列表

编号	传感器名称
1	右侧雷达传感器及控制单元（RSR）
2	KAFAS 立体摄像机
3	右侧盲区及车道变更警告系统（主控单元）
4	左侧盲区及车道变更警告系统（副控单元）
5	左侧雷达传感器及控制单元（RSL）
6	自适应定速巡航控制系统（ACC）
7	夜视系统红外摄像机

KAFAS 立体摄像机主要用于支持以下辅助功能：

(1) 带制动功能的碰撞预警（Collision warning with city braking function）；

(2) 带制动功能的行人预警（Pedestrian warning with city braking function）；

(3) 车道偏离预警（Lane departure warning）；

(4) 交通标志识别（Road sign recognition）。

从中可以看到这些功能的核心都在于目标识别，即从摄像头采集到的图像中通过计算机视觉的方法识别道路环境中的其他车辆、行人、车道、交通标志、路面状况等。碰撞预警包括车辆碰撞预警和行人碰撞预警，这是通过摄像头与雷达综合实现的——远距离时依靠雷达，近距离时依靠摄像头；二者的区别仅在于目标检测对象不同，对行人及车辆的检测结果分别见图 1.3 及图 1.4。车道偏离预警则完全依靠摄像头去检测道路标线以此确定车辆是否偏离了车道，若在没有操作转向灯的时候超出道路标线方向盘会有震动反馈，同时也会启动主动转向干预辅助保持车道。宝马 7 系上的交通标志识别可以识别限速标志牌及禁止超车标志牌，识别到的标志将会在仪表盘上显示出来，显示效果如图 1.5 所示;除此之外系统会自动结合天气、当前位置等信息综合判断当前最高限速，并在超速行驶时给出提醒。

不过 KAFAS 立体摄像机也有其局限性，在以下情况下它是无法正常工作的：

(1) 大雾、大雨或大雪等极端天气情况；

(2) 汽车行驶方向存在强光照（如太阳等）直射；

(3) 挡风玻璃上有遮挡物或过脏；

(4) 急转弯；

(5) 道路标识不全；

(6) 过于靠近前方物体（车辆或其他障碍物）。

比如在太阳直射的情况下，摄像头采集到的图像如图 1.6 所示，可以看到由于光学元件的物理限制，此时仅依靠摄像头采集到的图像显然是无法正确进行识别判断的。

以上只是宝马 7 系使用前置 KAFAS 立体摄像机就可以完成的车载安全预警功能，除这些功能外宝马 7 系还支持大量其它高级辅助功能，不过限于篇幅此处就不再介绍了。宝马 7 系充分体现了当前车载安全预警系统的最高技术水平，除此之外，特斯拉、沃尔沃、奔驰等的一些车型在这方面的技术水平也不遑多让。不过这些都是国外汽车厂商已经实现的产品，目前中国一汽、东风汽车、厦门金龙客车等国内汽车厂商虽然在此方面也有所研究[6]，然而尚未能推出可与宝马 7 系相媲美的产品，这也说明我国在这一方面

与国际先进水平间尚存在不小的差距，对于国内研究人员和汽车厂商来说，还存在很多需要研究的问题。

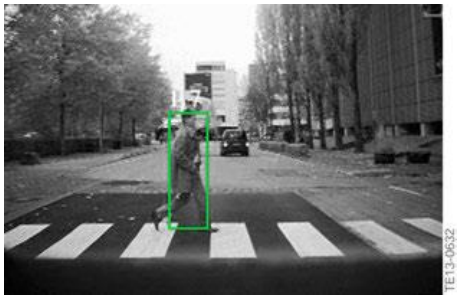

图 1.3 宝马 7 系对行人的检测效果

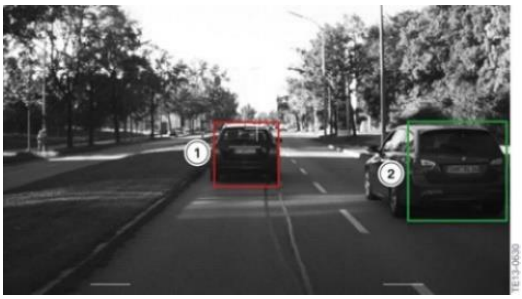

图 1.4 宝马 7 系对车辆的检测效果

图 1.5 宝马 7 系限速标志识别结果

图 1.6 强光照下 KAFAS 摄像头采集的图像

表 1.1 中所列举的 ADAS 系统都是整车的一部分，其传感器、计算单元、显示设备等与汽车其它部分无缝结合在一起，因此用户体验相当好。不过除此之外车载安全预警系统还可以作为一个可灵活的安装于已有车辆上的独立嵌入式设备存在，这方面市场上也已存在一些相关产品，此类产品主要以智能行车记录仪的形态出现，比如小蚁智能行车记录仪[7]，其产品的外观见图 1.7。

图 1.7 小蚁智能行车记录仪

这类产品都可以实现基础的行车记录功能，一般在此基础上添加了前方碰撞预警与车道偏离预警功能，然而实际测试表明这类产品的预警正确率不是很高，特别是在城市道路情况下表现并不理想[8]，且尚未有能实现交通标志检测识别的产品存在。不同厂商

的智能行车记录仪同质化程度极高且配套的 ADAS 功能也基本相同，这是由于它们大多都选用了同样的主控方案造成的。智能行车记录仪的核心技术是掌握在芯片厂家手中的，业界最先进的解决方案来自美国安霸公司（Ambarella）。安霸公司于 2004 年由王奉民（Fermi Wang）在硅谷组建，主要提供低功耗高清图像视频处理解决方案，在行车记录仪领域牢牢占据了绝大部分高端市场份额；目前安霸公司推出的面向车载视频领域最先进的 SoC（System on Chip，片上系统）芯片为 A12A 系列，其功能框图见图 1.8。

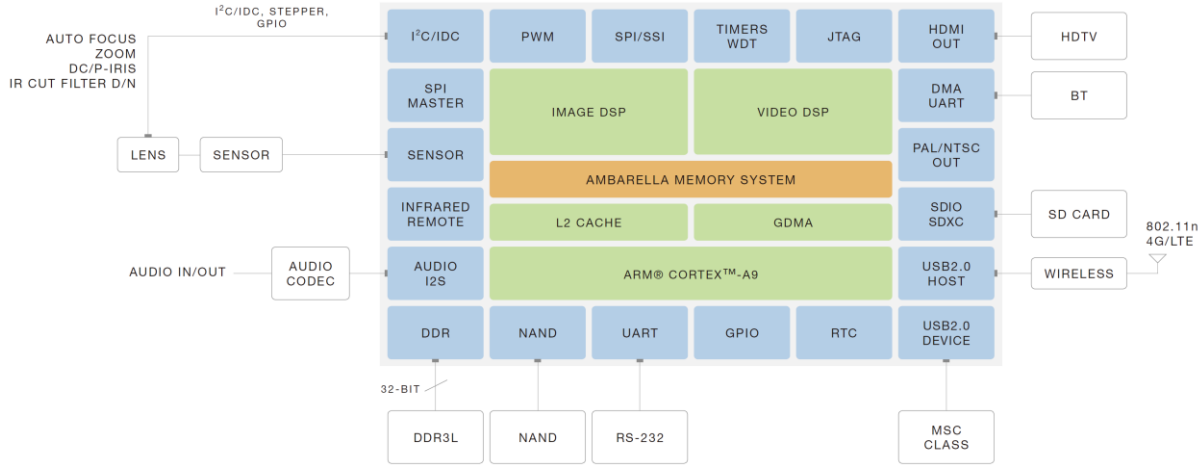

图 1.8 安霸 A12A SoC 功能框图

根据安霸公司公开的技术宣传资料[9]，A12A 系列 SoC 内置一个 792MHz 的 ARM Cortex-A9 CPU 核及两个其自行研发的图像及视频 DSP 核，在此基础上实现了车道偏离检测（LDW）、前车碰撞预警（FCW）、前车移动检测（FCMD）、弱光报警（LLW）功能；除此之外 A12A 系列还集成了 Wi-Fi 及 LTE 模块可极为方便的进行网络连接。不过安霸仅向其合作伙伴提供详细的技术支持文档，其技术实现细节通过公开渠道无法获取到，而且也无法在其上添加交通标志检测识别这类自定义功能。由于商业产品的保密性尽管目前已有很多成熟的商用产品存在，然而对于学术界来说车载安全预警系统中一些底层关键技术上依然有很大的研究空间和价值，特别是交通标志检测识别目前在小型嵌入式车载安全预警设备中基本未有实现。

上文提及的所有车载安全预警系统基本都是部分甚至是全部基于视觉的，这是因为视觉能提供最为完整丰富的环境信息。研究人员很早就充分认识到了视觉信息的重要性，在计算机出现后不久就产生了计算机视觉这门学科，计算机视觉技术发轫于上世纪中叶，

历经六十余载的发展，时至今日已建立起一套枝繁叶茂的理论体系。最初研究人员希望先从图像中恢复出物体的三维形态，之后再进行识别与判断；上世纪 80 年代，人们逐渐发现其实不需要重构三维形态也可以直接去理解图像，此时出现了很多几何与代数方法，通过构造先验表征去进行图像匹配；上世纪 90 年代，统计方法与局部特征的出现及流行成为了计算机视觉的发展史上的一个里程碑，此时提出的很多方法迄今依然有着广泛的应用[10, 11]；进入新千年后，基于大数据的机器学习方法被应用于计算机视觉中，使得识别的准确率得到了极大的提升[12]；最近十年来，深度学习、人工智能等领域迅猛的发展也使得计算机视觉进入了一个激动人心的新时代[13, 14]。计算机视觉研究中硕果累累的理论成果也在实践中得到了充分应用，其应用场景遍布各行各业，特别是在人脸识别、车牌识别、目标跟踪等方面已取得了大规模商业化应用，这些应用充分展现了计算机视觉领域辉煌的成就与光明的前景。

尤为值得一提的是，目前很热门的自动驾驶技术也很大程度上依赖于计算机视觉技术。比如 Google 公司的自动驾驶汽车项目（Self-Driving Car Project）中就综合使用了激光雷达和摄像头去感知周围的环境；Tesla 作为目前典型的正式上市的具有初级自动驾驶功能（Autopilot）的汽车没有使用激光雷达，完全依赖于摄像头图像识别进行环境感知及路况判断；百度的 Apollo 自动驾驶平台则可以将相机的视觉图像与来自毫米波雷达、激光雷达、组合惯导等的数据进行融合。不过自动驾驶技术本身目前尚不成熟，国内外都有驾驶员因为过分依赖特斯拉的 Autopilot 而导致车祸的报道。2016 年 5 月 7 日，美国佛罗里达州一车主在使用自动驾驶模式时发生事故死亡；2016 年 1 月，河北邯郸有车主驾驶特斯拉汽车发生车祸身亡，据报导这也与自动驾驶功能有关。驾驶安全直接关系到人的生命安全，在现阶段看来自动驾驶技术的安全性和鲁棒性尚不完美且立法与事故责任归属等问题也未能解决，出于对生命的珍视各大汽车厂商对此也都持谨慎态度，自动驾驶技术要真正得以推广还有很长的路要走。不过本文研究的交通标志检测技术也同样是自动驾驶研究中必不可少的基础技术之一，相信本文的研究内容对于自动驾驶领域的研究来说也会有一定参考价值。

无论是车载 ADAS 系统还是独立的智能行车记录仪都属于广义上的嵌入式系统，所谓嵌入式系统一个比较通用的定义是：嵌入到对象体系中的专用计算机应用系统，可见本文研究的车载安全预警系统无疑是属于一个典型的嵌入式系统。嵌入式系统一个很重要的特征是其系统资源与性能一般都比较有限，然而与此同时对系统响应的实时性要求却通常都比较高，这样一种矛盾的境地就对算法选择与系统优化提出了很高的要求。当

然由于嵌入式系统的广泛性，智能行车记录仪与车载 ADAS 系统在性能上存在着极大的差异，前者一般基于上文所述的基于 ARM 的专用 SoC 芯片，其性能与通用计算机相比存在较大差距；而后者一般使用专用的车载电脑实现，如 Intel 最新的车载电脑 GO，其配置包括 2 颗服务器级别的 Xeon 处理器及 2 颗 Arria 10 FPGA，可提供多达 20 余个高主频 CPU 核心及基于 FPGA 的 DSP 硬核等；NVIDIA 预计于 2018 年发布的 DRIVE PX Pegasus 平台则包含 2 颗 Tegra Xavier CPU 及 4 颗最新 Volta 架构 GPU，其峰值计算能力高达 320 TOPS（Tera Operations Per Second, 每秒执行万亿条指令数）；这类车载电脑的性能其实大幅优于一般的通用计算机系统，本文的研究主要针对诸如智能行车记录仪这类性能较为有限的嵌入式系统进行，重点探讨如何保证交通标志检测算法在其上的实时性，不过对性能优化方面的研究就算是对于高性能平台来说也是颇有意义的，毕竟如何使用合适的算法高效的解决问题、如何充分发挥硬件的潜力始终是研究人员在不懈探索和追求的。

对嵌入式车载安全预警系统而言系统的实时性约束是极为重要的，这也是此系统不同于其它离线图像处理系统的地方，提高系统的实时性一般可以从三方面着手：提升硬件平台的处理性能；改进优化所用算法及代码的实现效率；进行并行化优化。提升硬件性能是加快系统响应速度最直接的办法，然而对于嵌入式系统而言考虑到成本及其它因素的制约很多时候硬件平台的选取上存在较大的局限性；并行化优化则是针对当前广泛使用 CMP（Chip Multi-Processors, 单芯片多处理器，即多核 CPU）架构系统上最行之有效的性能优化手段，而在诸多并行化手段中多线程技术是最直接且最易于实现的。

1.3 课题研究目标

从上一节对相关研究背景的分析中可以看到，车载安全预警系统作为 ADAS 系统的一个重要组成部分其范畴相当广泛，涉及到的研究内容也极为丰富。根据使用功能车载安全预警系统可分解为若干个功能模块，具体见表 1.1，其中碰撞预警、车道偏离预警及交通标志识别这三个领域是目前研究得最多及已取得最广泛应用的。在这些诸多模块中对交通标志的准确识别是正确理解周围道路环境的基础，而对交通标志的检测定位又是识别的基础；由于大部分交通标志都有固定的颜色和形状，如红色的圆、黄色的三角形等，借助这些局部特征进行交通标志检测定位是目前的主流做法，因此本文将对颜色分割算法及在此基础上进行的定位算法进行研究；除此之外考虑到目前 CMP 架构的多核 CPU 已十分普遍，本文也将对如何充分利用多核 CPU 的性能以提高系统实时性进行研

究。下面将对本文的具体研究目标进行简要阐述。

(1) 针对交通标志检测识别领域目前缺乏针对中国道路交通标志的公开视频数据集的研究现状，本文研究中需要自行建立中国道路交通标志数据集；

(2) 对基于颜色分割的交通标志定位算法进行研究，重点对目前已有颜色分割方法的实现方法及分割效果进行比较研究，并在此基础上实现圆形交通标志的定位；

(3) 针对交通标志检测定位问题进行建模分析从而明确其实时性要求及定量指标，进而设计合适的多线程任务调度策略以提高系统的实时性；

(4) 开发算法仿真验证软件用以评估算法在数据集上的效果，此软件需通过良好的软件架构设计保证后续的可维护性，并能够较为方便的添加新功能与新算法；

(5) 搭建嵌入式系统原型样机并在实际道路上进行的系统集成测试，以此验证以上各方法的有效性。

1.4 国内外研究现状

本节首先交通标志检测识别系统整体上的研究进展进行综述，之后针对本文所研究的几个关键技术分别对其国内外研究现状进行具体综述。

1.4.1 交通标志检测识别系统研究综述

交通标志检测识别作为诸多安全预警功能的基础技术是一个极为重要的研究内容。本文“1.2 研究背景及意义”一节中介绍了计算机视觉技术的发展历程，而交通标志检测识别作为一个经典的计算机视觉与模式分类问题，其研究进展基本上也是沿着这一脉络逐步深入的。早在 1964 年计算机视觉的发展初期，就有研究人员对交通标志的识别展开了研究[15]，之后随着计算机视觉技术的发展各种新方法也被应用于交通标志的识别中，目前最为成熟的方法是基于局部特征的统计分类方法。在交通标志定位方面方法相对较为单一，主要有基于交通标志形状的图像分割算法[16]与基于颜色空间的图像分割算法[17]，作为本文的主要研究内容，这部分算法的研究综述将在下一节中具体展开。在特征提取方面，研究人员进行了很多不同的尝试，如方向梯度直方图（Histogram of Oriented Gradients, HOG）特征[18]、Hu 及 Zernike 不变矩（Invariant Moment）特征[19]、Gabor 小波主特征[20]、加速稳健特征（Speeded Up Robust Features, SURF）[21]、主成分分析（Principal Component Analysis, PCA）特征[22]等，其中使用得最广泛的是不变矩特征。在分类器设计方面，最简单的方法是聚类算法，如 Sompoch 等人采用的最近邻域法[23]、Betke 等人

采用的相似系数法[24]、De La Escalera 等人采用的互相关性法[25]等；不过这些方法计算量较大，故研究人员又提出了新的分类方法，如支持向量机（Support Vector Machine, SVM）[18, 26]、极限学习机（Extreme Learning Machine, ELM）[27]、决策树[28]等方法；近年来，随着人工智能、深度学习等技术的发展，研究人员也使用了这些技术来进行交通标志识别，如 Dan Cireşan 等人使用了多列深度神经网络（Multi-column deep neural network）[29]；Junqi Jin 等人使用了卷积神经网络（Convolutional Neural Networks, CNN）[30]；Yujun Zeng 等人使用了核极限学习机（Kernel Extreme Learning Machines）[31]；JaWon Gim 等人采用的随机森林（Random Forest）[32, 33]等。Mogelmose 等人对当前交通标志识别中分割（Segmentation）、检测（Detection）、分类（Classification）及追踪（Tracking）这几个主要步骤中所采用的主流策略和方法进行了较为全面的综述，并对现有方法进行了评估比较[34]，对此有兴趣的读者可进一步参阅其论文。

不过上述研究大部分都停留在理论研究与算法仿真上，实际使用嵌入式系统实现出原型产品的研究相对较少，而在嵌入式系统性能相对有限的情况下要保证交通标志检测及其它安全预警算法的实时性也是一个重要的研究课题，这方面的研究中仲崇峰使用 Samsung S3C2440 ARM9 处理器实现了基本的交通标志识别，不过只能对三种特定的交通标志牌进行分类，局限性较大[35]；Yan Han 等人使用 Zynq-7000 FPGA 和 ARM 协同工作实现了交通标志检测与识别，与其余 FPGA 实现相比执行效率提升了 8 倍，不过由于 FPGA 相对较高的成本也限制了其应用推广[36]；Phalguni 等人使用 TMS320C6748 DSP 实现了交通标志的实时检测定位[37]。从已有研究来看对如何进行系统优化以保证算法的实时性方面基本没有相关研究，特别是多线程并行任务调度策略上存在研究空白。

1.4.2 颜色分割及交通标志定位算法研究综述

对于交通标志检测定位来说颜色分割与几何形状特征定位可以独立使用也可结合使用，不过单独使用颜色分割进行定位的做法较少，一般在进行颜色分割后均需继续根据几何形状特征进行定位；不过颜色分割并不是必要的，很多研究中并没有这一步而是直接根据几何图像特征进行匹配定位。

对于颜色分割来说，其目的在于从目标图像中提取出红色、黄色、蓝色等特征颜色区域，目前绝大多数研究人员均采用了各种颜色空间转换策略，如 HSV/HSI 颜色空间[38]、CIE 颜色空间[39]、LUV 颜色空间[40]等，其中最为常用的策略是提取 HSV/HSI 空间中的色调（Hue）分量进行颜色分割；除此之外直接在原始 RGB 空间中进行颜色分割也是一

种常用方案，使用的策略一般是线性阈值法，即直接根据经验公式判断颜色[41]，也有研究人员使用了其他方法，如 Eunchong Lee 等人使用了查找表进行颜色分割[42]。以上方法均可具体区分出多种不同颜色，不过也有研究人员针对交通标志检测这个具体应用另辟蹊径，Asakura 等人于 2000 年提出了一个专门用于交通标志提取与检测的 SVF（Simple Vector Filter, 简单向量滤波器）算法[43]，此方法并不能精确的对不同颜色进行分类，其作用是将较为“鲜艳”的颜色提取出来，考虑到交通标志的颜色一般也都较为鲜艳，此方法以极小的运算量实现了很不错的效果。面对这么多种不同的颜色分割方法如何评价其优劣也成了一个值得研究的问题，尽管基于 HSV/HSI 颜色空间的分割方法是目前的主流，然而亦有一些研究人员指出此方法并不比直接在 RGB 空间中进行分割效果更好，且还会引入不必要的运算量，如 Gómez-Moreno 等人在其文章中通过实际实验测试得出了这一结论[44]；不过也有完全相反的结论，如 Fleyeh 认为考虑到光照及阴影，HSV 空间与其它颜色空间相比更具鲁棒性，其效果也是最优的[45]。另外考虑到颜色分割其实就是一个典型的模式分类问题，也有研究人员在探索是否可以用已有的成熟模式分类方法来解决此问题，如 Lillo-Castellano 等人使用 k 近邻法及 SVM 对交通标志中常见的多种颜色进行了分类，并对其效果进行了比较评估[46]；Marinas 等人通过建立混合高斯模型使用朴素贝叶斯分类器进行道路交通标志颜色分割[47]；不局限于交通标志识别的话，其它领域的颜色分割中也有使用此思路进行研究的，如浙江理工大学的常卫使用了 SVM 对纺织品染料颜色进行了分类[48]。

对于几何形状定位来说，基本检测方法有两种——基于掩膜的模板匹配法和 Hough 变换。模板匹配法的基本原理是使用待检测图形的标准图片作为掩膜，可根据实际需要对其进行缩放、旋转、变形等操作，之后在待检测图片中遍历寻找此模板，当图片中的某一部分与模板的相似度高于阈值时即认为找到目标图形。此方法的研究重点在于相似度的判断上，Giosan 等人使用了 Chamfer 距离作为判断依据[49]；Shuihua Wang 等人使用了相关系数作为判断依据[50]；Junhee Youn 等人则使用了 Hausdorff 距离作为相似性测度[51]。模板匹配法最大的问题在于其鲁棒性不佳，很难找到一个适用于各种情况下的通用模板，故对于交通标志检测来说使用得更多的方法是 Hough 变换。Hough 变换可以用于检测直线、圆及其它形状，由于交通标志形状特征一般都较为明显，使用 Hough 变换已成为一种极为普遍的做法[52, 53]，然而 Hough 变换存在计算量较大的缺陷，针对这一问题研究人员在原始 Hough 变换的基础上进行了很多改进，如 Eickeler 等人用来检测圆形限速标志及正多边形的 Hough 变换变种快速径向对称检测器[54]；何江萍等人利用快速辐射

对称性算法与 Hough 变换结合进行三角形交通标志检测[55]；使用边缘链码法先对边缘的形状、大小、连通性等进行判断后再进行 Hough 变换也是一种较多研究人员感兴趣的方法[56]。除了模板匹配法与 Hough 变换外亦有研究人员使用了其它方法，如最小二乘拟合[57]，快速傅里叶变换（FFT）[58]、对称性分析[59]等。

1.4.3 并行化及线程调度策略研究综述

正如前文所述，提高系统实时性一个常用且重要的手段就是对系统进行并行化优化，一个系统的并行化可按照其粒度分为几个层次，划分方法在不同文献中不尽相同[60, 61]，不过一般来说可粗略分为指令级别和线程级别的并行化。

指令级别并行化（Instruction-Level Parallelism, ILP）主要由 CPU 硬件实现，典型技术包括：流水线（Pipeline）、乱序执行（Out-of-order execution）、超标量（Superscalar）、单指令多数据（Single Instruction Multiple Data, SIMD）等。这些技术基本都属于 CPU 内部晶体管逻辑门层面的设计，除 SIMD 外上层软件无法感知其存在，也基本无法干涉其运行，仅能通过编译器优化生成有助于 CPU 充分发挥这些特性的代码以此提高运行效率；而 SIMD 作为现代 CPU 的重要特性，则需要软硬件协同配合才能发挥其最大作用。不同于一般的 CPU 指令，SIMD 指令使用多个执行单元同时对一组数据中的每一个分别执行相同的操作，故又将其称为矢量指令。SIMD 一般以指令集的形式提供，典型的 SIMD 指令集有：Intel 的 MMX、SSE、SSE2、SSE3、SSE4、AVX、FMA 等；AMD 的 3DNow!、SSE5、XOP、FMA4 等；ARM 的 NEON。凭借以上诸多技术，当前主流 CPU 已经可以实现在单个时钟周期内单核平均执行超过一条的指令，定量表征 CPU 这一性能的常用指标为 DMIPS/MHz[62]，对于当前最新的消费级桌面处理器而言，AMD Ryzen 7 1800X 和 Intel Core i9-7980XE 可分别实现单核 10.6DMIPS/MHz 及 12.9DMIPS/MHz 的性能[63]；对于嵌入式处理器来说，最新的 ARM Cortex-A73 亦可实现单核 4.8 DMIPS/MHz 的性能[64]。不过要实现更高程度的指令级别并行化主要是 CPU 设计人员的工作，对于上层应用而言一般不需要关心这一层次的优化，在视频图像处理领域中对这方面的研究主要集中在 FPGA 及 ASIC 设计[65]或算法 SIMD 优化加速[66, 67]上，本文不会涉及此方面的工作，读者若对此有兴趣可参考相关文献。

线程级别并行化（Thread-Level Parallelism, TLP）则主要由上层软件实现，这是针对拥有超过一个 CPU 核心的系统中最行之有效的优化加速手段之一。多 CPU 系统的历史可以追溯到上世纪 60 年代诞生的 Burroughs D825[68]，此类系统中包含若干个完全一样

的 CPU，各 CPU 间通过总线进行通信并共享内存，一般将这种系统结构称为 SMP（Symmetrical Multi-Processing, 对称多处理）架构，针对 SMP 架构存在的扩容性不佳的问题，之后又产生了 NUMA（Non-Uniform Memory Access, 非一致存储访问）架构和 MPP（Massive Parallel Processing, 海量并行处理）架构，这三种架构就是当前商用服务器的三种主流架构，在这类系统中物理 CPU 的数量都是超过一个的，若将 SMP 架构中的多个 CPU 集成到同一芯片内那就成为了 CMP（Chip Multiprocessors, 单芯片多处理器），也就是通常所说的多核处理器。上世纪 90 年代 IBM、Sun 等公司就推出过针对高端服务器市场的 CMP，然而由于价格过高和应用面窄并未引起广泛关注；在本世纪前，CPU 性能的提高主要依赖于主频的不断提高，然而在 2005 年左右 Intel 与 AMD 的 CPU 主频均已接近 4GHz，此时研究人员发现继续提高主频面临着极大的困难，一方面是功耗的急剧增加导致散热困难，另一方面是过长的流水线反而会带来单位主频性能的下降，这就使得提高主频已不能显著提高系统整体性能。面对这一发展瓶颈研究人员将目光转向了 CMP 架构，2006 年发布的 Intel Core 被视为多核处理器大规模推广的开始，至此 CMP 逐步成为 CPU 领域的主流发展方向，今天就算是在嵌入式移动端所用的 CPU 也已包含多个 CPU 内核。在当前如此普遍的 CMP 上，如果只使用传统的单线程方式运行程序算法则只会用到一个 CPU 核心，其余 CPU 核心均处于闲置状态，此时 CPU 占用率会很低，故多线程就成为了一个自然而然的选择，这也就是所谓的线程级别并行化；与指令级别并行化相比线程级别并行化更多依赖于软件实现，其灵活性和易用性都更好一些。

多线程技术的研究中最为重要的两个问题是线程同步与线程调度，线程同步的核心是如何解决数据访问冲突问题，当前主流解决方案是采用锁（Lock）机制，即一个时刻只允许一个线程访问特定数据以此避免访问冲突的发生，针对多样化的应用场景发展出了互斥量（Mutex）、自旋锁（Spin Lock）等多种不同实现方式；然而使用锁本身会降低系统系统性能，故近年来无锁（Lock-Free）实现成为了研究人员更为关注的领域[69]；由于本文涉及的多线程系统较为简单，使用经典的锁机制即可很好的解决数据访问冲突，故本文将不会过多的讨论线程同步问题。线程调度的核心是如何设计性能更优的调度器，绝大部分研究均以负载均衡、最高并行度（即最高 CPU 使用率）或任务处理时间最短为目标[70-72]，通用调度器设计的主要困难在于任务执行时间及数量均是未知的，特别是对于操作系统底层的核心任务调度器而言其设计是很富有挑战性的。然而本文研究的问题与这类通用调度问题不同，其任务是确定的且始终在不断重复执行，故线程调度的目的也并不是最高并行度，对于这一具体问题目前尚未有研究人员对其进行过研究讨论。

1.5 主要研究内容与课题创新点

根据前文提出的课题研究目标并通过对国内外研究现状进行分析可确定本文的主要研究内容，主要包括基于颜色分割的交通标志定位算法及混合切换多线程任务调度策略两大方面，具体如下：

（1）针对交通标志检测识别领域缺少针对中国道路交通标志公开视频数据集的研究现状，建立基于视频片段的中国道路交通标志数据集，并公开发布此数据集供其他研究人员使用。

（2）在颜色分割方面对目前主流的 RGB 阈值法、HSI 空间转换法及 SVF 方法进行比较研究，并对 HSI 空间转换算法不同实现方式的效果及执行速度进行对比，在此基础上针对红色及黄色分割问题提出一种新的混合颜色分割策略，并将此策略的分割效果及算法执行时间与已有方法进行对比；此外采用成熟的 Hough 圆检测方法实现对红色圆形禁令标志的检测定位并在自行建立的数据集上验证算法的有效性。

（3）通过建模分析得出交通标志检测识别问题中系统实时性需求的定量指标，并讨论通用多线程调度策略在解决此问题上存在的局限性，进而由推导分析提出理论最优的理想任务调度算法，针对此算法实际不可实现的问题提出了两种可实现的调度策略，并在此基础上将其融合形成混合切换调度策略并给出此策略在 POSIX 标准下的实现方法；此过程中通过建立控制系统模型进行数值仿真对各调度策略的性能进行分析评估。

（4）开发基于 Windows PC 平台的算法验证软件，采用异步非阻塞处理流程及多线程同步通信策略保证主界面随时均能响应用户请求不会出现无响应情况。

（5）对不同嵌入式平台的性能进行比较以此确定原型样机最合适的平台，并在此基础上选用合适的外部模块（显示屏、摄像头等）并完成硬件结构设计以搭建完整的嵌入式系统原型样机，使用此原型样机对基于颜色分割的交通标志定位算法及混合切换多线程任务调度策略在实际道路上进行集成测试验证。

本文主要创新点在于：

（1）在对比分析了目前已有颜色分割方法的基础上，提出了一种针对红色及黄色分割问题的混合颜色分割策略，此策略融合了 RGB 阈值法、HSI 空间转换阈值法及 SVF 分割方法的优点，通过测试验证表明其分割效果优于单一方法，特别是在黄色分类问题上很好的解决了使用普遍使用的 HSI 空间转换阈值法对黄色及绿色区分度不够的问题；除此之外通过算法分析优化使用简单的线性分类器实现了此分割方法，避免了经典 HSI 颜

色空间转换过程中的非线性环节造成的性能瓶颈，整体算法执行效率与 RGB 阈值法相近，大幅优于 HSI 空间转换法。

（2）通过分析交通标志检测识别系统的实时性要求提出了使用采样间隔时间作为此类系统的实时性度量指标，并在此基础上提出了一种混合切换多线程任务调度策略，通过控制系统模型数值仿真及原型样机上实际测试验证均表明此调度策略能有效的改善采样间隔时间的统计分布情况，使整个系统的采样间隔时间更为平均以此实现提高系统实时性的目的。本文提出的混合切换多线程任务调度策略不仅适用于交通标志检测识别，在类似的检测识别系统中均可采用此策略，目前的多线程任务调度策略研究中基本没有涉及对线程间隔时间进行控制的研究，本文对此进行了一些探索性工作。

（3）建立并公开发布了一个中国道路交通标志视频数据集，弥补了目前尚未有中国道路交通标志公开数据集存在的研究空白，此数据集由 600 余段采集自真实道路的高清视频片段组成，且覆盖多种天气光照情况及中国各种路况。

1.6 论文结构安排

根据上一节中课题主要研究内容可确定全文的结构安排如下：

第一章，绪论。本章主要介绍本文相关的研究背景及意义、课题研究目标，针对研究目标分析相关领域当前国内外的研究现状，在此基础上介绍本文的主要研究内容与课题创新点，并给出了全文的章节结构安排。

第二章，基于颜色分割的交通标志定位算法。本章首先简要介绍我们建立的中国道路交通标志公开数据集的基本情况，之后重点对交通标志识别中常用的几种颜色分割方法进行比较研究，在此基础上提出混合颜色分割策略并对其性能进行评估分析，最后介绍基于 Hough 变换的圆形交通标志定位算法并在测试数据集上验证算法的有效性。

第三章，混合切换多线程任务调度策略。本章首先对交通标志检测识别系统的实时性要求进行建模分析，之后提出了理论最优的理想任务调度算法，针对此算法实际无法实现的问题进一步比较了多种任务调度算法后提出了一种混合切换任务调度策略，并设计了一种动态更新参数估计方法，最后给出了完整的任务调度策略在 POSIX 标准下的程序框架实现方法并在嵌入式原型样机上对算法的性能进行了测试验证。

第四章，仿真验证平台及嵌入式原型的搭建与系统集成测试。本章主要对算法验证平台软件的开发及嵌入式原型样机的搭建方法进行了介绍，并对摄像头输出分辨率优化问题进行了讨论，最后在嵌入式原型样机上对本文前两章中的算法进行了校园环境及城

市道路环境下的集成测试验证，并对验证结果进行了评估分析。

第五章，总结与展望。对本文的研究内容及研究成果进行总结，并分析其中存在的不足及将来的研究工作中需要进一步改进的地方。

2 基于颜色分割的交通标志定位算法

针对交通标志检测识别领域目前尚缺少完善的针对中国道路交通标志公开数据集的现状，本章将首先对自行建立的中国道路交通标志公开视频数据集进行介绍；之后针对红色与黄色这两种交通标志特征颜色的分割方法进行研究，并提出一种混合颜色分割策略；最后介绍基于 Hough 变换的圆形交通标志定位算法并在数据集上测试验证算法的有效性。

2.1 中国道路交通标志公开视频数据集的建立

对于基本所有道路交通领域的计算机视觉研究而言，测试数据集都是极为重要的，目前已有的公开数据集基本都是国外研究人员贡献的。Sebastian Houben 等人在 2013 年 IJCNN（International Joint Conference on Neural Networks）会议的附属比赛上公开了 GTSDB（German Traffic Sign Detection Benchmark，德国交通标志检测数据集）及 GTSRB（German Traffic Sign Recognition Benchmark，德国交通标志识别数据集），这是当前公开领域影响力最大的交通标志数据集[73]，其中 GTSDB 侧重于交通标志的检测定位，GTSRB 侧重于交通标志的识别；其余世界各地的公开数据集还有瑞典交通标志数据集（The Swedish Traffic Signs Dataset）[74]、比利时交通标志数据集（The Belgium Traffic Sign Dataset）[75]、美国交通标志数据集（LISA Dataset）[34]等。不过这些数据集中基本都只有图片而没有视频，由于仅靠静态的图片无法反映车辆经过交通标志这一完整的动态过程，因此对用于车载安全预警系统中的交通标志检测来说视频序列会更为有用；且上述几个数据集都是针对国外交通标志和路况的，国内道路交通标志公开数据集基本没有，能检索到的仅有中国科学院合肥物质科学研究院与西安交通大学人工智能与机器人研究所曾经联合发布过一个公开测试数据库，然而目前下载链接已失效[76]。正是由于中国道路交通标志公开视频数据集的匮乏，为支撑本文及课题组后续相关研究我们自行建立了一个中国交通标志视频数据集用于仿真分析及其它研究，同时出于学术共享精神为便于相关领域研究人员开展研究，我们将此数据集公开发布，有需要的读者可从以下百度云盘链接下载：

https://pan.baidu.com/s/1jIgUG8u

本节将对此数据集的基本情况进行介绍。

2.1.1 视频数据采集设备及覆盖范围

数据集原始视频数据由凯立德 CK55 行车记录仪采集得到，设备外观图片见图 2.1，核心技术参数见表 2.1。

表 2.1 凯立德 CK55 行车记录仪核心技术参数表

项目	描述
主控芯片方案	安霸 A7LA50
COMS 传感器型号	OV4689 4.0MP
光圈	F2.0
镜头广角	170°
图像分辨率	最高 1296P (2304*1296)，实际使用 1080P (1920*1080)
图像帧率	30fps
GPS 模块型号	AT1583C

图 2.1 凯立德 CK55 行车记录仪

原始视频数据主要采集于 2017 年 1 月~2 月春节期间，共计 30 余小时，后续整理过程中从中手工截取出包含交通标志的高清视频片段 619 段，这些视频片段来源于日常行车记录，很具有代表性和实际意义。由于凯立德 CK55 行车记录仪带有 GPS 模块，所以得到的视频片段也同时带有地理位置坐标信息和速度信息，这些信息都是目前公开数据集中无法获取的。数据集中 619 段视频片段覆盖云南省昆明市主城区（图 2.2）、昆明太平镇及 G56 杭瑞高速、昆明至周边村县 X213 等县道、G80 广昆高速、G8011 开河高速、红河哈尼族彝族自治州蒙自市至屏边苗族自治县 G326 国道、屏边大围山国家自然保护区及 S235 省道（图 2.3）等，限于论文篇幅此处只给出了两幅地图，地图中每一个蓝色标记点即代表一段视频片段，在发布的数据集中包含索引文件，其中整理了每一段视频的坐标位置，若有需要可使用 Google 地图定位具体位置。下面将对数据集中视频片段的

道路情况、天气情况、车速等详细信息做一简单统计分析，同样可以从数据集索引文件中获取每段视频对应的具体信息。

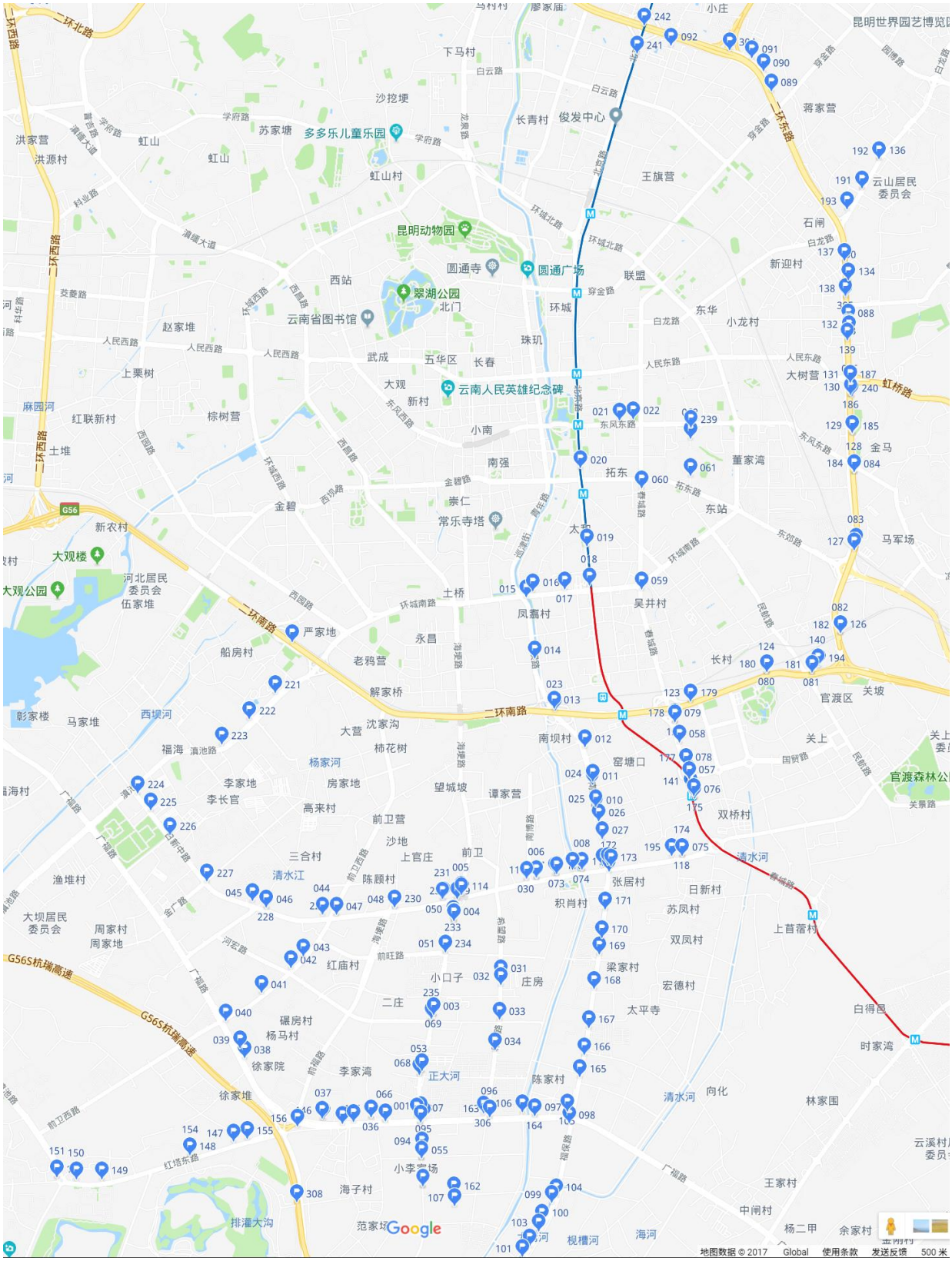

图 2.2 数据集视频片段地理位置分布图——昆明主城区

图 2.3 数据集视频片段地理位置分布图——屏边大围山及 S235 省道

2.1.2 数据集视频片段统计分析

根据道路种类与天气光照情况对数据集中的视频片段进行分类，所得结果见表 2.2；根据当前车速进行统计可得到图 2.4，从中可以看到数据集中的视频片段基本覆盖了各种天气、各种道路种类与各种车速，与现有公开或其他研究人员论文中使用的非公开数据集相比更为完善，且均为高清（1080P）视频片段，对于研究仿真验证来说极为有用。

表 2.2 数据集视频片段路况及天气光照情况统计表

	城市道路	乡镇道路	高速公路	合计
晴天	108 (17%)	132 (21%)	57 (9%)	297 (48%)
多云	88 (14%)	0	0	88 (14%)
阴天	11 (2%)	161 (26%)	0	172 (28%)
大雾	0	16 (3%)	0	16 (3%)
夜间、凌晨	28 (5%)	0	8 (1%)	36 (6%)
凌晨大雾	0	10 (2%)	0	10 (2%)
合计	235 (38%)	319 (52%)	65 (11%)	619 (100%)

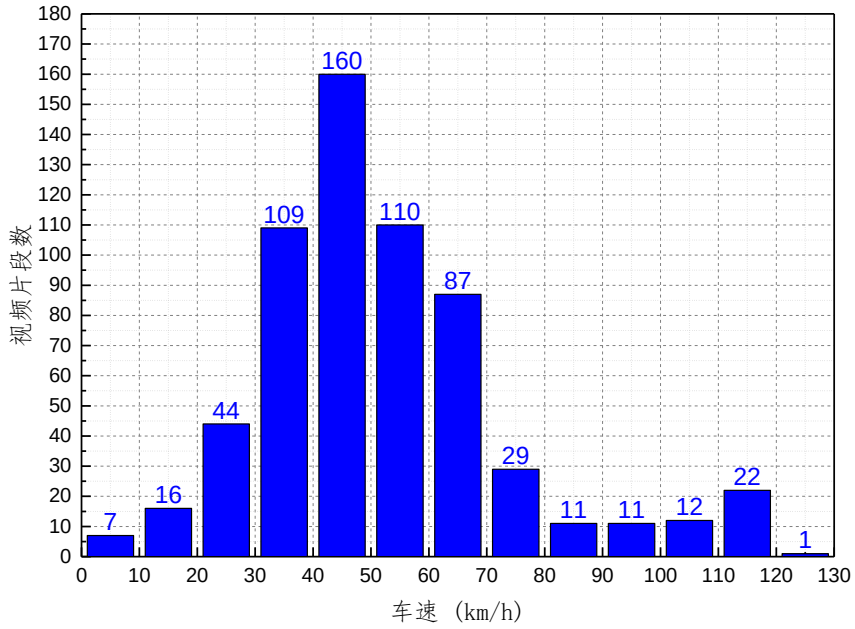

图 2.4 数据集视频片段对应当前车速统计直方图

车载安全预警系统主要需要识别禁令标志（如限速标志、禁行标志等）与警告标志（如注意行人、事故易发路段等），指示标志与指路标志基本不能提供安全预警信息，因此数据集中的交通标志根据本文研究需求仅包含禁令标志及警告标志，其中禁令标志视频 271 段，警告标志视频 392 段，部分视频片段中同时包含这两种交通标志。

考虑到本文研究重点是基于颜色分割的交通标志定位算法，夜间或某些恶劣天气光照情况下由于摄像头性能限制基于颜色的方法会基本失效，根据宝马公司技术报告，在强逆光情况下就算是宝马 7 系这样成熟的商用产品也是无法正常工作的[5]。数据集中有部分视频片段反应的就是这类极端情况，如夜间弱光照（图 2.5）、强逆光（图 2.6）、大雾（图 2.7）、遮挡（图 2.8）等，这些情况下由于摄像头光圈及曝光时间的限制，得到的视频图像中交通标志的颜色会出现偏差或几何形状会不完整，本文研究的算法在这些情况下表现不佳，具体见本章最后“2.5 算法在数据集上测试验证”一节中的分析。虽然本文的研究没能很好的处理这部分极端光照天气情况下交通标志的检测定位问题，然而考虑到数据集的完整性并没有剔除这部分视频片段。

图 2.5 夜间弱光照情况示例视频截图

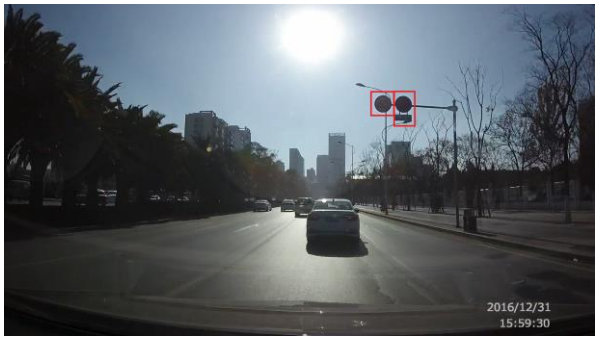

图 2.6 强逆光环境示例视频截图

图 2.7 大雾天气示例视频截图

图 2.8 存在遮挡情况示例视频截图

2.2 交通标志定位中红色及黄色分割方法比较研究

颜色分割的目的在于从原始图像中提取出特定颜色区域，由于交通标志有其固定颜色，使用颜色分割与单纯使用几何形状分割相比可以更好的利用此先验知识。交通标志中与车载安全预警系统联系较为紧密的是禁令标志与警告标志，这类标志在中国交通标志中特征颜色为红色及黄色，故本节将对红色与黄色的分割方法进行研究分析。

2.2.1 颜色成像原理简述

颜色从本质上说就是特定波长的电磁波，之所以通常将红、绿、蓝称为光的三原色（或三基色）与人眼的感光原理有关，人眼对颜色的感知由视网膜上的三种视锥细胞（Cone Cell）实现，一般将其称为 S、M、L 视锥细胞，分别对蓝、绿、红光敏感，归一化的光谱响应曲线见图 2.9。彩色相机就是根据人眼视锥细胞的这一特性设计制造的，目前主流的 CCD 或 COMS 传感器大部分使用的都是红、绿、蓝三种颜色的滤光片，其组合结构有两种基本形式——传统的马赛克结构滤色器阵列[77]（Color Filter Array, CFA）及日本适马公司发明的更为先进的 Foveon X3 传感器[78]，它们的滤光片基本组合结构对

比见图 2.11。不过无论哪种滤光片结构最终得到的都是红、绿、蓝三色分量的大小，与人眼类似，COMS 或 CCD 传感器也有其自己的光谱响应曲线（Spectral Response），如图 2.10 就是佳能 T3i APS-C CMOS 传感器的光谱响应曲线[79]。不同型号传感器的光谱响应特性有很大差异，不过通过传感器内置 ISP（Image Signal Processor，图形信号处理器）处理后得到的图像差异就不是很大了。

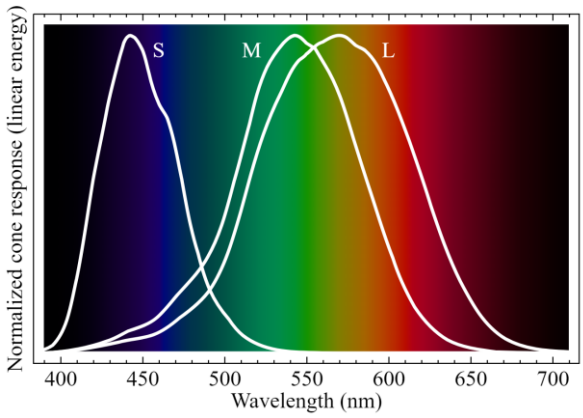

图 2.9 归一化后的人眼视锥细胞光谱响应曲线

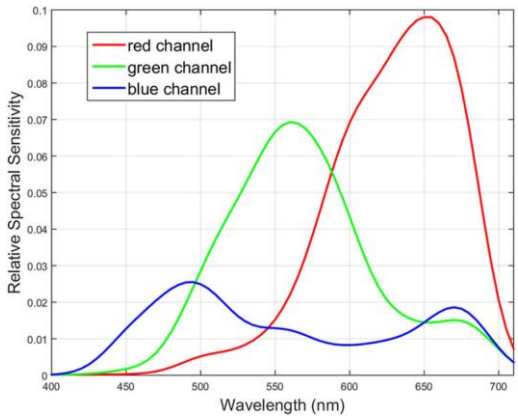

图 2.10 佳能 T3i APS-C CMOS 传感器光谱响应曲线

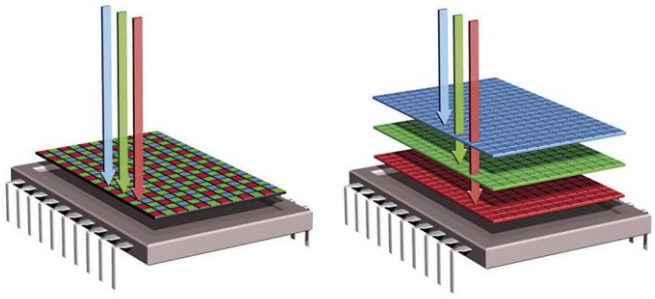

图 2.11 CFA 与 Foveon X3 结构对比图

2.2.2 基于 RGB 空间的颜色分割方法

2.2.2.1 基本 RGB 颜色空间

从以上原理性的分析中可以看到，对于颜色成像来说红、绿、蓝三色是最原始的分量，故可将其视为一组最基本的基，这样得到的颜色空间就是 RGB 颜色空间（RGB Color Space），一般使用正交直角坐标系的形式表示此空间，如图 2.12 所示，每一种颜色对应坐标系中的一个坐标点，此图通常称为 RGB 立方体（RGB Cube）。由于实际 COMS 或 CCD 传感器的采样均是通过 ADC（Analog to Digital Convert，模数转换器）实现，而

ADC 的精度是有限的，故得到的 RGB 值并不是连续值而是离散的，当前主流 COMS 传感器的 ADC 精度为 8-bit 或 10-bit，不过并不是每个像素点的 RGB 分量均有这么高精度，对于使用 CFA 结构颜色滤波器的传感器来说一个像素点只有一种颜色（参考图 2.11），其余两种缺失的颜色值是由 ISP 根据插值算法计算得到的。不过只从传感器的输出结果看可以简单的认为对于每个像素点而言，其 RGB 分量取值是一个离散的范围，绝大部分应用及文献中此范围均是 0~255，故本文也采用此惯例，用R、G、B分别代表红色、绿色、蓝色分量的值，满足$R,G,B \in [0,255]$且$R,G,B \in \mathbb{Z}$。

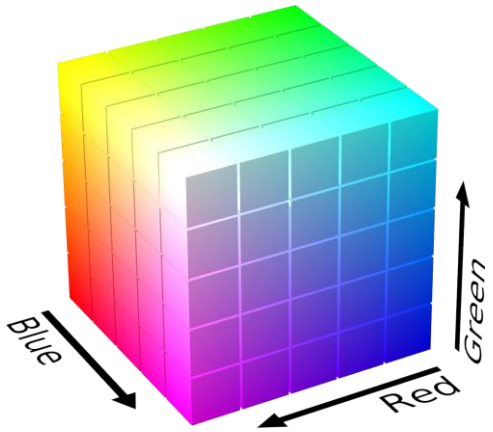

图 2.12 RGB 立方体

2.2.2.2 RGB 阈值分割方法

在 RGB 空间中直接进行颜色分割一般使用的都是固定阈值法，其中最为通用的一种策略中红色判据为：

$$R - B \geq Th_{R1} \text{ and } R - G \geq Th_{R1} \tag{2.1}$$

黄色判据为：

$$R - B \geq Th_{Y1} \text{ and } G - B \geq Th_{Y1} \tag{2.2}$$

根据文献中的普遍观点[80]，基于大量实际经验数据确定的阈值为$Th_{R1} = 20$，$Th_{Y1} = 30$。然而在本数据集上测试表明此方法及阈值并不合理，以红色为例仔细对比式(2.1)与式(2.2)，显然存在$R - G \geq 20$及$G - B \geq 30$同时成立的情况，此时就会把一些黄色误判为红色，这一点从图 2.13 的结果中可以很明显的看到，图中上方为原始图片下方为分割后的结果。事实上在数据集中测试发现此方法就算调整阈值Th_{R1}的值也很难准确区分出红色与黄色，这可以认为是此方法固有的局限性。

图 2.13 RGB 阈值法 1 分割红色结果图

除上述公式外 Gómez-Moreno 等人使用了另一套分割公式，其原始论文中使用的是归一化后的 R、G、B 值[44]，为统一使用本文定义的R、G、B需要对此进行一些变换，最终得到的红色判据为：

$$\frac{R}{R+G+B} \geq Th_{R2} \text{ and } \frac{G}{R+G+B} \leq Th_{G2} \tag{2.3}$$

黄色判据为：

$$\frac{R+G}{R+G+B} \geq Th_{Y2} \tag{2.4}$$

其中阈值$Th_{R2}=0.4$，$Th_{G2}=0.35$，$Th_{Y2}=0.85$，这也是通过实验确定的经验值。在数据集上测试表明，此方法对于红色的提取效果比式(2.1)与式(2.2)给出的方法要好，然而式(2.4)对于黄色的分割效果则不尽人意，图 2.14 所示的结果说明了这一点，此处使用的是式(2.4)进行黄色分割，然而实际上红色区域也出现在了分割结果中。考虑式(2.4)，当R值很大而G值很小时此式也会成立，不过此时对应的颜色显然是红色而不是黄色，这就是造成图 2.14 中错误结果的原因。

图 2.14 RGB 阈值法 2 分割黄色结果图

虽然可以针对这些缺陷进一步改进 RGB 阈值法中使用的线性分类器形式，然而已有研究均表明颜色在 RGB 空间内是很难实现线性可分的，故 RGB 空间内的线性分类器有其固有的局限性，为解决此问题研究人员将目光投向了其它颜色空间。

2.2.3 基于 HSI 空间的颜色分割方法

2.2.3.1 HSI 颜色空间

RGB 颜色空间中，光照亮度的变化会同时导致R、G、B值的大幅变化，且上一节研究中已经展现出 R、G、B 分量间存在的紧密耦合关系，所以很难直接根据一个颜色坐标向量对其所属颜色进行准确分类，故研究人员从色调、饱和度、亮度这一直观的角度出发提出了一类圆柱（或圆锥）坐标系下的颜色空间。这类颜色空间不止一个，较为常用的有 HSV（Hue、Saturation、Value, 色调、饱和度、明度）、HSB（Hue、Saturation、Brightness, 色调、饱和度、明度）、HSI（Hue、Saturation、Intensity, 色调、饱和度、亮度）、HSL（Hue、Saturation、Lightness, 色调、饱和度、亮度）、HCI（Hue、Chroma、Intensity, 色调、浓度、亮度）等，一般认为 HSV 与 HSB 类似，HSI 与 HSL 类似。这类颜色空间可用圆柱坐标系表示出来，如图 2.15 及图 2.16 所示的 HSL 及 HSV 圆柱体，

在某些文献中认为使用如图 2.17 所示的双圆锥坐标系表示 HSL/HSI 空间会更为准确。

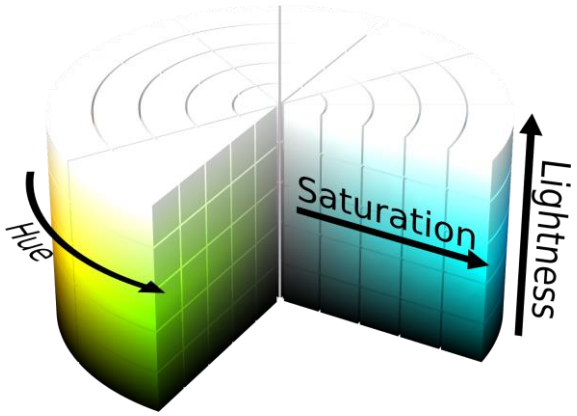

图 2.15 HSL 圆柱体

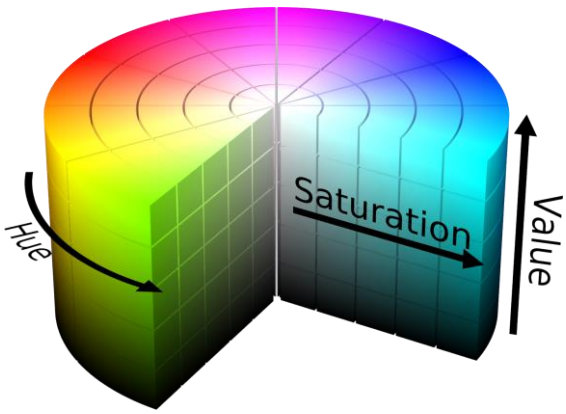

图 2.16 HSV 圆柱体

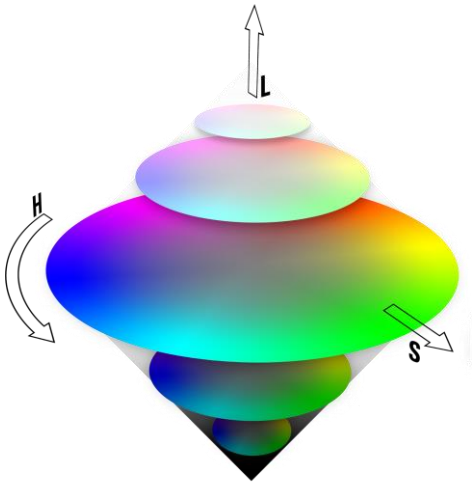

图 2.17 HSI 空间双圆锥坐标系表示

这类颜色空间中，使用色调（又称色相）H 分量表示颜色的基本属性，这一般是一个角度值，如红绿蓝三色对应的角度分别为 0°、120°、240°，下文在不引起混乱时为方便起见一般略去角度单位，直接使用一个数字表示 H 分量；使用饱和度 S 分量表示颜色的纯度，其值越大表示对应的颜色越接近纯净广谱的颜色，反应在视觉上就是越浓越艳；而最后一个分量则用于表征亮度。色调分量的含义在不同颜色空间中基本一致，不过饱和度与亮度的含义在不同颜色空间中并不相同，对应的与 RGB 空间之间的转换公式也不相同；除此之外就算是同一个颜色空间，不同文献中给出的转换公式也并不完全相同。面对这种不统一的研究现状，本文从实用的角度出发选取了在交通标志识别中最为常用的 HSI 空间进行讨论研究，HSI 空间与 RGB 空间最通用的转换公式为[81]：

$$\begin{cases} H = \begin{cases} \theta & ,B \le G \\ 360° - \theta & ,B > G \end{cases} \\ S = 1 - \dfrac{\min\{R,G,B\}}{I} \\ I = \dfrac{R+G+B}{3} \end{cases} \tag{2.5}$$

其中

$$\theta = \arccos\left\{ \frac{0.5 \cdot [(R-G)+(R-B)]}{[(R-G)^2 + (R-B)(G-B)]^{0.5}} \right\} \tag{2.6}$$

此处假设arccos反余弦函数返回的是角度值而不是弧度值，否则需要进行换算。根据上文对R、G、B范围的定义，即$R,G,B \in [0,255]$，易知H、S、I的值域范围为$H \in [0°, 360°)$、$S \in [0,1]$、$I \in [0,255]$。特别地，若$S = 0$，H无意义，这些无定义点被称为奇异点，在奇异点附近R、G、B值微小的变化会导致H、S、I值明显的变化，这会导致系统的稳定性问题，所以这一特性是 HSI 空间的一个缺陷。不过由下文可知在颜色分类中都存在$S > S_{min}$的条件判断，故此缺陷从实际使用的角度上看没有明显影响。

HSI 颜色空间最大的优点在于其三个分量间耦合关系较小，与 RGB 空间相比更符合人类对颜色的认知，且用于表征颜色特性的色调 H 分量受光照的影响相对较少，很适合用于进行颜色分类。其实颜色分割问题从本质上说就是一个模式分类问题，上一节中已经指出，在原始的 RGB 空间中由于 R、G、B 三个分量间的耦合关系导致颜色分类问题其实是很难用线性分类器来解决的，在模式分类的一般理论中解决此类问题一个通用的思路就是进行特征空间变换，将原始难以分割的空间通过某种变换转换为一个容易分割的空间。经典的 SVM 算法使用的就是此思路，通过所谓的核技巧（Kernel Trick）将低维空间中不可分的向量映射到高维空间中使其变成可用超平面进行分割的，这一过程可参考图 2.18 所示的示意图。事实上 HSI 颜色空间变换实现的也是这样一个过程，将原始 RGB 空间中不好分类的颜色向量通过式(2.5)及式(2.6)的非线性函数映射到 HSI 空间中，这样就可以实现各分量间的解耦以便于使用线性分类器颜色进行分类，具体分类算法将在下文中进行介绍。由于 Cover 定理指出复杂模式分类问题非线性映射到高维空间中更有可能是线性可分的，故 SVM 等通用模式分类方法都倾向于提升原始数据维度，而 HSI 空间变换并没有提升数据维度，而是针对颜色分类问题的具体需求引入了式(2.6)中的反余弦函数，通过这个巧妙的变换使得 HSI 空间具有比 RGB 空间更好的线性可分性。

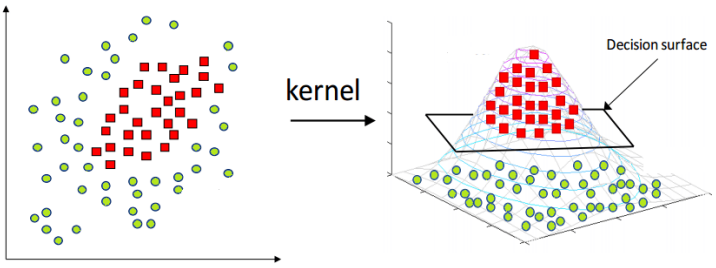

图 2.18 SVM 中核函数作用示意图

然而也正是由于式(2.6)中反余弦函数的存在导致 HSI 变换运算量较大，在低性能嵌入式车载安全预警系统（如行车记录仪）中应用受限，为降低运算量 Been-Chian Chien 等人提出了使用分段函数替代非线性反余弦函数的变换方法[82]：

$$\begin{cases} H = \begin{cases} 60° \cdot \left(\dfrac{G-B}{C_{max}-C_{min}}\right) & ,R = C_{max} \\ 60° \cdot \left(2 + \dfrac{B-R}{C_{max}-C_{min}}\right) & ,G = C_{max} \\ 60° \cdot \left(4 + \dfrac{R-G}{C_{max}-C_{min}}\right) & ,B = C_{max} \\ \text{undefine} & ,C_{min} = C_{max} \end{cases} \\ S = \begin{cases} 0 & ,C_{min} = C_{max} \\ \dfrac{C_{max}-C_{min}}{C_{max}+C_{min}} \cdot I' & ,I \le 127.5 \\ \dfrac{C_{max}-C_{min}}{2 \cdot 255 - C_{max} - C_{min}} \cdot I' & ,\text{otherwise} \end{cases} \\ I = \dfrac{C_{min}+C_{max}}{2} \end{cases} \tag{2.7}$$

其中$C_{min} = \min\{R,G,B\}, C_{max} = \max\{R,G,B\}, I' = 1 - \frac{|I-127.5|}{127.5}$；若$H < 0°$则$H = H + 360°$。式(2.7)与文献[82]中给出的形式稍有区别，主要是为了保证H、S、I的值域与式(2.5)中相同，即$H \in [0°, 360°)$、$S \in [0,1]$、$I \in [0,255]$。事实上著名的计算机视觉库 OpenCV 中 RGB 空间至 HSL 空间（OpenCV 未提供 HSI 空间）的转换函数使用的算法就是式(2.7)，只是在此基础上做了进一步简化，去掉了I'项，即认为I'恒为 1[83]。

由于没有反余弦函数式(2.7)的运算量从直觉上看是要小于式(2.5)的，为定量衡量二者计算效率及计算结果的差异性，对 RGB 空间内所有 16777216(256^3)个颜色向量分别根据式(2.5)及式(2.7)进行颜色空间转换，耗时情况见表 2.3。由于实际计算机程序中对较短时间计时的误差很大，为保证计时的有效性此处得到的时间是重复执行 100 次转换运

算后的结果，表中最后一列单像素点耗时是在考虑了重复执行情况下计算得到的，可以视为算法单次执行平均耗时。表 2.3 中 Debug 模式和 Release 模式分别对应”-O0”及”-O3”优化级别，其余一些编译参数设置也有区别，关于测试平台、编译器、编译参数等将在本文第四章“仿真验证平台及嵌入式原型的搭建与系统集成测试”中统一具体介绍。从表 2.3 中可以看到式(2.7)给出的转换公式的确在计算速度上远快于式(2.5)给出的公式，特别是在实际发布时使用的 Release 模式下式(2.7)所需时间仅是式(2.5)的 7%。

表 2.3 两种 HSI 颜色空间转换方法耗时比较

	Debug 模式	Release 模式	Release 单像素点
式(2.5)耗时	134092 ms	29869 ms	17.8 ns
式(2.7)耗时	79401 ms	2105 ms	1.3 ns
耗时比	59.2%	7.0%	-

表 2.3 中的结果说明了式(2.7)在计算效率上的巨大优势，为进一步比较这两种方法计算结果的差异性，对其输出结果的差值（定义为式(2.7)的结果减去式(2.5)的结果）进行统计分析，结果见图 2.19~图 2.21，为方便直观的比较图中横坐标换算成了相对偏差的形式。

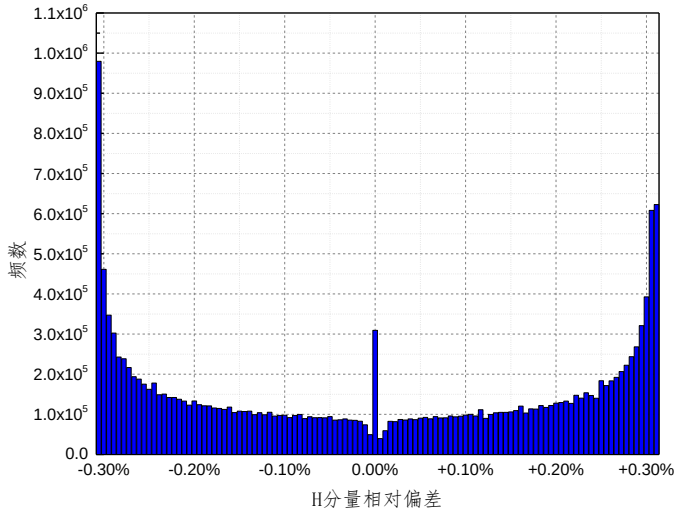

图 2.19 两种 HSI 空间变换方法 H 分量相对偏差对比图

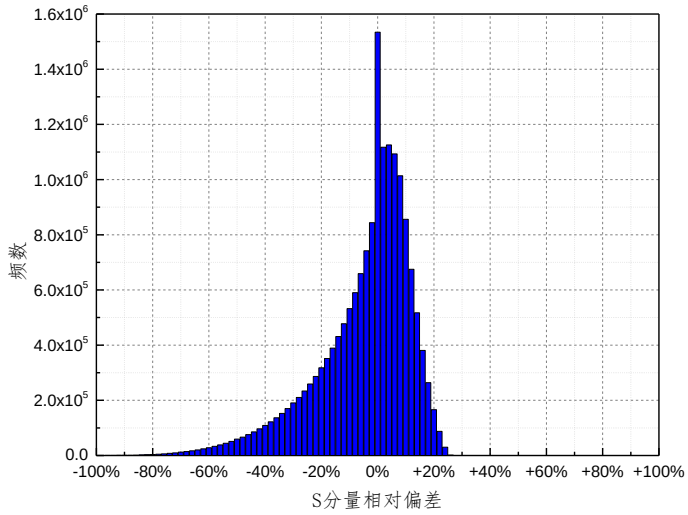

图 2.20 两种 HSI 空间变换方法 S 分量相对偏差对比图

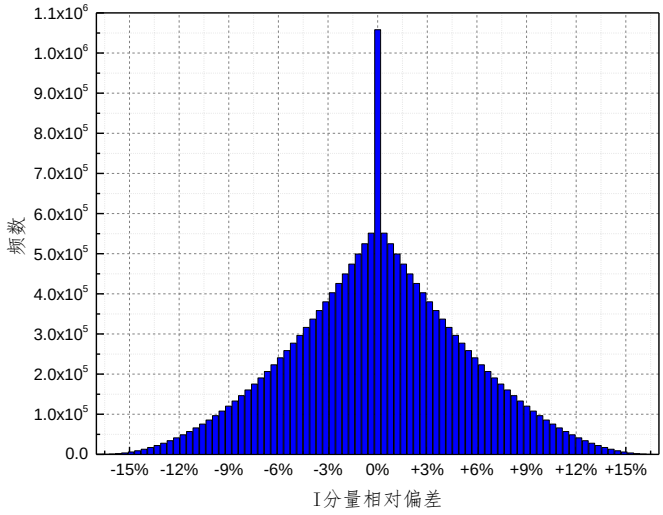

图 2.21 两种 HSI 空间变换方法 I 分量相对偏差对比图

从图 2.19 中可以看到式(2.5)与式(2.7)得到的 H 分量相差很小，全空间偏差均在±1°(±0.3%)的范围内，对于颜色分类来说基本没有差别；而根据图 2.20 及图 2.21，S 及 I 分量偏差相对较大，不过这可以理解为不同方法对亮度及饱和度的映射逻辑不同导致的，对于颜色分割来说 S 与 I 分量仅用于去掉亮度及饱和度太低的颜色，此处的较大的相对偏差并不会对算法性能造成明显影响，因此下文将使用式(2.7)作为 HSI 变换公式介绍基于 HSI 空间的颜色分割方法。

除式(2.5)与式(2.7)外还有其它一些 HSI 空间转换公式，不过这些方法在计算量上不比式(2.7)有明显优势，本文就不再介绍了，读者若有兴趣可进一步参考文献[84]。

2.2.3.2 HSI 阈值分割方法

与 RGB 空间类似在 HSI 空间中颜色分类一般使用的也是固定阈值法，根据色调 H 分量的值即可确定其所属颜色，HSI 空间的优点就是 H 分量的值随光照变化的影响较小，故此方法鲁棒性相对较好。只根据 H 分量判断存在一些问题，当整体亮度较高或较低时对应的颜色会趋向于白色或黑色；当饱和度较低时对应的颜色趋向于灰色；故一般要同时加上 S 与 I 分量的约束条件才能较为准确的判断颜色种类。结合目前已有研究论文中普遍使用的方法与实际仿真测试的经验，本文使用以下方法判断红色与黄色：

红色判据：

$$\begin{cases} H \geq H_{R(min)} \text{ or } H \leq H_{R(max)} \\ S \geq S_{min} \\ I_{min} \leq I \leq I_{max} \end{cases} \tag{2.8}$$

黄色判据：

$$\begin{cases} H_{Y(min)} \leq H \leq H_{Y(max)} \\ S \geq S_{min} \\ I_{min} \leq I \leq I_{max} \end{cases} \tag{2.9}$$

其中式(2.8)与式(2.9)中 S 与 I 分量的阈值使用了相同的S_{min}、I_{min}及I_{max}；由于红色对应色调 H 分量的值为 0，且$H \in [0,360)$，$H_{R(min)}$事实上是大于$H_{R(max)}$的，故式(2.8)中将H的判断条件分开写成两个条件。Maldonado-Bascón 等人对西班牙交通标志特征颜色（红、黄、蓝）在 HSI 空间中 H 与 S 分量分布情况进行了详尽的统计分析[85]，结合其研究成果及其它已有文献资料并在本文数据集上进行测试后可确定各阈值取值，结果见表 2.4，此时对红色及黄色的分割结果见图 2.22，这两张图中对红色及黄色的分割结果较为准确干净。然而进一步测试发现此方法对黄色的分割结果存在缺陷，交通标志的黄色与冬季植物的绿色很难区分开来，一个典型的视频片段分割结果见图 2.23。

表 2.4 HSI 空间颜色分割阈值取值表

$H_{R(min)}$	$H_{R(max)}$	$H_{Y(min)}$	$H_{Y(max)}$	S_{min}	I_{min}	I_{max}
320	20	35	60	0.24	20	210

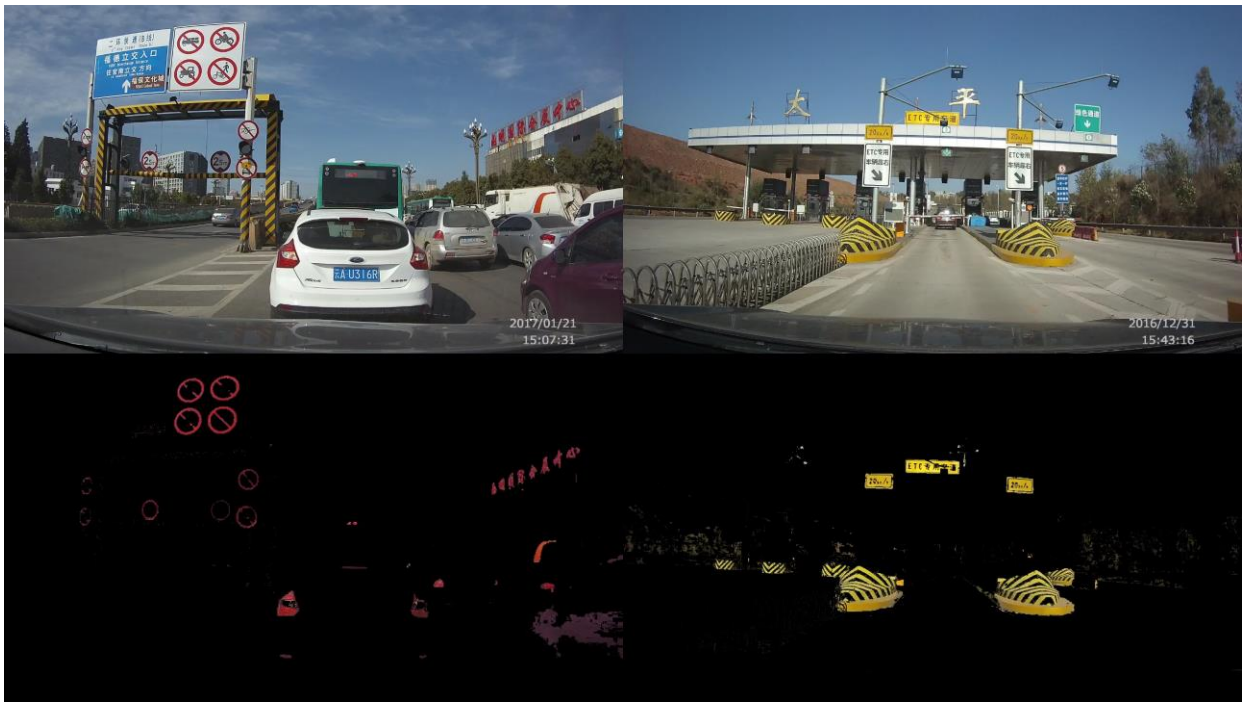

图 2.22 HSI 阈值法分割红色及黄色结果图

图 2.23 HSI 阈值法在绿色背景较多时分割黄色结果图

2.2.4 SVF 分割方法

上文介绍的 RGB 及 HSI 空间的颜色分割方法均用于对某种具体颜色（红色、黄色、蓝色等）进行分割，然而即使使用效率较高的式(2.7)所示方法进行 HSI 颜色空间转换其计算量依然较大，因此很多研究人员也在尝试先用某种计算量小的算法快速过滤背景区域从中提取出感兴趣的区域后再进行进一步处理。在交通标志识别中，此类算法的一个典型代表就是由 Asakura 等人提出的 SVF（Simple Vector Filter, 简单向量滤波器）算法[43]。SVF 的核心思想基于以下事实：颜色可以粗略的分为彩色（Chromatic）与单色（Achromatic），其中彩色指那些较为鲜艳的颜色，而单色指诸如阴影等较接近灰色的颜色；一般而言交通标志等我们感兴趣的区域均为彩色区域，故可以先从图像中过滤掉单色区域以此实现目标区域与背景的快速分离。

SVF 中将每一种颜色视为一个向量（Vector），类似 HSI 颜色空间，每一类相似的颜色具有相同的方向，对于单色而言其方向接近三角形重心方向（参考图 2.24），通过简单的计算颜色向量的方向即可快速去除单色区域,这也就是其名称“简单向量滤波器”的来源。

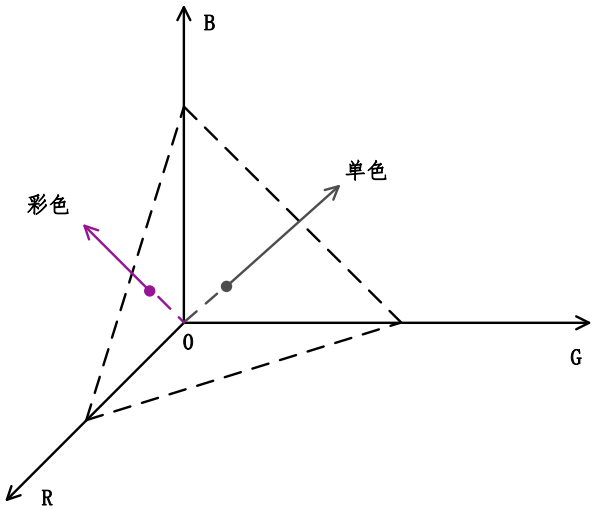

图 2.24 SVF 基本原理示意图

为计算颜色向量方向 SVF 中使用了如下简单公式：

$$result_{SVF} = \frac{(|R-G| + |R-B| + |B-G|)}{3D} \tag{2.10}$$

当$result_{SVF} < 1$时对应颜色为单色，当$result_{SVF} \geq 1$时对应颜色为彩色；式中D为单

色提取系数，一般而言D越大单色区域越多彩色区域越小，Asakura 等人的原始论文中将D固定为 20，中南大学的常卢峰对 SVF 算法进行了进一步研究[80]，其论文中提出晴天时D取$[22,25]$，阴天时D取$[13,16]$。本文使用 SVF 是为了快速去除单色区域减小计算量，故D设定为 20 以保留更多的彩色区域。

事实上式(2.10)可以进一步化简，与上文一样使用C_{min}、C_{max}分别表示R、G、B中的最大值及最小值，即$C_{min}=\min\{R,G,B\}$，$C_{max}=\max\{R,G,B\}$，代入式(2.10)可得：

$$result_{SVF}=\frac{C_{max}-C_{min}}{D'} \tag{2.11}$$

其中$D'\triangleq 1.5\cdot D$。

从式(2.11)中可以看到，只要计算出C_{min}及C_{max}，其差值大于等于D'即为彩色，小于D'即为单色。此方法计算量极小，$D'=30$时分割效果见图 2.25，从中可以看到，路面、乌云、阴影等单色区域被过滤掉了，而交通标志、信号灯、车牌等较为鲜艳的彩色特征物被很好的保留了下来，特别是交通标志及车牌的边界提取效果极好。不过也要看到，SVF 对同样鲜艳的天空、楼房等物体是无能为力的，故需要配合上文所述的 RGB 及 HSI 空间颜色分割方法进行进一步处理。

图 2.25 SVF 分割彩色区域结果图

2.3 混合颜色分割策略

上一节中对3种颜色分割方法进行了讨论研究，然而可以看到每种方法都各有优劣，一个自然的想法就是将这些方法结合起来使用以达到更优的检测效果及更高的效率，本节就按照此思路对混合颜色分割策略进行研究，取以上3种方法的交集作为颜色判据以此取得更好的分类准确性。

分析上文提出的若干颜色空间转换方法及颜色分割判据（式(2.1)~式(2.11)）可以发现其中涉及的变量只有3个——R、G、B，且分类依据均可写成分段线性不等式形式，这意味着我们可以用统一的形式去描述这几种颜色分割方法，以便在此基础上对算法进行进一步精简优化。

为方便下文讨论，此处先引入一个最初用于偏序集合（Partially ordered set, poset）理论中的符号$\geq_e$用于比较两个大小相同的矩阵或向量[86]，其定义为：对于任意矩阵$\boldsymbol{A}=[a_{ij}],\boldsymbol{B}=[b_{ij}]\in\mathbb{R}^{n\times m}$，若对任意$1\leq i\leq n, 1\leq j\leq m$均有$a_{ij}\geq b_{ij}$，则我们将其记为$\boldsymbol{A}\geq_e\boldsymbol{B}$。

2.3.1 红色混合分割策略

重新整理式(2.1)及(2.3)给出的RGB空间红色阈值分割判据并将其写成矩阵形式有

$$\begin{bmatrix}1 & 0 & -1\\ 1 & -1 & 0\end{bmatrix}\begin{bmatrix}R\\G\\B\end{bmatrix}\geq_e\begin{bmatrix}Th_{R1}\\Th_{R1}\end{bmatrix} \tag{2.12}$$

及

$$\begin{bmatrix}1-Th_{R2} & -Th_{R2} & -Th_{R2}\\ Th_{G2} & Th_{G2}-1 & Th_{G2}\end{bmatrix}\begin{bmatrix}R\\G\\B\end{bmatrix}\geq_e\begin{bmatrix}0\\0\end{bmatrix} \tag{2.13}$$

式(2.7)给出的HSI空间转换公式及式(2.11)给出的SVF方法均需要计算C_{min}及C_{max}，故可分为6种情况分别讨论，将式(2.7)的转换公式与式(2.8)的判断依据结合到一个式子中。由于当$C_{min}=C_{max}$时式(2.11)的结果为0，SVF算法会过滤掉此单色点，故下文在处理式(2.7)HSI空间转换公式时不考虑此情况。

(1) $R\geq G\geq B$

通过遍历整个颜色空间可计算得到此时$H\in[0,60]$，HSI空间内红色判据可写为：

$$\begin{bmatrix} -1 & 0 & -1 \\ H_{R(max)} & -60 & 60-H_{R(max)} \\ 1-S_{min} & 0 & -1-S_{min} \\ 1 & 0 & 1 \\ -1 & 0 & -1 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \geq_e \begin{bmatrix} -255 \\ 0 \\ 0 \\ 2\cdot I_{min} \\ -2\cdot I_{max} \end{bmatrix} \tag{2.14}$$

或

$$\begin{bmatrix} 1 & 0 & 1 \\ H_{R(max)} & -60 & 60-H_{R(max)} \\ S_{min}+1 & 0 & S_{min}-1 \\ 1 & 0 & 1 \\ -1 & 0 & -1 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \geq_e \begin{bmatrix} 255 \\ 0 \\ 510\cdot S_{min} \\ 2\cdot I_{min} \\ -2\cdot I_{max} \end{bmatrix} \tag{2.15}$$

SVF 方法：

$$\begin{bmatrix} 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \geq_e [D'] \tag{2.16}$$

(2) $R \geq B \geq G$

同理可计算得到此时$H \in [300,360)$，HSI 空间内红色判据可写为：

$$\begin{bmatrix} -1 & -1 & 0 \\ -H'_{R(min)} & H'_{R(min)}+60 & -60 \\ 1-S_{min} & -1-S_{min} & 0 \\ 1 & 1 & 0 \\ -1 & -1 & 0 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \geq_e \begin{bmatrix} -255 \\ 0 \\ 0 \\ 2\cdot I_{min} \\ -2\cdot I_{max} \end{bmatrix} \tag{2.17}$$

或

$$\begin{bmatrix} 1 & 1 & 0 \\ -H'_{R(min)} & H'_{R(min)}+60 & -60 \\ S_{min}+1 & S_{min}-1 & 0 \\ 1 & 1 & 0 \\ -1 & -1 & 0 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \geq_e \begin{bmatrix} 255 \\ 0 \\ 510\cdot S_{min} \\ 2\cdot I_{min} \\ -2\cdot I_{max} \end{bmatrix} \tag{2.18}$$

其中$H'_{R(min)} \triangleq H_{R(min)} - 360$。

SVF 方法：

$$\begin{bmatrix} 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \geq_e [D'] \tag{2.19}$$

(3) $G \geq R \geq B$

此时$H \in [60,120]$，考虑到$H_{R(max)}$的取值在 320 左右，$H_{R(min)}$的取值在 20 左右，显然此条件下不存在红色。

(4) $G \geq B \geq R$

此时$H \in [120,180]$，同理此条件下没有红色。

(5) $B \geq R \geq G$

此时$H \in [240,300]$，同理此条件下没有红色。

(6) $B \geq G \geq R$

此时$H \in [180,240]$，同理此条件下没有红色。

考虑各参数实际取值范围以上各式间并不独立，如 RGB 阈值法式(2.12)与 SVF 方法式(2.16)及式(2.19)间存在包含关系，结合各参数取值范围去除其中冗余式子，并对各式顺序进行重排，最终可整理得到完整的红色混合分割策略流程图，见图 2.26；化简过程中引入了$D' \geq Th_{R1}$的约束条件。此策略的处理效果见图 2.27，为方便对比同时给出了其他几种方法的处理结果，各参数设置与上文相同。

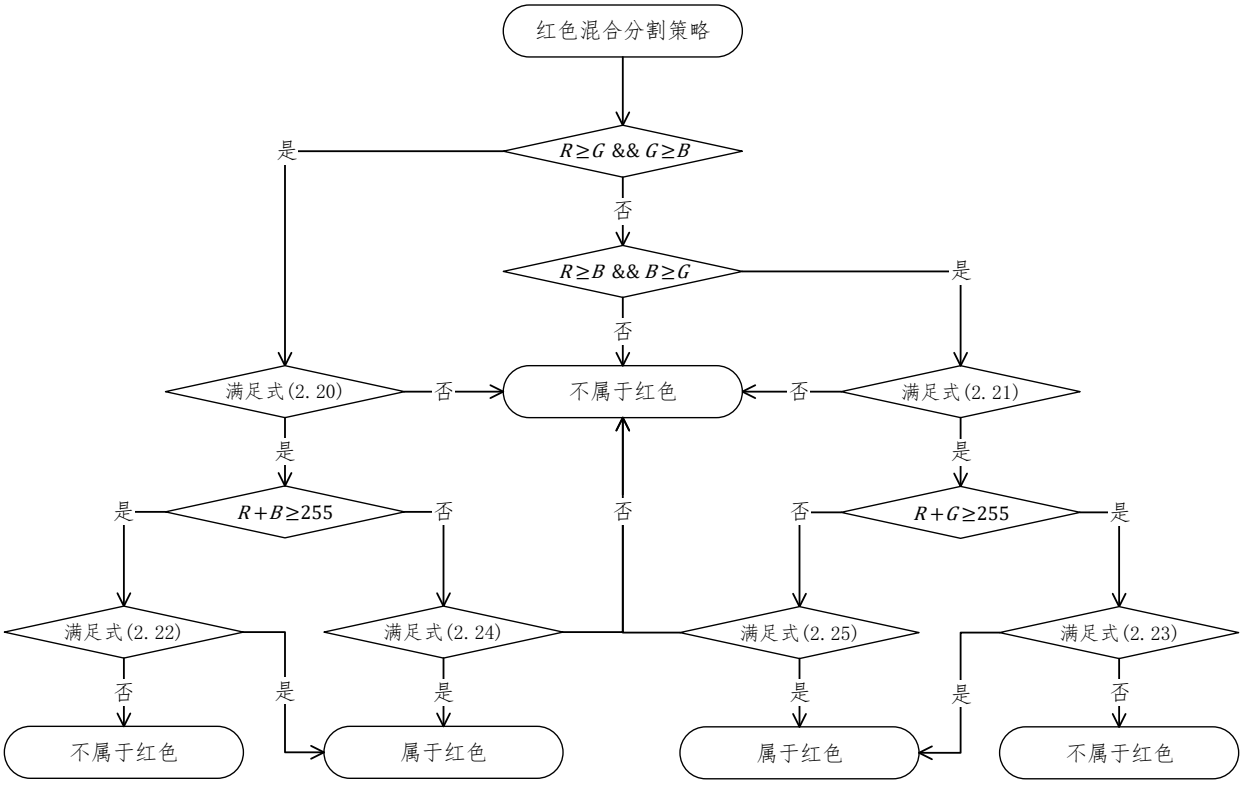

图 2.26 红色混合分割策略流程图

图 2.26 中涉及的各式如下：

$$\begin{bmatrix} 1 & -1 & 0 \\ 1 & 0 & -1 \\ H_{R(max)} & -60 & 60 - H_{R(max)} \\ 1 - Th_{R2} & -Th_{R2} & -Th_{R2} \\ Th_{G2} & Th_{G2} - 1 & Th_{G2} \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \geq_e \begin{bmatrix} Th_{R1} \\ D' \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.20}$$

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & -1 & 0 \\ -H'_{R(min)} & H'_{R(min)} + 60 & -60 \\ 1 - Th_{R2} & -Th_{R2} & -Th_{R2} \\ Th_{G2} & Th_{G2} - 1 & Th_{G2} \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \geq_e \begin{bmatrix} Th_{R1} \\ D' \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.21}$$

$$\begin{bmatrix} S_{min} + 1 & 0 & S_{min} - 1 \\ -1 & 0 & -1 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \geq_e \begin{bmatrix} 510 \cdot S_{min} \\ -2 \cdot I_{max} \end{bmatrix} \tag{2.22}$$

$$\begin{bmatrix} S_{min} + 1 & S_{min} - 1 & 0 \\ -1 & -1 & 0 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \geq_e \begin{bmatrix} 510 \cdot S_{min} \\ -2 \cdot I_{max} \end{bmatrix} \tag{2.23}$$

$$\begin{bmatrix} 1 - S_{min} & 0 & -(1 + S_{min}) \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \geq_e \begin{bmatrix} 0 \\ 2 \cdot I_{min} \end{bmatrix} \tag{2.24}$$

$$\begin{bmatrix} 1 - S_{min} & -(1 + S_{min}) & 0 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \geq_e \begin{bmatrix} 0 \\ 2 \cdot I_{min} \end{bmatrix} \tag{2.25}$$

从图 2.27 中可以看到，本节提出的红色混合分割策略与上文介绍的几种已有方法相比分割效果要更好，其分割准确度大幅优于 RGB 阈值法，且与 HSI 空间分割方法相比得到的结果也要更干净一些；比如图中 HSI 分割结果中几个交通标志下方有一些阴影区域也被算在了红色范畴内，而混合颜色分割策略则很好的去除了这些区域。

图 2.27 红色混合分割策略与其它方法效果对比图

2.3.2 黄色混合分割策略

黄色混合分割策略的推导过程与红色混合分割策略类似，限于篇幅此处从略，直接给出最终的算法流程图，见图 2.28，由于$H_{Y(min)}$及$H_{Y(max)}$分别为 35 及 60，只有$R \geq G \geq B$的时候满足此条件，因此黄色混合分割策略比红色混合分割策略要简单一些。此策略的处理效果见图 2.29，与上一节一样此处同时给出了其它几种方法的结果对比。

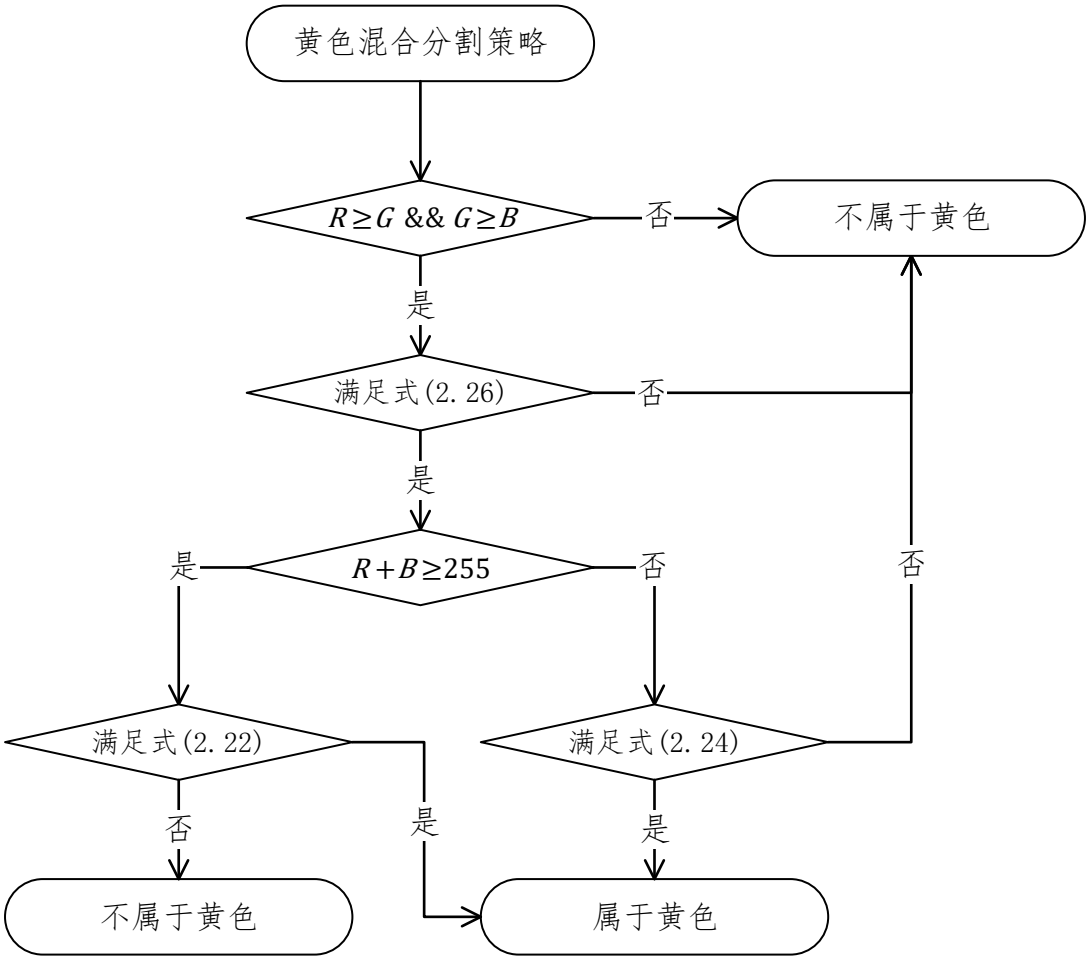

图 2.28 黄色混合分割策略流程图

图 2.28 中的式(2.22)与式(2.24)与红色混合分割策略中用到的式子是一样的，为方便阅读一并列于此处：

$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ H_{Y(max)} & -60 & 60-H_{Y(max)} \\ -H_{Y(min)} & 60 & H_{Y(min)}-60 \\ 1-Th_{Y2} & 1-Th_{Y2} & -Th_{Y2} \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \geq_e \begin{bmatrix} Th_{Y1} \\ Th_{Y1} \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.26}$$

$$\begin{bmatrix} S_{min}+1 & 0 & S_{min}-1 \\ -1 & 0 & -1 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \geq_e \begin{bmatrix} 510 \cdot S_{min} \\ -2 \cdot I_{max} \end{bmatrix} \tag{2.22}$$

$$\begin{bmatrix} 1-S_{min} & 0 & -(1+S_{min}) \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \geq_e \begin{bmatrix} 0 \\ 2 \cdot I_{min} \end{bmatrix} \tag{2.24}$$

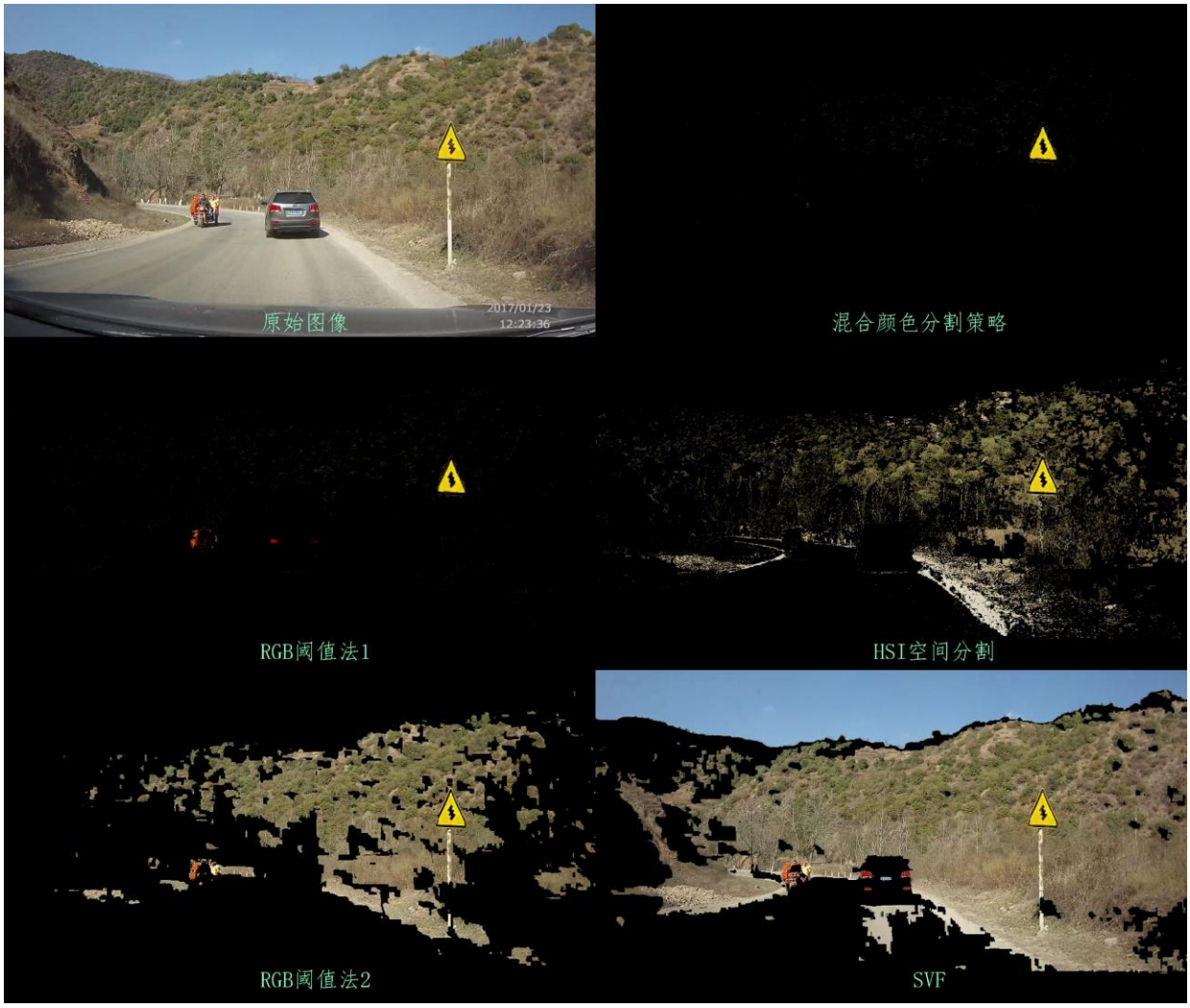

图 2.29 黄色混合分割策略与其它方法效果对比图

图 2.29 使用的原始图像与图 2.23 取自数据集中同一段视频，HSI 方法对此图的分割结果不理想，背景中大范围的绿色与土黄色没有被去除掉，这对后续形状定位很不利；而 RGB 阈值法对黄色与红色的区分度不够；从图 2.29 中可以看到混合颜色分割策略很好的融合了几种算法的优点，对黄色交通标志的提取极为准确干净。

2.3.3 算法执行时间对比

以上测试结果表明混合颜色分割策略在分割准确性上优于目前研究中普遍采用的各种方法，不过除了准确度外，算法执行时间也是评估算法优劣的一个重要指标，特别是对资源有限的小型嵌入式设备来说这一点十分重要。为定量比较以上几种方法的执行速度，与 2.2.2.1 节中的做法一样，对 RGB 空间内所有颜色颜色向量分别执行各算法判断其是否为红色（对于 SVF 来说是彩色），耗时情况见表 2.5，耗时为重复执行 500 次的结果，使用 Release 模式编译测试。

表 2.5 各种颜色分割算法执行时间对比表

分割方法	耗时(ms)	单像素点耗时(ns)	1920*1080 图像耗时(ms)	800*600 图像耗时(ms)
RGB 阈值法 1	2514	0.30	0.62	0.14
RGB 阈值法 2	6317	0.75	1.6	0.36
HSI 阈值法	134618	16	33	7.7
SVF	17206	2.1	4.3	0.98
混合分割策略	18911	2.3	4.7	1.1

从表 2.5 中可以看到，本节提出的混合颜色分割策略在执行时间上慢于简单的 RGB 阈值法，不过大幅快于 HSI 阈值法，所需时间仅为其 14%左右。从图 2.26 的算法流程图中可以看到，其效率较高的原因在于此策略通过比较R、G、B的大小关系快速去除了大量不可能为红色的点，之后再通过高效的线性运算进行后续判断。由此可见混合颜色分割策略虽然从效果及原理上看是融合了其余几种算法得到的，然而通过分析优化算法执行流程取得了与原始的 SVF 算法（即式(2.10)）相似的执行效率。

2.4 基于 Hough 变换的圆形交通标志定位算法

在颜色分割得到的二值图中交通标志表现为特定的几何形状区域，如空心圆环、三角形等，本节将以图 2.30 所示这类红色圆形交通标志为例对其检测定位方法进行研究。这类标志属于我国禁令标志的范畴，包括很多对于车载安全预警来说很有价值的交通标志，如限速标志、限高标志、限重标志、禁行标志等。本节将先介绍针对颜色分割二值图需要采用的形态学处理方法，之后使用 Hough 圆检测算法对圆形交通标志进行检测并在数据集上测试评估算法效果。

图 2.30 红色圆形交通标志示意图

2.4.1 形态学闭运算预处理

形态学一般指数学形态学（Mathematical morphology），而形态学运算包括一类基于拓扑学的图像变换操作，如膨胀、腐蚀、开闭运算、形态学梯度、骨架抽取等，其中最基本的操作是膨胀（Dilate）与腐蚀（Erode）。形态学运算可用于消除噪声、分割图像、寻找极值区域等，本节将使用形态学闭运算去除颜色分割结果中的噪点以构造完整的连通域。由于形态学闭运算是基于膨胀与腐蚀操作的，此处先对膨胀与腐蚀的基本原理做一介绍。图像形态学可分为二值形态学（针对二值图）与灰度形态学（针对灰度图），上文颜色分割的结果是二值图，故本节只涉及二值形态学操作。在二值形态学惯例中，取值为 1 的点对应物体，取值为 0 的点对应背景，形态学操作都是针对值为 1 的物体来说的；由于 OpenCV 中像素点的取值代表亮度，取值为 0 为黑色，所以实际程序中的膨胀腐蚀是针对图像中白色部分而言的。

膨胀与腐蚀分别是求取图像局部最大值及最小值的操作，由于白色像素点亮度值高于黑色像素点，经过求取局部最大值操作后相当于白色区域扩大，黑色区域缩小，故将此操作命名为膨胀；经过求取局部最小值操作后结果相反，故名腐蚀。膨胀和腐蚀可用集合论的方式加以定义[87]，膨胀定义为：

$$A \oplus B = \{a | B_a \cap A \neq \emptyset\} \tag{2.27}$$

腐蚀定义为：

$$A \ominus B = \{a | B_a \subseteq A\} \tag{2.28}$$

其中A是原图像，B通常被称为结构元素或核元素；a是A中元素的坐标，b是B中元素的坐标；$B_a \triangleq \{a + b | b \in B\}$，代表结构元素$B$平移$a$后点集。结构元素$B$的尺度一般都远小于$A$，其本身也是一个图像集合，一般为对称正方形，也可为圆形、椭圆或其它形状，B中有一个定义出来的参考点被称为锚点（Anchor point），其坐标为原点（即(0,0)），锚点不同形态学运算的结果也不相同，大部分情况下锚点位于图形的对称中心上。式(2.27)与式(2.28)可视为B在A上移动的过程，式(2.27)的含义是A用B膨胀后的结果是所有使B平移a后得到的新集合B_a与A仍存在交集时对应的a的集合；式(2.28)的含义是A用B腐蚀后的结果是所有使B平移平移a后得到的新集合B_a属于A时对应的a的集合。这一操作很类似于图像卷积运算，膨胀与腐蚀直观的原理示意图分别见图 2.31 及图 2.32，图中结构元素B中心的五角星符号就代表其锚点[88]。

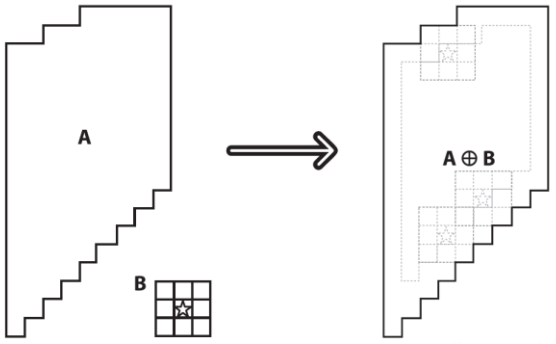

图 2.31 膨胀操作原理示意图

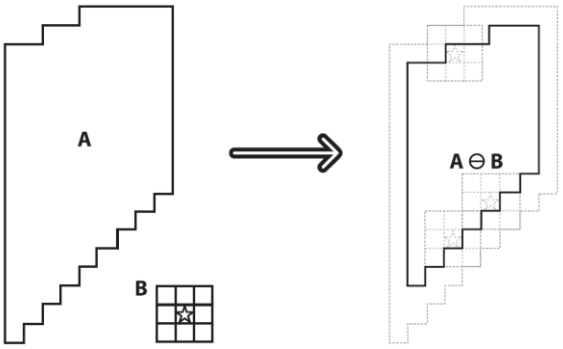

图 2.32 腐蚀操作原理示意图

膨胀与腐蚀可以级联结合使用，先膨胀后腐蚀称为闭运算（Closing Operation），先腐蚀后膨胀称为开运算（Opening Operation）；本文只会用到闭运算，其定义为：

$$A \bullet B = (A \oplus B) \ominus B \tag{2.29}$$

可以证明闭运算恒使原图像扩大，其主要用处在于消除图像中的小空洞（即被包围在一片白色区域内的小型黑色区域）。由于本文采用的 Hough 圆变换是基于边缘提取结果的，若图像中有较多零散区域会产生大量无效甚至是错误的边缘信息，这会导致后续更大的运算量及更高的误识别率；另一方面由于实际道路上红色物体不止有交通标志，广告牌、其余车辆等都有可能存在大面积红色区域，这些区域中一般都在红色底色上有其他图案，颜色分割后有可能形成圆形空洞进而造成误判；最后由于道路上的交通标志可能存在褪色、污损等问题，且光照、阴影也会在一定程度上影响其呈现出的颜色，所以通过颜色分割方法得到的二值图中可能无法包括完整的交通标志轮廓。针对这些问题本文使用闭运算对颜色分割后的二值图进行处理，以此消除零散区域、噪点及圆形小空洞，抑制边缘检测过程产生过多的错误边缘信息，同时也起到一定程度的构造完整交通标志区域的作用，有助于更准确的使用 Hough 圆变换定位图像中圆形交通标志区域。

闭运算使用的结构元素为正方形，锚点位于其中央，其大小为$Size_{stru} \times Size_{stru}$，为保证图像中央为一确定点，OpenCV 中要求$Size_{stru}$为奇数，通过在数据集上实验测试，$Size_{stru} = 15$时效果较好，此时处理结果见图 2.33 及图 2.34，图中左侧图片上蓝色圆圈代表 Hough 圆检测的结果。从中可以看到这两幅图中均存在红色车辆，且车体上有其它颜色图案，这样颜色分割的结果中就会存在很多噪点和零散区域，在不使用闭运算时会产生大量错误边沿信息进而导致错误的检测结果；而引入闭运算预处理后由于形成了连续的白色区域消除了过多的边缘信息，这样就很好的抑制了错误的 Hough 圆检测结果。

图 2.33 闭运算减少错误边缘信息对比图 1

图 2.34 闭运算减少错误边缘信息对比图 2

不过同时需要指出的是在测试过程中发现闭运算也存在一定副作用，参考图 2.35 所示处理结果，在未使用闭运算时可以正确检出全部 3 个圆形红色交通标志，不过有一个错误检测结果；使用闭运算后消除了错误检测结果，然而也导致两个禁止停车标志无法

检出。分析颜色分割结果可以看到，使用了闭运算后导致禁止停车标志与周围红色区域连在了一起，而此标志与限速标志不同不存在圆形内边缘，这就造成了漏检。不过此问题在实际应用中影响不大，由于车载安全预警系统的检测是一个连续动态过程，对于交通标识检测识别来说并不要求对每一帧图像都实现完全正确的检测，只要保证在车辆通过交通标志的全过程中实现正确的检出即可。由于本文使用的数据集是视频片段，图 2.35 对应视频片段在车辆与交通标志距离更近一些的时候使用闭运算也是可以正确检出禁止停车标志的。

图 2.35 闭运算影响交通标志正确识别对比图

2.4.2 Hough 检测算法定位圆形交通标志

Hough 变换由 Paul Hough 于 1962 年首次提出并使用其名字为此算法命名；目前广泛使用的 Hough 变换则是由 Richard Duda 和 Peter Hart 对其进行推广改进后得到；当前最新的实用版本则在此基础上进行了更多的优化，如多尺度 Hough 变换（Multi-Scale Hough Transform, MSHT）及累计概率 Hough 变换（Progressive Probabilistic Hough Transform, PPHT）[89]等。最初的 Hough 变换是用于检测直线与曲线的，之后被推广到检测圆、椭圆等其它形状上，其算法的基本思想均是将在一个空间中具有相同形状的曲线或直线映射到另一个坐标空间的一个点上形成峰值，进而把检测任意形状的问题转化为统计峰值问题[88]。Hough 圆变换是当前最常用的通用圆形检测算法，此处先对其基本原理做一简单介绍。

圆上任意一点在极坐标下都可以使用如下形式表示

$$\begin{cases} x = x_0 + r \cdot \cos\theta \\ y = y_0 + r \cdot \cos\theta \end{cases} \tag{2.30}$$

假设其半径r已知而圆心(x_0, y_0)未知，则以圆上任意一点为圆心，r为半径作圆得到的就是这一点可能对应的圆心(x_0, y_0)的轨迹；圆上所有点的轨迹必交于真实的圆心(x_0, y_0)处，这也就意味着(x_0, y_0)是轨迹坐标的峰值；这一过程可用图 2.36 表示。

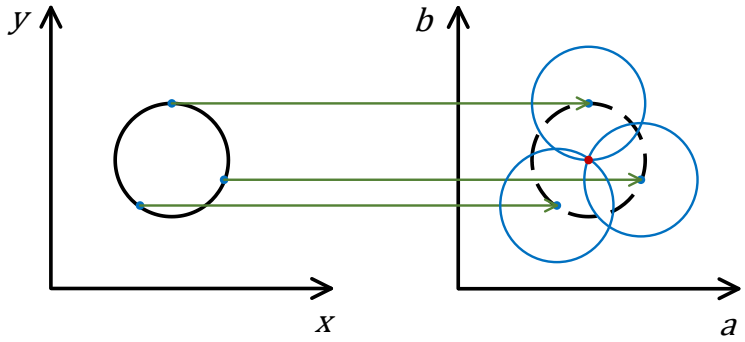

图 2.36 Hough 圆变换中单个圆上点的映射关系

事实上圆的半径r也是未知量，此时可在图 2.36 右侧图中增加第三个维度r，求解若干圆的交点就变为求解若干圆锥面的交点问题，如图 2.37 所示。实际图像中的圆形不会像图 2.36 或图 2.37 那样标准，严格的交点是无法求解的，此时实际 Hough 圆变换算法中使用的是峰值统计方法求解圆心(x_0, y_0)及半径r。由于 Hough 变换的常用性 OpenCV 中已经包含了 Hough 圆检测函数可直接调用，其内部使用的算法是 Hough 梯度法（Hough Gradient Method），这是此处介绍的标准 Hough 圆检测算法的优化改进，关于实际算法的具体实现方法此处从略，可参考 OpenCV 文档[90]。

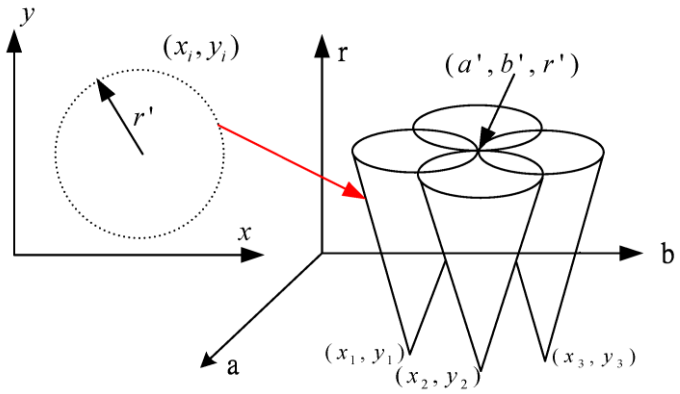

图 2.37 半径r未知时 Hough 圆变换原理示意图

一般来说 Hough 变换是针对边缘二值图进行的，故需要对原始灰度图（或二值图）先提取其边缘再进行 Hough 圆检测；当前最为成熟的边缘检测算法是 John F.Canny 于 1986 年提出的 Canny 算子和 Irwin Sobel 于 1968 年提出的 Sobel 算子；本文使用 Canny 算子作为边缘检测方法，由于原始图像在经上一步颜色分割算法处理后得到的已是二值图，边缘检测的阈值不影响结果，将其设置为黑色与白色中间任意一灰度值均可。

Hough 圆变换会检出很多大小位置各异的圆，其中很多都是虚假结果，为解决此问题一般会根据实际需要对圆的半径r及圆心间最小距离$Dist_{min}$加入约束。交通标志的大小形状有固定标准且基本不可能重叠，通过对测试数据集的视频片段进行统计分析总结得到的约束设置见表 2.6，其中r_{min}代表最小半径，r_{max}代表最大半径，单位均为像素。

表 2.6 Hough 圆变换约束条件设置表

r_{min}	r_{max}	$Dist_{min}$
15	90	50

从上文对 Hough 圆变换的原理介绍中可知，判断是否存在一个圆使用的是峰值统计方法，故 Hough 变换中最重要的参数就是峰值判断累加器阈值，本文将其记为Th_{Hough}，取值范围为$(0,+\infty)$。Th_{Hough}越小意味着可以检出越多的圆，然而很多圆可能其实是并不存在的；Th_{Hough}越大意味着检出的圆越接近完美的圆形，然而也有可能会造成漏检。由此可见需要通过调整Th_{Hough}实现检出率与误检率间的平衡，Th_{Hough}对检测结果的影响见图 2.38，图中可以很直观的看到Th_{Hough}取值不同对结果有明显影响。为更准确的定量评估Th_{Hough}的取值与系统性能的关系，对若干视频片段进行统计分析，这些视频片段中一共包含 40 个圆形交通标志，在Th_{Hough}取不同值时检出结果见图 2.39，从图中可以看到随着Th_{Hough}的增大检出率与误检率同时下降，不过二者下降趋势不同，当$Th_{Hough} < 10$时检出率保持在 100%，而误检率迅速降低；当$Th_{Hough} > 20$后误检率已经很低，继续增加Th_{Hough}会导致检出率迅速降低。根据图 2.39 可以确定Th_{Hough}的取值在 10~20 间较为合理，考虑到车载安全预警系统对交通标志的检测是对一个区间内若干采样帧进行的，

Th_{Hough}应该更多的偏向于降低误检率，因此本文取$Th_{Hough} = 20$。

图 2.38 Th_{Hough}不同时检测结果对比图

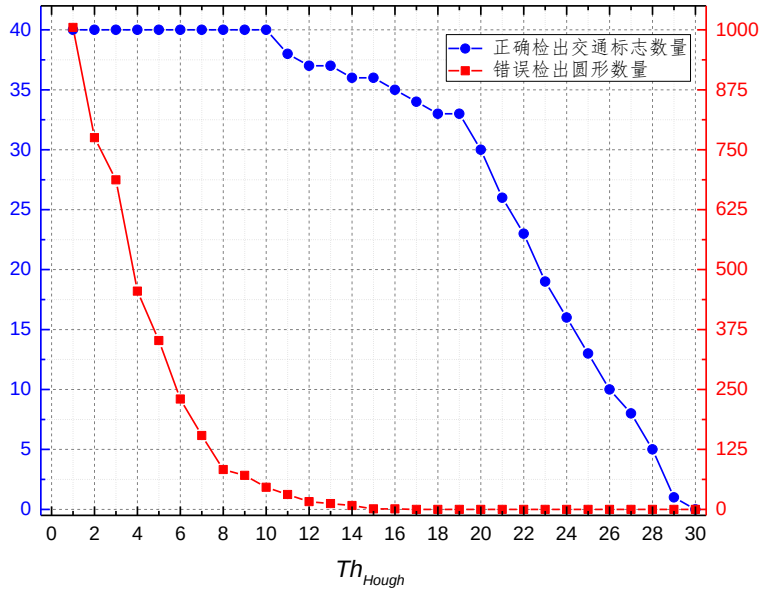

图 2.39 Th_{Hough}取值与检出结果关系图

2.5 算法在数据集上测试验证

对数据集中 271 段包含圆形红色交通标志的视频片段使用红色混合分割算法及 Hough 圆变换进行测试验证，其中一共包含 632 个交通标志，成功检出 437 个，误检 24 个，检出率 69.1%；正确检出的示例视频片段见图 2.40，从中可以看到，在天气光照良好时本文采用的算法具有较好的检出效果。

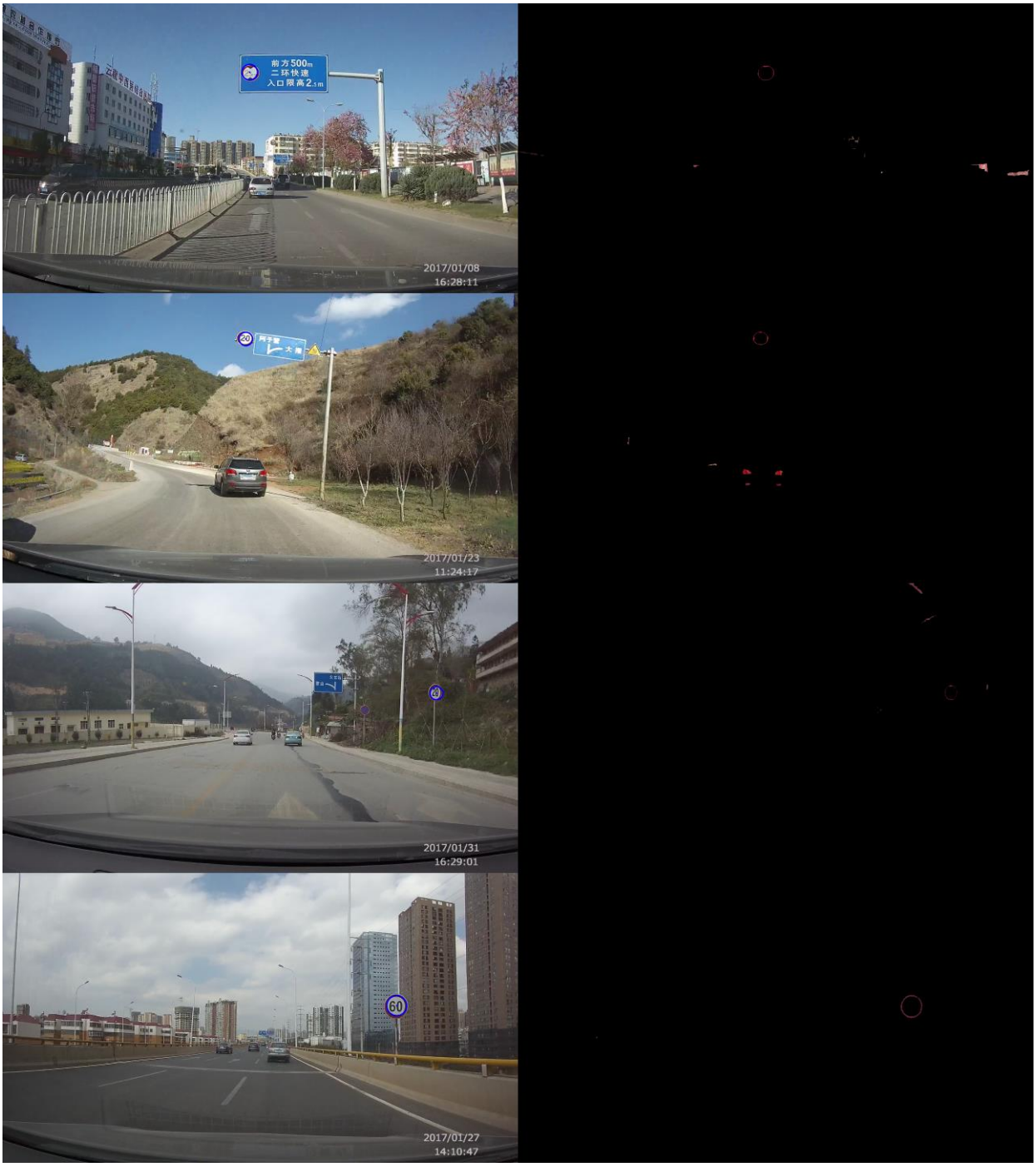

图 2.40 白天光照良好时交通标志定位效果示例

这 271 段视频片段中同时包含夜间及逆光情况下拍摄的视频，正如“2.1.2 数据集视频片段统计分析”一节中分析指出的，在这些情况下颜色分割方法会基本失效。夜间情况下只有当光照条件不复杂，车灯直接照射到交通标志上时方能正确定位交通标志，若光照复杂或车速较快会导致颜色偏移很严重或图像模糊，此时颜色分割结果基本无法使用，这两种情况的示例结果见图 2.41；逆光情况下，由于摄像头一般采用平均测光策略进行曝光控制，此时由于天空和太阳亮度极高交通标志区域在采集得到的视频中亮度会很低且颜色偏移严重，这些情况下颜色分割方法会变得很不可靠，逆光程度不强时尚可通过放宽各阈值来解决（见图 2.42），强逆光情况下由于饱和度和亮度会大幅降低且色调 H 分量也会发生变化，此时通过调整阈值反而会因为引入了更多的错误分割结果导致误识别（见图 2.43）。

若去除数据集中夜间及逆光情况下的视频片段，仅保留天气光照良好时的视频片段进行测试，此时检出率可以提高到 90%左右，部分仍不能正确定位的交通标志主要是由于拍摄角度倾斜导致其不是标准圆形而是椭圆，此时 Hough 圆检测算法会失效。

图 2.41 夜间情况下交通标志定位效果示意图

图 2.42 逆光情况下放宽阈值检测结果对比图

图 2.43 强逆光情况下放宽阈值导致误识别对比图

以上测试验证是使用数据集中的视频片段在计算机上运行算法仿真得到的，本文也同时在嵌入式系统原型样机上实现了此算法，嵌入式原型在实际道路上的测试验证结果见“4 仿真验证平台及嵌入式原型的搭建”一章。

2.6 本章小结

本章首先介绍了我们建立的中国道路交通标志公开数据集的基本情况并提供了获取链接，这是目前公开领域唯一的中国道路交通标志数据集，与国外同类公开数据集相比中这也是少有的高清视频数据集。建立数据集需要大量繁杂的重复工作，而对于当前热门的深度学习等方法来说完善的数据集又是至关重要的，故本文作者进行的这一基础性工作将对课题组后续在道路交通视觉感知领域开展研究提供支持，相信对相关领域的研究人员来说也将是颇有帮助的。对于车载安全预警系统来说警告标志与禁令标志的检测最为重要，而这两类标志在中国道路交通标志中的特征颜色为红色及黄色，因此本章对红色与黄色的分割方法进行了研究，详细对比了目前主流的 RGB 阈值分割法、基于 HSI 空间的分割方法、SVF 分割方法，提出了一种能充分发挥各方法优势的混合颜色分割策略，对红色的分割效果与 HSI 空间分割方法接近，对黄色的分割效果则优于 HSI 空间分割方法；并且通过分析优化此算法仅使用线性判别式就可以高效实现，在执行效率上大幅优于经典的 HSI 变换方法。最后本章采用形态学闭运算对颜色分割得到的二值图进行预处理，之后通过 Hough 圆变换实现了对红色圆形禁令标志的检测与定位，在数据集上的测试表明在天气光照良好时可以取得 90%左右的检出率。不过测试过程中也发现本文的颜色分割方法鲁棒性有所欠缺，在夜间及逆光环境下很难使用与普通环境下同样的阈值参数完成检测定位，而修改阈值参数则会导致正常情况下误检率上升，这一问题在本文的研究中未能很好的解决，仍需在后续研究中探索更有效的交通标志检测定位方法。另外由于时间有限，本章仅实现了对车载安全预警系统中最重要的红色圆形禁令标志的检测定位，后续研究中还需要对黄色三角形警告标志的检测定位方法进行研究。

3 混合切换多线程任务调度策略

本文第一章绪论中已经指出，对于车载安全预警系统而言系统响应的实时性是极为重要的，对于很多硬件资源有限的小型嵌入式设备而言多线程并行化技术是提高系统实时性的一个重要途径。本章将以交通标志检测识别系统为例，首先对系统的实时性要求进行分析，之后通过对此问题建立数学模型将其转换为控制系统设计问题，在此基础上提出混合切换调度策略并完成参数估计器设计，最后给出了 POSIX 标准下实际程序实现框架并在嵌入式原型样机上对多线程系统的实际性能进行了评估测试。

3.1 系统实时性要求分析

前文已经多次指出，对于车载安全预警系统而言系统实时性是一个十分重要的设计指标，然而对于系统应该具备怎样的处理响应速度才能满足实时性要求这一问题在目前的研究中普遍缺乏细致分析。一般来说，实时性是指在满足规定时间约束条件下系统及时做出正确响应的能力，对于不同系统来说实时性要求是不相同的，车载安全预警系统中不同子模块的时间约束条件也存在差异，下面以交通标志识别为例分析其实时性要求。

交通标志识别的实时性要求较为特殊，不同于前方防撞预警这类响应速度越快越好的系统，交通标志识别中响应速度其实并不是核心指标，相反漏检率更为重要，比如前方有一个限速标志，其实在哪一个时间点检测出这个标志并不重要，重要的是在车辆通过标志这段时间内一定要检测出这个标志的存在。正是由于这种特殊的实时性要求，采样时机变得尤为重要，只有在一个特定区域内采样并进行处理后得到的结果才是正确的，如图 3.1 所示。图中 C 点为交通标志放置点；A 点为有效识别区域的最远点，超过 A 点以外的区域交通标志在摄像头拍摄得到的图像中太小，无法正确识别；B 点为有效识别区域的最近点，BC 段由于距离交通标志太近，摄像头拍摄角度有限无法拍摄得到完整的交通标志，也无法正确识别；AB 段本文将其称为有效识别采样区域，即只有在这段区域中进行采样才能成功识别出 C 点的交通标志。图 3.1 中展现的是无遮挡长直道路情况下的有效识别采样区域，实际道路中，由于弯道或前方车辆遮挡等情况，AB 段的长度会缩短，不过这并不影响下文分析得到的结论。理论上说只要在有效识别采样区域中完成一次采样即可实现有效的交通标志检测，这就是交通标志识别系统中实时性约束的下界。当然，若能在有效识别采样区域中完成多次采样，系统的可靠性自然会随之增强，也就是两次采样间的时间间隔T_{sample}越短系统的实时性也就会越高。不过此处重点讨论的是

如何保证系统能达到最基本的实时性要求，故使用实时性约束的下界进行分析。

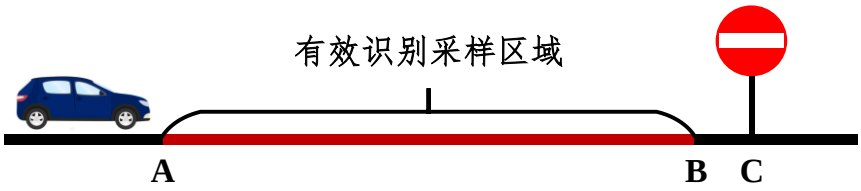

图 3.1 交通标志识别有效采样区域图示

上文的分析已经得出，交通标志识别系统实时性约束的下界是在有效识别采样区域内进行至少一次采样，这就意味着T_{sample}不能太长，即系统存在一个可允许的最长采样间隔时间$T_{sample(limit)}$，使得$\forall T_{sample}$均有$T_{sample} \leq T_{sample(limit)}$。要定量计算$T_{sample(limit)}$的核心在于估算 AB 段的长度，然而 A 点的位置和摄像头焦距、广角、图像分辨率等诸多因素有关，无法得到一个很通用的结论，仅能结合具体的系统实现进行分析。针对本文搭建的嵌入式原型验证平台，使用 KS2A17 摄像头模组搭配 100°广角镜头，并将摄像头输出分辨率设置为 800*600 的情况下，AB 段长度大约为 15m。摄像头分辨率的确定依据以及 AB 段长度的估算方法将放在本文最后一章“4.2.4 摄像头最佳输出分辨率分析优化”一节中进行讨论，此处直接给出最终计算结论：中国大部分城市道路最高限速为 80km/h，以此速度计算车辆前进 15m 所需的时间为 0.7s，这也就是本系统采样间隔时间的约束，即$T_{sample(limit)} = 0.7\text{s}$。

此处分析了交通标志识别系统的实时性要求，不过事实上这样的实时性要求对于很多相似系统来说都是适用的，这类系统一般都是基于采样的识别检测系统，如视频安防识别等，其典型特征是对采样间隔T_{sample}的上限存在约束，不满足此约束就有可能导致漏检，进而不能满足实时性要求。本章将以车载安全预警系统中交通标志识别系统为例进行讨论，然而相关分析及结论并不局限于此，类似系统均适用于本章所讨论的内容。

一般的，此类系统的实时性约束就是对T_{sample}上限的约束，故此处定义系统采样间隔最大值$T_{sample(max)}$用以确定系统是否满足实时性约束：

$$T_{sample(max)} \triangleq \max_{\forall k}\{T_{sample(k)}\} \tag{3.1}$$

其中$T_{sample(k)}$表示第k次采样间隔，当$T_{sample(max)} < T_{sample(limit)}$时，即称系统满足实时性约束。

粗略的看系统的实时性优化目标就是令$T_{sample(max)}$最小，即

$$\min\{T_{sample(max)}\} = \min\{\max_{\forall k}\{T_{sample(k)}\}\} \tag{3.2}$$

不过严格来说只用最大值$T_{sample(max)}$来衡量系统实时性是存在一些缺陷的，$T_{sample(max)}$只能判断是否满足实时性约束而不能很好的衡量实时性好坏，原因在于$T_{sample(max)}$相同的情况下T_{sample}的分布可能并不相同，故应该从统计分析的角度出发研究T_{sample}的分布，本章最后测试验证时就将采用此方法。不过下文提出的控制策略是针对每一次采样独立进行控制的，相当于以最小化每次T_{sample}为控制目标，即$\min\{T_{sample(k)}\}$，此时式(3.2)作为系统的实时性优化目标是正确的，只是其中的$\forall k$要修改为一个区间内的k。一言以蔽之，要提高系统实时性就是要提高系统整体的吞吐量，进而减小采样间隔T_{sample}，本章就将对此问题进行详细分析讨论，重点研究如何进行多线程任务调度以实现T_{sample}最短的目标。

3.2 基本任务调度问题建模

交通标志检测识别系统的基本任务是采集来自摄像头的图像，之后进行颜色分割、标志定位、标志分类识别等步骤后输出交通标识的含义，我们将进行一次这一过程称为完成一次处理任务。由于不同时刻环境存在差异，完成一次处理任务所需的时间是不固定的，比如当周围环境较复杂或存在较多交通标志时，颜色分割的结果中就会存在较多的边缘信息，之后 Hough 变换所需的时间就会偏长。由于车辆的运动和周围环境的变化均是与时间相关的，完成一次处理任务所需时间自然也是时间的函数。我们不妨用$task(t)$函数表示t时刻进行摄像头图像采样并完成一次处理任务所需的时间，对于交通标志检测识别系统及车载安全预警系统其它基于视觉的模块而言，完成一次处理任务所需的时间基本取决于输入图像的复杂程度，故此处将完成一次处理任务所需时间视为图像采样时间点t的函数是合理的。

假设系统存在超过一个线程在并行执行处理任务，我们将第一个线程称为主线程，其余N个线程称为从线程，此处要求总线程数$N+1$小于等于处理器核心数，这样可以保证多个线程间是真正同时并行运行的。若总线程数$N+1$大于实际处理器能提供的逻辑核心数，多个线程间其实并不是真正同时并行运行的，操作系统会使用时间片轮转（Round robin）或其他类似算法进行任务调度，此时单个线程的执行时间是和线程数量会耦合在一起，上文所做的基本假设——$task(t)$仅是时间的函数就不再成立。

为进一步说明上述线程总数约束条件的合理性，可通过实际测试加以证实。使用一个固定的程序（实际使用的程序将计算小于 100000 的质数个数）作为任务，这样期望有$task(t) \equiv \text{constant}$，即完成一次任务所需时间固定为常数。在拥有 16 个 CPU 核心的性能测试平台上同时新建若干个线程，统计单线程执行时间可以得到表 3.1 和表 3.2，分别对应线程数小于等于 CPU 核心数与线程数大于 CPU 核心数的情况。

表 3.1 线程数与单线程执行时间关系表（线程数小于等于 CPU 核心数）

线程数	1	2	4	8	12	16
单线程执行时间（s）	24.9	23.9	23.9	24.6	25.5	26.5

表 3.2 线程数与单线程执行时间关系表（线程数大于 CPU 核心数）

线程数	20	24	32	64	128	256
单线程执行时间（s）	31.9	37.6	52.2	99.3	200.6	400.8

分析表 3.1，当线程数小于CPU核心数时，单线程执行时间均值$task(t) = \mu = 24.9$，标准差$\sigma = 0.91$，基本为一固定常数；然而分析表 3.2，当线程数大于 CPU 核心数时，单线程执行时间显然不是常数，对线程数与单线程执行时间作图并进行线性拟合，得到图 3.2，从中可以看到当线程数大于等于 CPU 核心数时，线程数与单线程执行时间为严格的线性关系，$task(t) = 1.56N + 0.82$，拟合直线的决定系数$R^2 = 0.99995$。可以看到此时$task(t)$是与线程数N有关的，为避免这种耦合关系，本章的讨论中对线程数量进行了限制，要求总线程数小于等于处理器核心数。

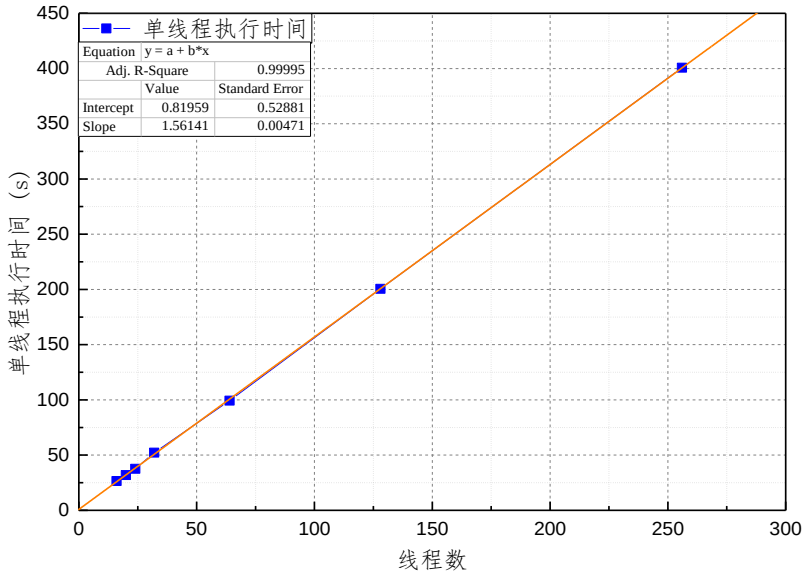

图 3.2 线程数与单线程执行时间关系图（线程数大于等于 CPU 核心数）

事实上对于高度计算密集型任务而言建立超过处理器核心数的线程也是没有意义的，当线程数等于核心数时处理器利用率已基本达到最大值，进一步增加线程数量不但不能提高系统运行效率，因为线程调度存在开销等原因可能会导致系统效率反而下降。可用单线程执行时间与线程数的比值定量衡量系统效率，此值表示的是等效完成一次处理任务所需的时间，结果见表 3.3 及图 3.3，从中可以清晰的看到当线程数超过处理器核心数之后，系统效率维持不变，这一点也可以由图 3.2 中拟合直线的表达式推导得到：

$$\lim_{N\to\infty}\frac{task(t)}{N+1}=\lim_{N\to\infty}\{1.56+\frac{0.82}{N+1}\}=1.56\approx 1.6 \tag{3.3}$$

此结论也可以说明上文对线程总数所做的约束是合理的。

表 3.3 线程数与系统效率关系表

线程数 $N+1$	1	2	4	8	12	16	20	24	32	64	128	256
$\frac{task(t)}{N+1}$	24.9	12.0	6.0	3.1	2.1	1.7	1.6	1.6	1.6	1.6	1.6	1.6

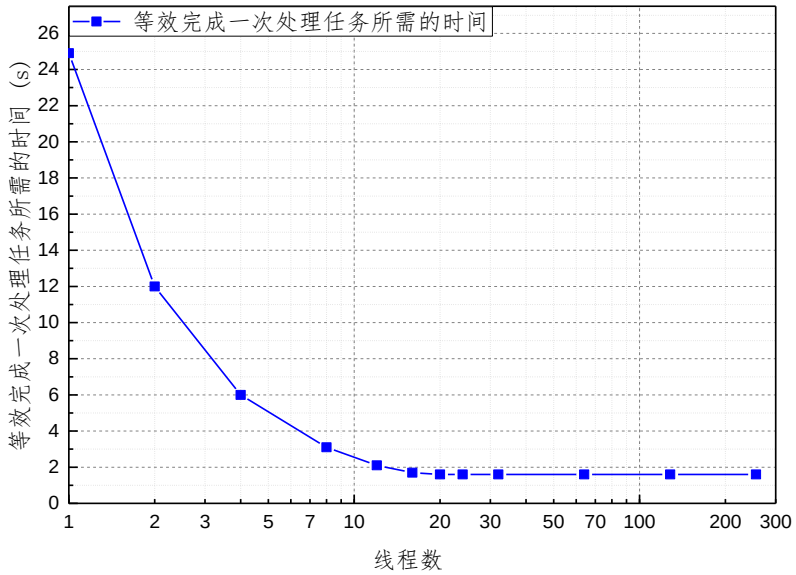

图 3.3 线程数与系统效率关系图

对于主线程而言，为实现最高 CPU 占用率，显然连续进行采样处理是最佳策略，即主线程本身并不进行任务调度，固定采用图 3.4 所示这种循环执行策略运行。

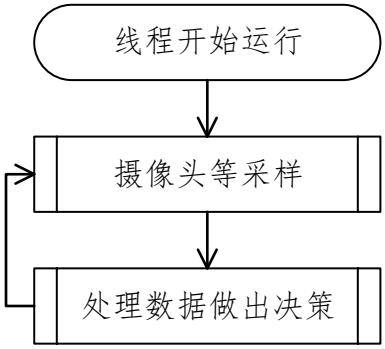

图 3.4 主线程执行流程图

在此策略下，如果我们用M_k表示主线程开始进行第k次处理任务的时间点，则有：

$$M_{k+1} = M_k + task(M_k) \tag{3.4}$$

即下一次处理任务开始于本次处理任务完成之后。

特别的，对于只有一个主线程的单线程系统来说，系统第k次采样间隔$T_{sample(k)}$为：

$$T_{sample(k)} = M_k - M_{k-1} = task(M_{k-1}) \tag{3.5}$$

此时有

$$T_{sample(max)} = \max_{\forall k}\{task(M_k)\} \tag{3.6}$$

由于M_k采样的离散性，$\max_{\forall k}\{task(M_k)\}$并不等于$\max\{task(t)\}$，这一点可以用图 3.5 说明。作为一般规律存在$\max_{\forall k}\{task(M_k)\} \le \max\{task(t)\}$，据此考虑最坏情况，可以得到单线程系统满足实时性约束的充分不必要条件为：

$$\max\{task(t)\} \le T_{sample(limit)} \tag{3.7}$$

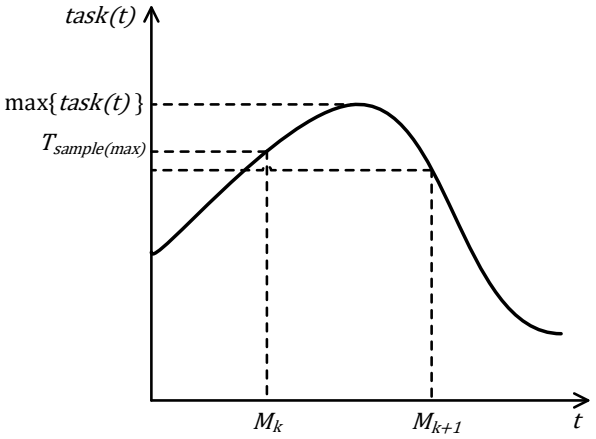

图 3.5 $T_{sample(max)}$与$\max\{task(t)\}$的差异性

从式(3.8)中可以看到，单线程情况下要提高系统实时性只能从加快算法执行速度进而减小$task(t)$入手，上一章中采用的混合颜色分割策略与传统 HSI 空间转换相比执行速度上就有很大的优势，这对提高系统实时性是很有用的。然而正如本文绪论部分所述，使用多线程并行化技术可显著提升多核处理器上程序执行效率，考虑到对算法本身进行并行化难度较高，本文使用的策略是多个线程执行相同的算法，下文将会证明，配合恰当的任务调度算法此策略可以显著提升系统实时性。

若简单的以最高 CPU 占用率或最高任务并行度为任务调度算法设计目标，其余从线程按照与主线程相同的图 3.4 所示策略执行即可，即：

$$T_{k+1}^n = T_k^n + task(T_k^n)\,, n \le N \tag{3.8}$$

其中T_k^n表示第n个从线程开始进行第k次处理任务的时间点，n满足线程总数的约束。

然而式(3.8)给出的任务调度策略是存在问题的，考虑一个特殊情况，当$M_k = T_k^n$时，对于任意$i > k$来说都有$M_i = T_i^n$，由于各个线程执行的算法是完全相同的，使用同一个时间点t采样得到的图像进行处理后的结果也是完全相同的，此时多线程除了占用更多 CPU 资源的副作用外并没有带来任何有用的价值，因此我们需要一种新的调度方法来解决这一问题。

3.3 任务调度策略设计

3.3.1 理想任务调度算法

式(3.8)所示调度策略存在问题的根本原因在于T_k^n可能会和M_k重合，为避免此问题，可附加一个延时项：

$$\begin{aligned} T_{k+1}^n &= T_k^n + task(T_k^n) + d_k^n \\ s.t.\quad & d_k^n \geq 0, n \leq N \end{aligned} \tag{3.9}$$

其中d_k^n为附加延时项，增加这一项后从线程的执行流程图见图 3.6，由于附加延时显然只能为正值，故存在$d_k^n \geq 0$的约束条件。不过为简单起见在本节中讨论理想多线程调度算法时忽略此约束条件，之后再讨论此约束条件的影响。

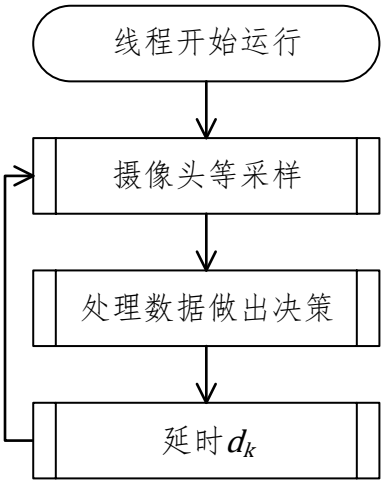

图 3.6 附加延时项后从线程执行流程图

为方便讨论，对若干从线程及主线程间的同步关系加上如下约束：

$$M_k \leq T_k^n < M_{k+1} \tag{3.10}$$

$$T_k^i < T_k^j, \forall 1 \leq i < j \leq N \tag{3.11}$$

式(3.10)的约束可以保证主线程与从线程执行次数是相同的，从线程通过式(3.9)中d_k^n项的调整可以使其开始时间T_k^n落在M_k与M_{k+1}之间；式(3.11)则是对各从线程关系的约束，将较先执行的从线程表示为序号较小的从线程，这主要是为了下文数学推导的方便所做的约束。加上这两个约束条件后各线程开始时间在时间轴上的关系就可以用图 3.7 表示，其中M_{k-1}、M_k、M_{k+1}的位置根据式(3.4)由$task(t)$唯一确定，这是无法控制的；而T_k^n的位置则可由式(3.9)中$d(k)$项任意控制，因此从控制的角度看我们只要对每一个$[M_k, M_{k+1})$子区间内T_k^n的位置进行控制即可。

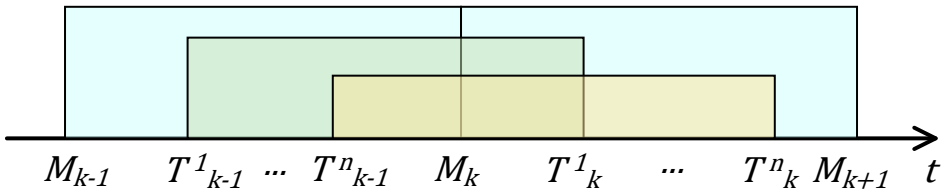

图 3.7 各线程在时间轴上的关系

结合式(3.2)，对于任意$[M_k, M_{k+1})$子区间内的优化目标均为

$$\min\left\{\max_{1\le i\le N+1}\left\{T_k^i - T_k^{i-1}\right\}\right\} \tag{3.12}$$

其中最内层的max函数求出的是对特定的某个T_k^i来说，其与前一个线程间的采样间隔，考虑到式(3.11)的约束条件，$T_k^i - T_k^{i-1}$恒为正值。为了公式的统一性，此处引入了T_k^0及T_k^{N+1}这两个虚拟量（并不存在第 0 及第$N+1$个从线程），从图 3.7 及式(3.4)中易知

$$T_k^0 \triangleq M_k, \qquad T_k^{N+1} \triangleq M_{k+1} = M_k + task(M_k) \tag{3.13}$$

将此式视为T_k^0及T_k^{N+1}的定义式。

在将$task(t)$视为已知函数时，式(3.12)是存在解析解的：当且仅当T_k^n符合式(3.14)时，式(3.12)取得最小值$\frac{task(M_k)}{N+1}$

$$T_k^n = M_k + n\cdot\frac{task(M_k)}{N+1}, n = 1,2,3\ldots N \tag{3.14}$$

下面来证明此结论，先引入辅助变量a_i，令

$$a_i = T_k^i - T_k^{i-1} \tag{3.15}$$

代入式(3.12)有

$$\min\left\{\max_{1\le i\le N+1}\{a_i\}\right\} \tag{3.16}$$

此外

$$\sum_{i=1}^{N+1} a_i = \sum_{i=1}^{N+1}(T_k^i - T_k^{i-1}) = T_k^{N+1} - T_k^0 = M_{k+1} - M_k = task(M_k) \tag{3.17}$$

根据最大值的定义，显然有

$$\max_{1\le i\le N+1}\{a_i\} \ge a_j, \forall j \in [1, N+1] \tag{3.18}$$

$$\Longrightarrow \quad (\mathrm{N}+1)\cdot\max_{1\le i\le N+1}\{a_i\} \ge \sum_{j=1}^{N+1} a_j = task(M_k) \tag{3.19}$$

$$\Longrightarrow \quad \max_{1\le i\le N+1}\{a_i\} \ge \frac{task(M_k)}{N+1} \tag{3.20}$$

“=”号可取，此时结合式(3.18)与式(3.20)有

$$\max_{1\le i\le N+1}\{a_i\} = a_j = \frac{task(M_k)}{N+1}, \forall j \in [1, N+1] \tag{3.21}$$

即当且仅当$a_i = \frac{task(M_k)}{N+1}, i = 1,2,3 \dots \mathrm{N}+1$时，$\max_{1\le i\le N+1}\{a_i\}$有最小值$\frac{task(M_k)}{N+1}$，代入式(3.15)有

$$T_k^i = T_k^{i-1} + \frac{task(M_k)}{N+1} \tag{3.22}$$

很显然这个递推式构成一等差数列，求出其通项公式有

$$T_k^n = T_k^0 + n\cdot\frac{task(M_k)}{N+1} = M_k + n\cdot\frac{task(M_k)}{N+1}, n = 1,2,3 \dots N \tag{3.23}$$

对比式(3.22)与式(3.14)，二者完全相同，证毕。

式(3.14)给出的线程调度方法即为理想任务调度算法，上文已证明按此策略进行调度可实现T_{sample}最小化的目标，将其带入式(3.9)，可求出附加延时项的表达式为

$$d_k^n = M_{k+1} + n\cdot\frac{task(M_{k+1})}{N+1} - {T'}_k^n \tag{3.24}$$

其中${T'}_k^n$表示没有附加延时d_k^n时从线程的结束时刻：

$${T'}_k^n \triangleq T_k^n + task(T_k^n) \tag{3.25}$$

之所以将此算法称为理想任务调度算法的原因在于这只是一个理论最优解，实际上是无法实现的，主要原因有两个：

(1) 式(3.24)中$task(M_{k+1})$这一项实际上是未知的，我们既不知道$task(t)$函数的表达式，由于M_{k+2}是未来时刻（参考图 3.7），也无法根据式(3.4)由$task(M_{k+1}) = M_{k+2} - M_{k+1}$求出$task(M_{k+1})$。

(2) 式(3.9)中其实是存在$d_k^n \ge 0$的约束条件的，而式(3.24)中显然有可能会出现$d_k^n < 0$的情况（${T'}_k^n > M_{k+1} + n\cdot\frac{task(M_{k+1})}{N+1}$时），此时意味着从线程实际结束的时间${T'}_k^n$晚于其本应开始的时间。

这两个问题实质上分别是估计和控制问题，下文就将分别解决之。

3.3.2 控制系统建模

由于理想任务调度算法实际是不可实现的，我们需要用一些实际可实现的算法来逼近理想任务调度算法，可以将设计构造这一算法的过程抽象为控制系统的设计，这样就可以借用控制理论中很多成熟的工具与方法对其进行分析。显然本系统是一个离散控制系统，我们将系统输入定义为附加延时d_k^n，系统状态变量定义为主线程开始时间M_k及从线程开始时间T_k^n，系统输出定义为实际从线程开始时间与理想任务调度算法给出的从线程开始时间之间的差值，整理上文得到的式(3.4)、式(3.9)、式(3.14)，可用向量的形式将其改写为标准的离散系统状态方程及输出方程：

$$\begin{bmatrix} M(k+1) \\ T_1(k+1) \\ T_2(k+1) \\ \cdots \\ T_n(k+1) \end{bmatrix} = \begin{bmatrix} M(k) \\ T_1(k) \\ T_2(k) \\ \cdots \\ T_n(k) \end{bmatrix} + task\left(\begin{bmatrix} M(k) \\ \boldsymbol{T}(k) \end{bmatrix}\right) + \begin{bmatrix} 0 \\ d_1(k) \\ d_2(k) \\ \cdots \\ d_n(k) \end{bmatrix} \tag{3.26}$$

$$\begin{bmatrix} y_1(k) \\ y_2(k) \\ \cdots \\ y_n(k) \end{bmatrix} = \begin{bmatrix} -1 & 1 & 0 & \cdots & 0 & 0 \\ -1 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -1 & 0 & 0 & \cdots & 1 & 0 \\ -1 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} M(k) \\ T_1(k) \\ T_2(k) \\ \cdots \\ T_n(k) \end{bmatrix} - \frac{task\big(M(k)\big)}{N+1} \begin{bmatrix} 1 \\ 2 \\ \cdots \\ n \end{bmatrix} \tag{3.27}$$

其中$T_n(k)$等即上文所用的T_k^n，二者含义完全相同，仅是使用了不同的表达方式，上文使用T_k^n的形式是为了避免过多的括号嵌套使得形式上更美观，而此处使用$T_n(k)$是为了符合控制理论中通用表达形式，下文在不引起歧义的情况下会根据需要交替使用这两种表达形式。$\boldsymbol{y}(k)$表示实际从线程开始时间与理想任务调度算法给出的从线程开始时间之间的差值，不妨将其称为调度偏差。

式(3.26)与式(3.27)中由于包含非线性项$task(t)$，故构成非线性离散系统。系统的控制目标显然是令调度偏差为 0，此时相当于按照理想任务调度算法进行任务调度，即

$$\boldsymbol{y}(k) \to \boldsymbol{0} \tag{3.28}$$

此系统控制难点在于非线性项$task(t)$未知，故我们可以构造如图 3.8 所示的控制系统来实现对$task(t)$的估计和对$\boldsymbol{y}(k)$的控制。

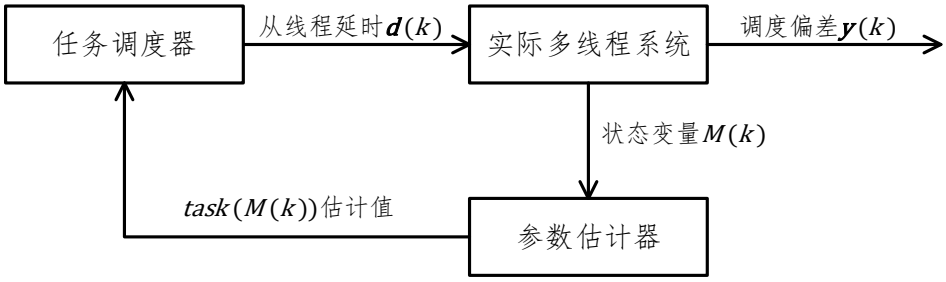

图 3.8 任务调度算法控制系统框图

由于在本文研究的线程调度问题中，式(3.26)与式(3.27)给出的模型本身是极为精准的，与系统的实际情况相差极小，故此处没有使用最常见的对输出量做闭环的控制方法，而是采用了图 3.8 中这种类似模型预测控制（Model Predictive Control, MPC）系统的结构。然而又不同于典型的 MPC 系统，式(3.26)中的各状态变量可以直接观测得到的，不需要再引入状态估计器（State Estimation）对其进行估计，模型中的未知量$task(t)$其实可以视为一个时变的模型参数，故此处将其抽象为一参数估计器；而 MPC 的主控制器在图 3.8 中就对应任务调度器，MPC 作为一个最优控制器，其基本原理可以视为通过模型对输入进行计算优化，从而使输出达到某种意义上的最优，对于本文研究的任务调度器而言，其目的在于根据线程模型计算结果调整输入向量$\boldsymbol{d}(k)$以实现系统输出$\boldsymbol{y}(k)$趋近于 0，显然此处的任务调度器可以视为 MPC 控制器的一个特例，其预测时域长度（Prediction Horizon）为 1。事实上，参数估计器和任务调度器的引入分别解决了上一节最后提出的理想任务调度算法不可行的两个问题（$task(t)$未知与$\boldsymbol{d}(k)$存在约束），他们可以独立进行设计，再进行搭配组合即可得到最终所需的任务调度策略。下文就将按照这个思路分别对任务调度器及参数估计器进行设计，最后将其结合起来进行仿真验证并评估其性能。

另外仔细分析式(3.26)与式(3.27)，$\boldsymbol{y}(k)$与$\boldsymbol{T}(k)$的各分量间是完全解耦的，这也就意味着完整的系统可以分解为几个独立的子系统分别进行控制。事实上这一点从问题的实际意义上也可以看到，多个从线程之间是没有任何联系的，他们的调度策略也是完全独立的。因此可以将$\boldsymbol{T}(k)$的维度降为一维标量，即只研究存在一个从线程时的任务调度策略，此时式(3.26)与式(3.27)可简化为

$$\begin{bmatrix} M_{k+1} \\ T_{k+1} \end{bmatrix} = \begin{bmatrix} M_k \\ T_k \end{bmatrix} + task\left(\begin{bmatrix} M_k \\ T_k \end{bmatrix}\right) + \begin{bmatrix} 0 \\ d_k \end{bmatrix} \tag{3.29}$$

$$y_k = [-1 \quad 1] \begin{bmatrix} M_k \\ T_k \end{bmatrix} - \frac{task(M_k)}{2} \tag{3.30}$$

从模型中可以看到更多数量从线程的调度策略与之完全相同，为简单起见下文对任务调度器及参数估计器的设计过程中就只考虑从线程数$N=1$时的情况，可以很容易将其推广到更多数量的从线程情况下，本章“3.4 任务调度策略程序框架的实现及”节中将对更多数量的从线程实现算法及实际性能进行说明。

3.3.3 任务调度器设计

在进行任务调度器设计时，我们将$task(t)$视为已知函数，此时重点要解决的问题是式(3.9)中$d_k \geq 0$的约束。严格来说加上此约束条件后式(3.10)的约束$M_k \leq T_k < M_{k+1}$就有可能会无法满足，考虑以下特例：

假设$M_k = t_0, T_k = t_0 + 0.5A$

$$task(t) = \begin{cases} A, & t \leq t_0 + 0.5A \\ \alpha A, & t > t_0 + 0.5A \end{cases}, \alpha < 0.5 \tag{3.31}$$

其中A为一大于 0 的常数，根据式(3.29)可计算出$M_{k+2} = t_0 + (\alpha+1)A < t_0 + 1.5A$，$T_{k+1} = t_0 + 1.5A + d_k$，由$d_k \geq 0 \Rightarrow T_{k+1} \geq t_0 + 1.5A$，此时有$T_{k+1} > M_{k+2}$，不符合式(3.10)的约束条件，这可用图 3.9 加以说明。

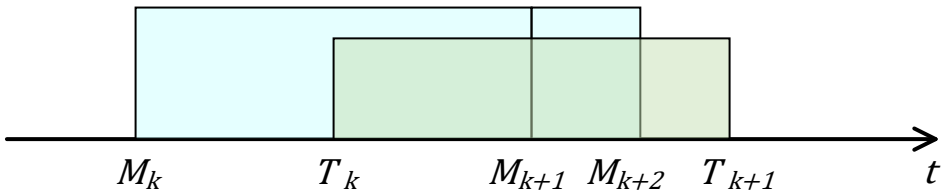

图 3.9 由于$task(t)$变化导致T_k不符合次序约束示意图

造成这一问题的本质原因是由于$task(t)$突然减小，导致主任务执行时间变短，在前一个从任务还没执行完的时候主任务已执行完毕。由于我们只能控制d_k，此问题并没有什么很好的解决方法，不过考虑加入式(3.10)约束的主要目的是保证主线程与从线程执行次数相同进而便于进行数学推导，对于实际算法影响不大，故此处引入一些特殊的数学处理手段来回避此问题。

具体来说，仔细分析图 3.9，其实完全可以直接跳过T_{k+1}将T_{k+1}视为T_{k+2}，考虑T_k下标序列应该有连续性，不妨强制将T_{k+1}取为M_{k+1}，并令$task(T_{k+1}) = task(T_k) - (M_{k+1} - T_k)$，这样这样就相当于引入了一次虚拟线程采样，由于$T_{k+1}$与$M_{k+1}$是重合的，这样做并不影响实际采样间隔$T_{sample}$。考虑到任务调度策略本身是针对每一个$[M_k, M_{k+1})$子区间内的从线程单独进行的，即每次计算$T_k$是根据其最近一次$M_k$进行，并不会保存之前的状态，这事实上已默认使用了此策略，故其实可以直接忽略此约束条件。

3.3.3.1 下界限幅调度策略

要处理$d_k \geq 0$的约束最简单的办法就是根据式(3.24)计算出d_k后，若其值小于 0，直接将其视为 0，即对d_k的结果进行下界限幅，此时的任务调度策略可表示为：

$$d_k = \begin{cases} M_{k+1} + \dfrac{task(M_{k+1})}{2} - T'_k & ,T'_k \leq M_{k+1} + \dfrac{task(M_{k+1})}{2} \\ 0 & ,T'_k > M_{k+1} + \dfrac{task(M_{k+1})}{2} \end{cases} \tag{3.32}$$

然而在式(3.32)给出的调度策略下调度偏差$\boldsymbol{y}(k)$并不一定收敛于 0，具体来说，即使在y_k已经等于 0 后，若$task(t)$出现负向阶跃变化（即由较大值变为较小值）时式(3.32)给出的策略即会存在静差，即$\lim_{k\to\infty} y_k > 0$。此时的仿真结果见图 3.10，仿真初始条件$M_0 = 0, T_0 = 0.5$，$task(t)$函数为：

$$task(t) = \begin{cases} 1, & t < 3 \\ 0.7, & t \geq 3 \end{cases} \tag{3.33}$$

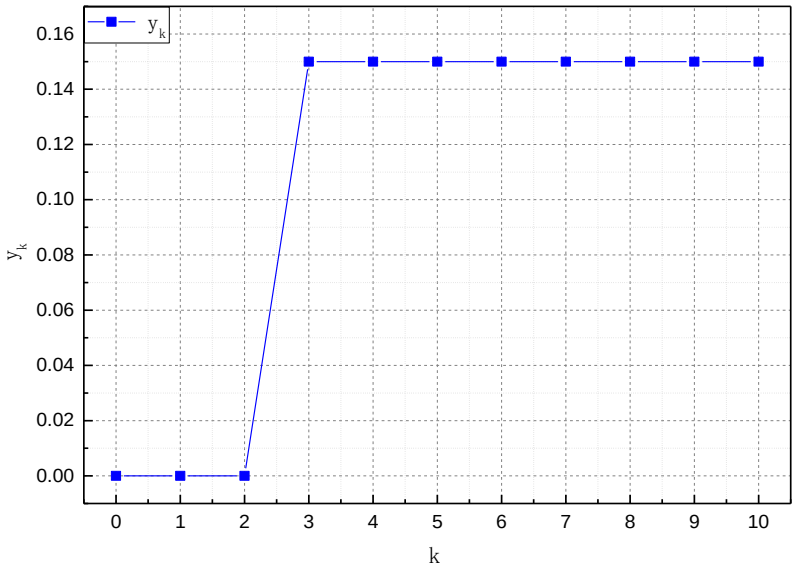

图 3.10 下界限幅调度策略仿真结果

可结合图 3.11 较为形象的说明此静差产生原因，当$task(t)$函数减小时主线程执行时间变短，导致从线程结束时间已经晚于理想调度算法给出的此线程下一次应该开始的时间，然而式(3.32)缺少将线程执行时间提前的能力（由于d_k最小值为 0），所以无法消除此静差。其实对比图 3.11 与图 3.9 及式(3.33)与式(3.31)会发现二者其实是同一个问题，

故可以使用相似的方法进行处理，即引入一次虚拟采样主动推迟一个周期，在下一个$[M_k, M_{k+1})$子区间内再进行调度，本文将这一策略称为“虚拟采样调度策略”。

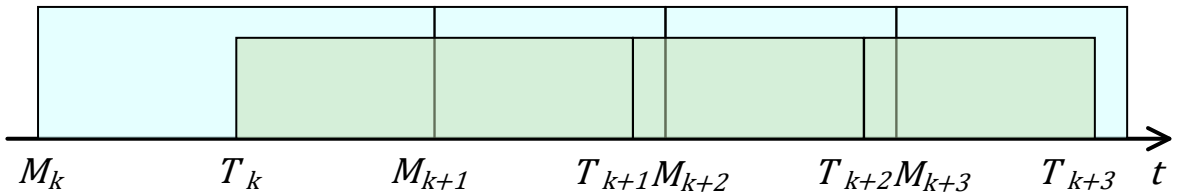

图 3.11 d_k下界限幅调度策略静差示意图

3.3.3.2 虚拟采样调度策略

实际算法中，虚拟采样和推迟一个周期也是用延时项d_k实现的，其基本原理可用图 3.12 表示，通过d_k将T_{k+1}设置为M_{k+2}，然而此时从线程并不实际执行采样处理任务，这样就相当于其完成时间$T'_{k+1} = T_{k+1} = M_{k+2}$，此时通过延时项$d_{k+1}$就可以将$T_{k+2}$设置为$[M_{k+2}, M_{k+3})$区间内任意值了。

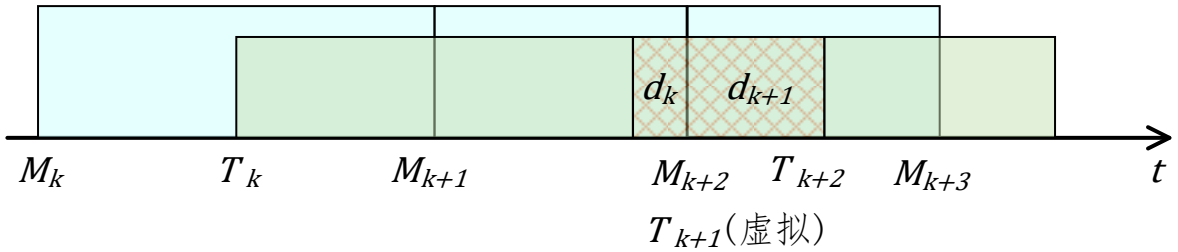

图 3.12 推迟采样调度策略原理示意图

将此策略用公式表示出来就是

$$d_k = \begin{cases} M_{k+1} + \dfrac{task(M_{k+1})}{2} - T'_k, & T'_k \le M_{k+1} + \dfrac{task(M_{k+1})}{2} \\ M_{k+2} - T'_k, & T'_k > M_{k+1} + \dfrac{task(M_{k+1})}{2} \end{cases} \tag{3.34}$$

为了能用统一的状态方程式(3.29)描述T_{k+1}时刻的采样为虚拟采样这一特性，此处需要对d_{k+1}进行一些特殊处理，根据理想调度算法式(3.14)并结合图 3.12，T_{k+2}的期望值为$M_{k+2} + task(M_{k+2})/2$；又根据状态方程式(3.29)，$T_{k+2} = T_{k+1} + task(T_{k+1}) + d_k$；且$T_{k+1} = M_{k+2}$；联立以上几式有

$$M_{k+2}+\frac{task(M_{k+2})}{2}=T_{k+1}+task(T_{k+1})+d_{k+1}$$
$$\Longrightarrow \quad d_{k+1}=\frac{task(M_{k+2})}{2}-task(T_{k+1}) \tag{3.35}$$

由于状态方程式(3.29)中缺乏描述虚拟线程采样的能力，此处引入了虚拟延时来解决这一问题，式(3.35)中计算出来的d_{k+1}相当于强制去掉了线程执行时间$task(T_{k+1})$后的结果，其值显然会小于 0，故称虚拟延时。这是为了仿真和模型需要引入的虚拟量，故无需考虑$d_k \geq 0$的约束条件，实际程序中在T_{k+1}时刻不建立采样处理任务就相当于$task(T_{k+1})=0$了，此时也无需使用式(3.35)来计算d_{k+1}，直接根据通用的式(3.34)进行即可。

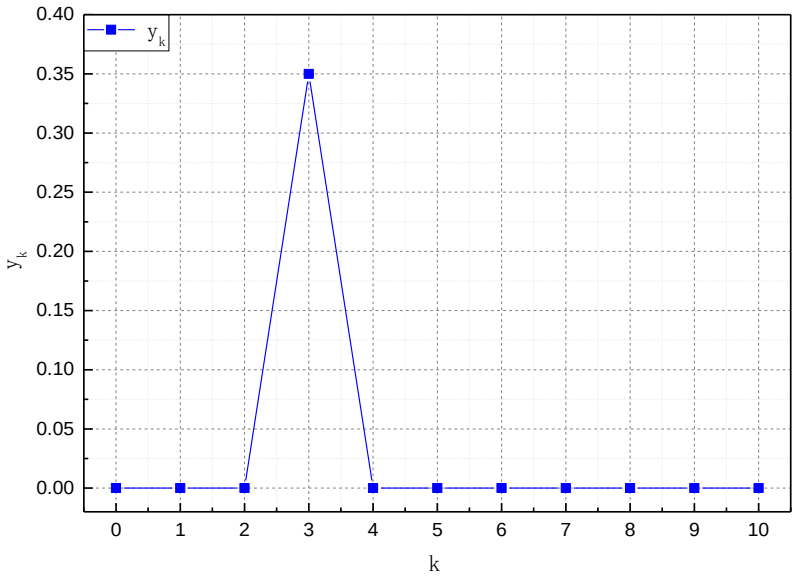

图 3.13 虚拟采样调度策略仿真结果

使用与上一节相同的仿真参数仿真此调度策略，结果见图 3.13，可以看到此策略稳态静差为 0。然而此策略依然存在缺陷，由于各种随机误差的存在，$task(t)$函数其实并不可能完全平稳的等于一个常数，为表征此特性我们可以在一个确定函数的基础上叠加一个随机误差项，即

$$task(t)=f(t)+w \tag{3.36}$$

其中w为系统白噪声，不妨假设其服从正态分布，即$w\sim\mathcal{N}(0,\sigma^2)$。

据此我们可以将仿真用$task(t)$函数式(3.33)进行修改：

$$task(t)=\begin{cases}1, & t<3\\ 0.7+w, & t\geq 3\end{cases} \tag{3.37}$$

其中$w\sim\mathcal{N}(0,0.001^2)$。

仿真结果见图 3.14，为方便对比此处同时给出了上一节的调度策略仿真结果，可以看到上一节提出的下界限幅调度策略尽管存在静差，然而它是稳定的；而本节提出的虚拟采样调度策略却会产生较为明显的振荡，可以说这是一种不稳定的算法。需要指出的是，式(3.37)中附加的随机噪声w其实是一个很小的扰动，此时的$task(t)$函数及w的概率密度函数分别见图 3.15 及图 3.16，根据3σ定理 99.9%以上的w取值均在$[-0.003,0.003]$区间内，然而即使只附加了这么小的扰动调度算法也会不稳定。

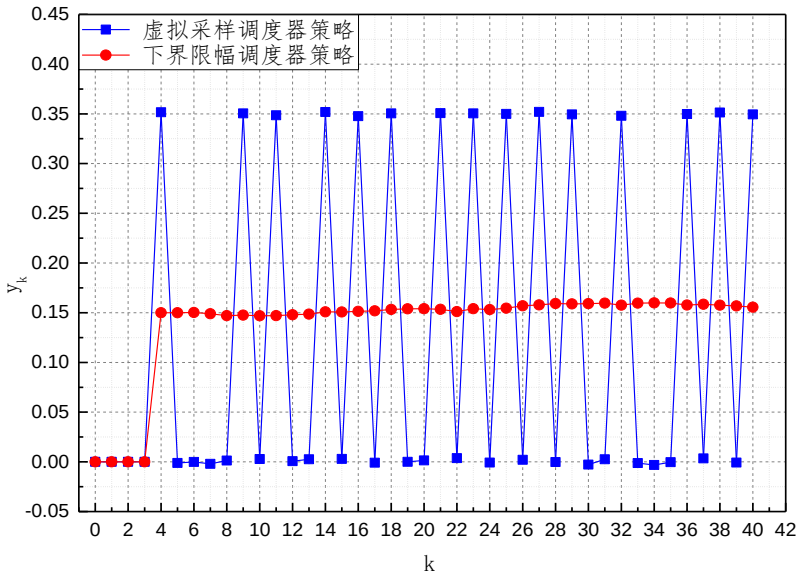

图 3.14 加入噪声扰动后下界限幅与虚拟采样调度策略仿真比较

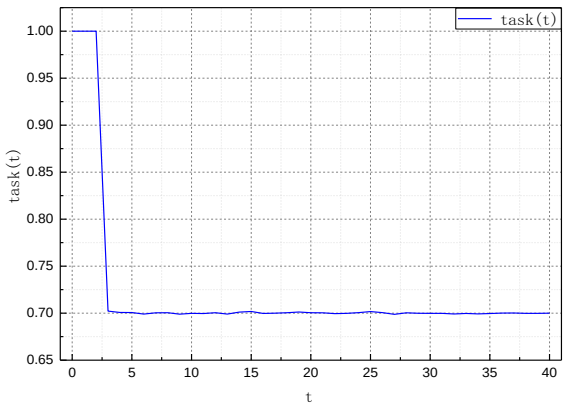

图 3.15 仿真用附加扰动$task(t)$函数

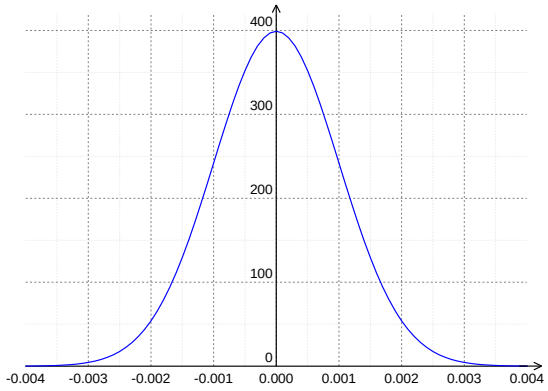

图 3.16 附加扰动的概率密度函数

下面来分析其原因，当$task(t) = A + w$，即常数加一扰动时，系统状态方程为：

$$\begin{bmatrix} M_{k+1} \\ T_{k+1} \end{bmatrix} = \begin{bmatrix} M_k \\ T_k \end{bmatrix} + \begin{bmatrix} A + w \\ A + d_k + w \end{bmatrix} \tag{3.38}$$

$$y_k = [-1 \quad 1] \begin{bmatrix} M_k \\ T_k \end{bmatrix} - \frac{A + w}{2} \tag{3.39}$$

调度器的控制目标是$y_k \to 0$，即

$$\begin{aligned} & T_k - M_k - \frac{A + w}{2} \to 0 \\ \Longrightarrow \quad & T_k \to M_k + \frac{A + w}{2} \end{aligned} \tag{3.40}$$

而调度策略式(3.34)中判断是否执行虚拟采样的依据是

$$\begin{aligned} & T'_k > M_{k+1} + \frac{task(M_{k+1})}{2} \\ \Longrightarrow \quad & T_k > M_k + \frac{A + w}{2} \end{aligned} \tag{3.41}$$

对比式(3.40)与式(3.41)，二者的比较对象都是$M_k + \frac{A+w}{2}$，然而由于w是一随机扰动，就算在稳态时T_k的值也不会严格等于$M_k + \frac{A+w}{2}$，此时式(3.41)的比较条件就很容易成立，此时就会频繁使用调度算法中的虚拟采样策略，而从图 3.12 的原理示意图及图 3.13 的仿真结果中均可看到，此策略其实是以放弃一次采样为代价的，被放弃那一次T_k时刻的采样对应的采样误差y_k就会很大，这就是造成图 3.14 中y_k值大幅振荡的原因。

根据以上分析为提高系统稳定性抑制振荡就要在调度算法中减少频繁的使用虚拟采样策略，一个很自然的想法就是当单次调度偏差或累积调度偏差超过一个阈值后再采用虚拟采样策略进行调整，否则使用下界限幅策略进行控制，本文将此策略称为“混合切换调度策略”。

3.3.3.3 混合切换调度策略

为很好的描述调度偏差及累积调度偏差的影响，可以借鉴 PI 控制器的基本原理来实现这一目的，PI 控制器是经典的 PID 算法去掉微分项后的形式，此处直接给出离散 PI 控制器控制律：

$$u_k = K_P(err_k + \frac{1}{T_I}\sum_{i=0}^{k} err_i) \tag{3.42}$$

为便于实际编程实现，一般使用其增量形式：

$$\Delta u_k = u_k - u_{k-1} = K_P(err_k - err_{k-1}) + \frac{K_P}{T_I} \cdot err_k \tag{3.43}$$

式(3.42)与式(3.43)是完全等价的，仅是形式略有差异。可以看到 PI 控制器中很好的融合了误差及历史误差，而在本系统中也正是需要根据当前调度偏差y_k及历史调度偏差$\sum y_k$的大小来确定采用虚拟采样策略还是简单的对d_k下界限幅，据此可仿照式(3.42)PI 控制器的控制律定义策略切换判断系数SEL（名称取 Select 之意）：

$$SEL_k \triangleq K_P(y_k + \frac{1}{T_I}\sum_{i=0}^{k} y_k) \tag{3.44}$$

并引入判断阈值THR作为策略切换依据，此时就可以得到图 3.17 所示的控制策略。

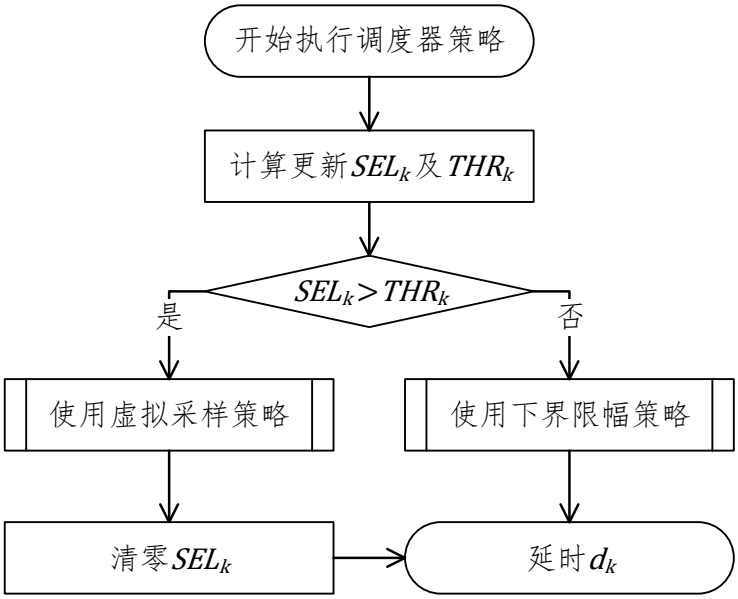

图 3.17 混合切换调度策略流程图

THR作为切换调度策略的判断阈值，其大小会显著影响系统性能，从上文讨论中不难发现调度偏差y_k的相对大小是与$task(t)$的值密切相关的，当$task(t)$较小时，要求的y_k也会越小，否则相对误差就会显得很大；因此在图 3.17 没有将THR视为一个固定常数，而是每次计算SEL_k时都同时计算更新THR_k，计算公式如下：

$$THR_k = \beta \cdot task(T_k) = \beta \cdot (T_k' - T_k) \tag{3.45}$$

THR_k表征的是当前线程执行时间乘以一个收缩系数β。另外由于判断条件是$SEL_k > THR_k$，对比式(3.44)与式(3.45)会发现其实K_P与β间存在冗余耦合关系，不妨将K_P固定为 1，此时SEL_k的计算公式可简化为：

$$SEL_k \triangleq y_k + K_I\sum_{i=0}^{k} y_k \tag{3.46}$$

其中为表达方便，令$K_I = \frac{1}{T_I}$。实际算法中使用增量递推形式进行计算：

$$SEL_k = SEL_{k-1} + (K_I + 1)y_k - y_{k-1} \tag{3.47}$$

另外在图 3.17 的策略中，当使用了虚拟采样策略后会对SEL_k进行清零操作，这是为了避免式(3.46)中累积误差项$\sum y_k$的影响，实际采用的增量递推计算式(3.47)中，对SEL_k清零也就相当于对之前所有历史状态清零。

使用与上一节相同的含噪声扰动的$task(t)$函数进行仿真，仿真结果见图 3.18 与图 3.19，其中$\beta = 0.2, K_I = 0.1$。从仿真结果中可以看到，此混合切换调度策略很好的融合了前文提出的两种调度算法的优点，既不会有稳态静差也基本不会产生振荡。

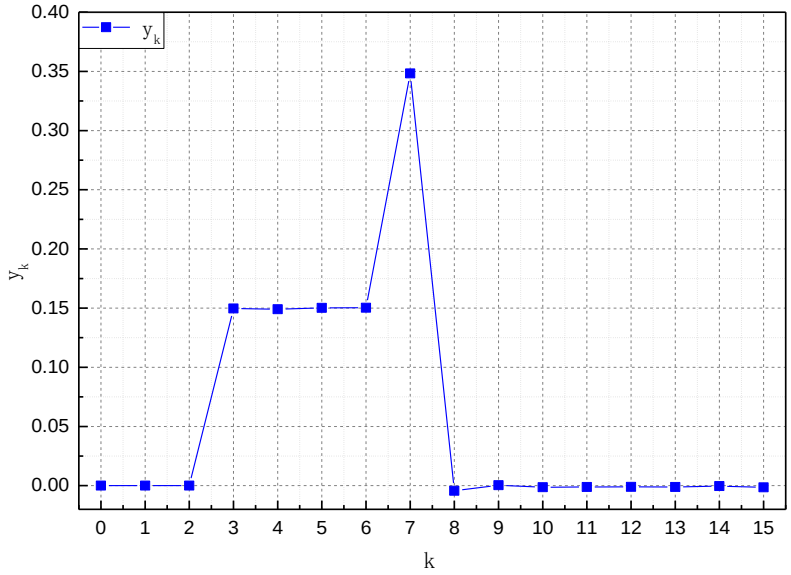

图 3.18 混合切换调度策略仿真结果

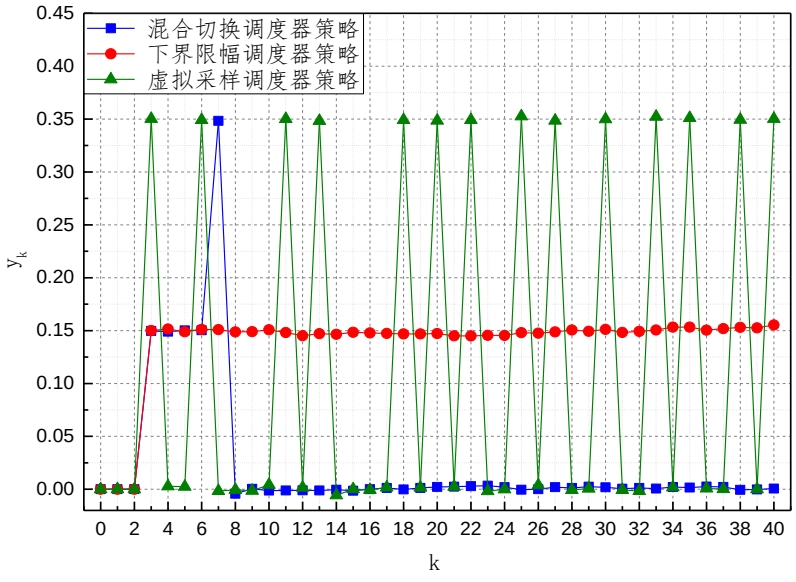

图 3.19 加入噪声扰动后三种调度策略仿真结果对比

此调度策略中存在两个参数，分别是式(3.45)中的β及式(3.47)中的K_I，从原理上分析这两个参数会影响调度算法的稳定性和响应速度，然而仿真及实际测试均中发现这两个参数的影响并不显著，在很宽的取值范围内调度算法性能都很稳定，因此本文研究中就将参数固定为$\beta = 0.2, K_I = 0.1$。

3.3.3.4 不同调度策略性能比较

为更好的比较上述三种调度算法的性能，此处使用一个更复杂的分段函数作为$task(t)$函数：

$$task(t) = \begin{cases} 1 + w & , \ t \in [0,100) \\ 0.7 + w & , \ t \in [100,200) \\ 1.5 + w & , \ t \in [200,300) \\ 0.8 + w & , \ t \in [300,400) \\ 2 + w & , \ t \in [400,500) \\ 1.2 + w & , \ t \in [500,800) \end{cases} \tag{3.48}$$

分别仿真比较扰动$w \sim \mathcal{N}(0, 0.001^2)$、$w \sim \mathcal{N}(0, 0.01^2)$、$w \sim \mathcal{N}(0, 0.05^2)$、$w \sim \mathcal{N}(0, 0.1^2)$时不同调度算法的性能，为方便定量比较引入两个评估指标：$\sum y_k$表示所有调度偏差的累积和；虚拟采样次数$N_{virtual}$表示根据“3.3.3.2 虚拟采样调度策略”一节中提出的虚拟采样方法进行调节的次数。在不同扰动水平下$task(t)$的函数图像见图 3.20，其中作图采样周期为 1；三种调度策略仿真结果对比见表 3.4 与表 3.5。

仿真时间长度设置为 800，由于式(3.48) $task(t)$中包含随机项w，故仿真时取 500 次重复运行的结果平均值作为$\sum y_k$及$N_{virtual}$的期望值。从表 3.4 与表 3.5 的仿真结果中可以看到，下界限幅策略调度偏差$\sum y_k$是最大的，且会随着扰动强度的变大显著增加；虚拟采样调度策略表现较为一致，然而在系统扰动很小的时候其$N_{virtual}$与$\sum y_k$值都较大，并不是一种很好的策略；而混合切换调度策略的确是三种策略中最优的，此策略会根据调度偏差的情况决定是否采用虚拟采样策略，这就使得$N_{virtual}$与扰动强度表现出正相关关系，当外部扰动较小时避免了频繁进行虚拟采样使得$\sum y_k$很小，当外部扰动较大时其性能与单纯的虚拟采样策略相似，很好的融合了其他两种算法的优点。

综上，从对于任务调度算法来说，混合切换调度策略是一种较不错的任务调度方法，通过仿真表明其稳态和动态性能都是令人满意的，本文在嵌入式原型样机上就将采用此策略作为任务调度器。

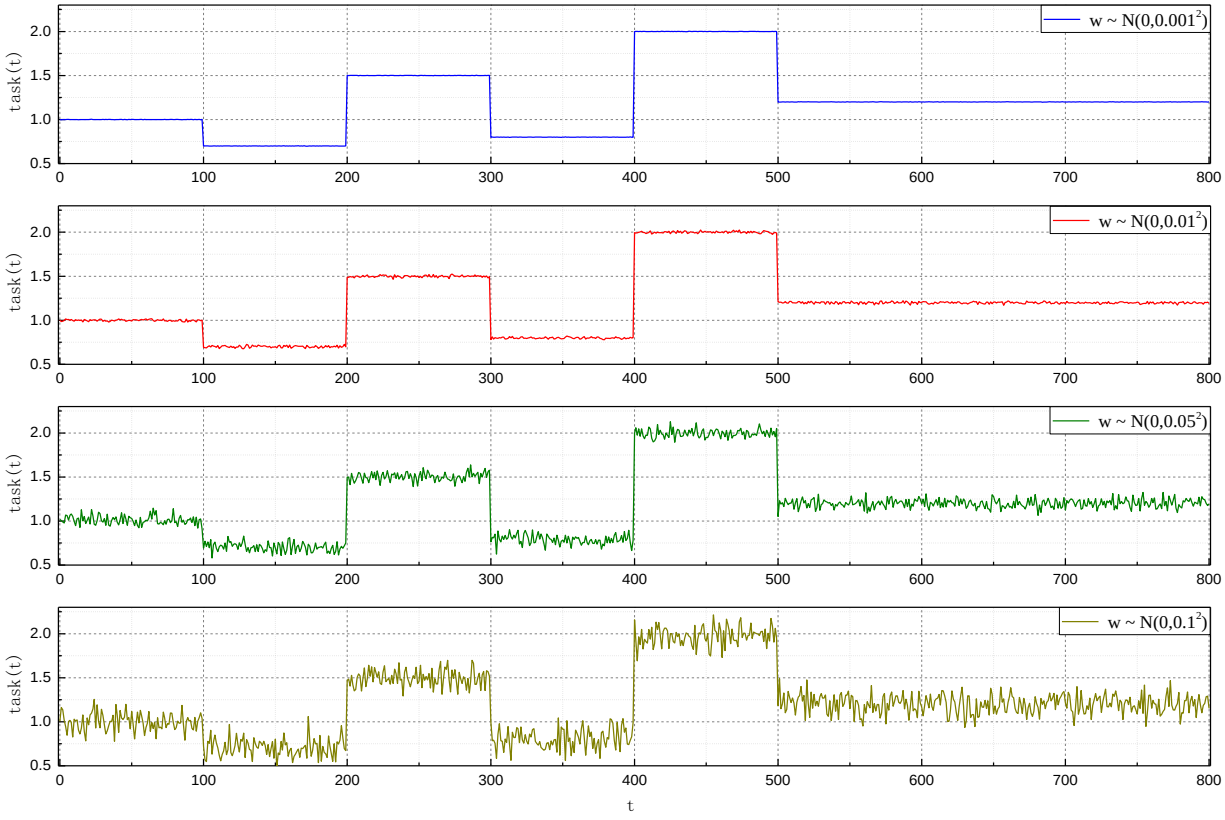

图 3.20 不同扰动水平下$task(t)$函数图像

表 3.4 不同扰动水平下三种调度策略$\sum y_k$对比表

	下界限幅策略	虚拟采样策略	混合切换策略
$w{\sim}\mathcal{N}(0,0.001^2)$	196	140	11
$w{\sim}\mathcal{N}(0,0.01^2)$	281	141	40
$w{\sim}\mathcal{N}(0,0.05^2)$	1003	145	97
$w{\sim}\mathcal{N}(0,0.1^2)$	1888	150	140

表 3.5 不同扰动水平下三种调度策略$N_{virtual}$对比表

	下界限幅策略	虚拟采样策略	混合切换策略
$w{\sim}\mathcal{N}(0,0.001^2)$	0	257	7
$w{\sim}\mathcal{N}(0,0.01^2)$	0	257	26
$w{\sim}\mathcal{N}(0,0.05^2)$	0	257	99
$w{\sim}\mathcal{N}(0,0.1^2)$	0	257	140

3.3.4 参数估计器设计

上一节进行任务调度器设计时将$task(t)$函数视为已知函数，而事实上$task(t)$是未知的，本节的目的就在于构造$task(t)$的估计值。由于在任务调度器中需要使用的只是几个离散点处$task(t)$的取值，因此无需对$task(t)$的完整形式进行估计，只需要对这几个离散点处的值进行估计即可。由于$task(t)$取决于外部输入图像的复杂程度，我们无法对$task(t)$的形式作太多的先验估计，不过一般来说考虑到物体运动的连续性，$task(t)$是不会出现大幅突变的，也就是$task(t)$在大概率上是一个相对平稳的函数，这启示我们可以根据其历史值对其未来值进行估计。

考虑各调度策略中d_k的计算公式（式(3.24)、(3.32)、(3.34)、(3.35)），在计算d_k时需要用到$task(M_{k+1})$，根据主线程开始时间递推关系式(3.4)，有

$$task(M_{k+1}) = M_{k+2} - M_{k+1} \tag{3.49}$$

d_k用于确定T_{k+1}开始时刻，考虑到约束条件$M_{k+1} \le T_{k+1} < M_{k+2}$（参考式(3.10)及图 3.7），此时$M_{k+1}$是已知的而$M_{k+2}$是未知的，故无法通过式(3.49)求出$task(M_{k+1})$。考虑到$task(t)$相对较为平稳，可使用$task(M_k)$作为 $task(M_{k+1})$的估计值，即

$$\widehat{task}(M_{k+1}) = task(M_k) = M_{k+1} - M_k \tag{3.50}$$

其中$\widehat{task}(t)$表示 $task(t)$的估计值。

对于T_{k+1}时刻来说M_{k+1}与M_k都是已知量，故可以使用式(3.50)计算d_k。当$task(t)$阶跃突变时，此方法会造成一个采样周期的延迟，不过对后续稳态情况下没有影响。

考虑到从线程开始时间与结束时间的关系式(3.25)，$task(t)$其实也可以用从线程执行时间进行计算：

$$\widehat{task}(M_{k+1}) = task(T_k^n) = T'^n_k - T_k^n \tag{3.51}$$

由于$task(T_k^n) > task(M_k)$，故使用$task(T_k^n)$作为 $task(M_{k+1})$的估计值会更准确一些，然而每一个从线程结束时间T'^n_k是不确定的，因此可以采用动态更新策略对$\widehat{task}(t)$进行实时更新。任意一个线程（主线程或从线程均可）执行完毕后，根据式(3.50)或式(3.51)计算$\widehat{task}(t)$并更新之前保存的估计值，在计算d_k需要使用$task(M_{k+1})$时直接读取当前最新的$\widehat{task}(t)$作为其估计值即可，此策略无需对各线程的开始结束时间做任何假设，相当便于实际编程实现。

3.4 任务调度策略程序框架的实现及测试验证

3.4.1 POSIX 标准下程序框架的实现

上文完成了任务调度器及参数估计器的设计并对调度器性能进行了仿真研究，本节将给出完整的多线程任务调度策略在 POSIX（Portable Operating System Interface，可移植操作系统接口）标准下实现的程序框架。POSIX 标准是 UNIX、Linux 等操作系统的通用 API（Application Programming Interface，应用程序接口）标准，目前 Windows 操作系统也部分实现了 POSIX 标准，其正式标准名为 ANSI/IEEE 1003.1-1988[91]。在 POSIX 标准多线程模型为 pthreads（POSIX Threads 之意），API 接口包含在头文件<pthread.h>中。

为方便起见主线程与从线程使用同一个线程执行函数，其函数原型为：

```
void * process_frame(void * arg);
```

为通过传入参数 arg 区分主线程与从线程（其调度方法不同），定义以下结构体：

```
typedef struct {
    uint32_t ID;
} thread_data_t;
```

其中 ID 为 0 代表主线程，ID 大于 0 代表第 N 个从线程。

使用宏 NofThread 定义线程总数（包括主线程），示例如下：

```
#define NofThread 2
```

主函数（或其他函数）中可使用如下方法创建各线程：

```
pthread_t tid;
thread_data_t thread_param[NofThread];
for (int i = 0; i < NofThread; i++) {
    thread_param[i].ID = i;
    pthread_create(&tid, NULL, process_frame, (void *)(thread_param + i));
}
```

需要使用几个必要的全局变量来实现线程间通信及历史值的保存：

```
time_t thread_start_time, thread_used_time;
int64_t Thread_err[NofThread] = {0};
int64_t Thread_yk_1[NofThread] = {0};
```

其中 thread_start_time 用于保存主线程开始时间，即上文中的M_k；thread_used_time 就是参数估计器更新的$\widehat{task}(t)$估计值；Thread_err 及 Thread_yk_1 则分别是根据式(3.47)递推计算SEL_k时所需的SEL_{k-1}及y_{k-1}的历史值。

本文多线程任务调度策略的核心算法就是线程执行的函数 process_frame 的设计，此处给出简化版的代码：

```
01 void * process_frame(void * arg) {
02  time_t t1, t2, t_ideal;
03  thread_data_t * param = (thread_data_t *)arg;
04  int64_t yk;
05
06  while(1) {
07      t1 = time(NULL);
08
09      if (param->ID == 0) {
10          // Main Thread
11          thread_start_time = t1;
12      } else {
13          // Slave Threads
14          t_ideal = thread_start_time +
                        thread_used_time / NofThread * param->ID;
15          yk = t1 - t_ideal;
16          Thread_err[param->ID] = Thread_err[param->ID] +
                                    (Ki + 1) * yk - Thread_yk_1[param->ID];
17          Thread_yk_1[param->ID] = yk;
18          if (Thread_err[param->ID] > beta * thread_used_time) {
19              while(t1 > thread_start_time);
20              Thread_err[param->ID] = 0, Thread_yk_1[param->ID] = 0;
21              continue;
22          }
23          sleep_to(t_ideal);
24      }
25      t1 = time(NULL);
26
27      // Place Algorithm Here!
28
29      t2 = time(NULL);
30      thread_used_time = t2 - t1;
31  }
32  return NULL;
33 }
```

代码片段中左侧带底色数字表示行号，此代码与实际程序代码相比进行了一些简化和改写，具体区别有以下三点：

(1) 代码片段中计时使用 time_t 类型及 time()函数，这仅能提供秒级精度，显然无法满足线程调度算法的需求。实际程序中使用的是 timespec 类型及 clock_gettime()函数，此函数可以提供纳秒级精度（取决于硬件及操作系统），然而由于其操作相对较复杂且与算法本身关系不大，此处就将其改写为使用 time()函数代表获取时间这一动作；

(2) 多线程编程中对于线程间共享的全局变量需要考虑互斥与同步问题，为了突出算法本身及减少篇幅，代码片段中删去了这部分代码。实际程序中使用读写锁（Reader-writer lock）对 thread_start_time 及 thread_used_time 进行了保护；

(3) 代码片段第 23 行使用的 sleep_to()函数并不存在，这是一伪函数，其实现的功能为线程休眠至指定时间点，若当前时间超过给定时间点则不进行休眠。实际程序中使用的是与 timespec 时间类型配套使用的 clock_nanosleep()函数，通过将其传入 flags 设置为 TIMER_ABSTIME 实现休眠至某一给定时间点。

尽管上述代码片段与实际程序不完全相同，不过其对任务调度算法代码实现方式的描述是完整的，下面对其中一些核心代码的作用进行说明。第 9 行判断是主线程还是从线程，若是主线程更新 thread_start_time 变量，此变量表示当前最新一次主线程开始时间 M_k；若是从线程根据第 14 行代码根据理想调度算法计算从线程应该开始的时间；15~17 行根据式(3.47)计算SEL_k，其中 Ki 及 beta 这两个参数使用宏定义常数实现；18 行就是混合切换策略（见上文图 3.17 所示流程图）的判断条件，若判断需要采用虚拟采样调度策略，则 19 行的循环会跳过本次采样等待 thread_start_time 更新（相当于等待主线程开始新一次采样），之后再进行调度；23 行就是下界限幅调度策略的实现，若给定时间已经晚于现在的时间则不会进行休眠延时；25~29 行放置算法的主体部分并对其执行时间计时；30 行更新 thread_used_time，这就是参数估计器的实现方法。

3.4.2 嵌入式原型样机上测试验证及性能评估

为实际验证本章提出的任务调度策略有效性，我们在嵌入式原型样机上实现了此算法，使用多个线程并行执行本文第二章提出的交通标志定位算法检测红色圆形禁令标志，并记录比较不同从线程数量下采样时间间隔T_{sample}的差别，为保证结果的一致性进行测试时摄像头固定拍摄同一个位置，这样可以保证交通标志定位算法执行时间基本保持恒定。由于嵌入式原型样机只有 4 个 CPU 核心，根据本章 3.2 节对线程数量的约束最多只

能存在 3 个从线程，因此对从线程数为 0~3 的情况进行测试，记录 60s 内每个线程开始采样的时间点，通过做差即可求出T_{sample}，所得结果见表 3.6 及图 3.21，其中图 3.21 为了保证图示的美观性仅截取了 10s 区间内的结果进行展示。从表 3.6 中可以看到随着从线程数的增加T_{sample}的平均值、最小值、最大值均逐渐降低（由于随机误差，$N=1$与$N=0$时的最大值可视为基本相同），这表明采用多线程技术对于减小采样间隔T_{sample}提高系统实时性有很大的帮助；表中也可以看到$N=1$时的标准差最大，这是由于当只有一个从线程时由于虚拟采样调度策略的存在采样间隔可能会出现较大波动，这一点可以从图 3.21 中直观的看到。

表 3.6 不同从线程数量下T_{sample}测试结果统计表

	平均值(ms)	标准差(ms)	最小值(ms)	最大值(ms)
$N=0$	361	10	338	373
$N=1$	202	73	79	376
$N=2$	126	37	25	351
$N=3$	101	36	14	300

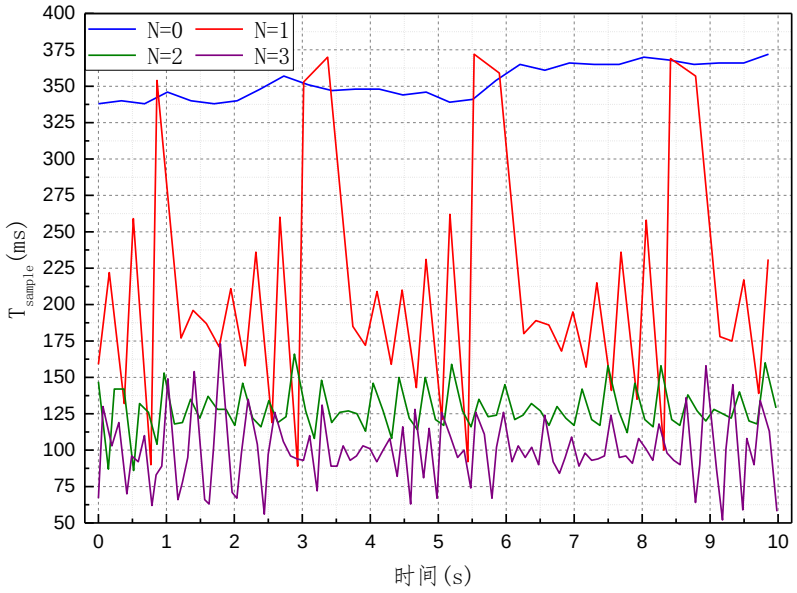

图 3.21 不同从线程数量下T_{sample}测试结果图（截取 10s 区间）

本章第一节中已经指出只用T_{sample}的最大值$T_{sample(max)}$衡量系统实时性存在一定缺陷，需要从统计分析的角度研究T_{sample}的分布情况，不同从线程数量下T_{sample}的统计直方图见图 3.22~图 3.25，对比这几张T_{sample}的统计直方图可以很明显的看到随着线程数量的增加，使用本章提出的多线程任务调度策略使得T_{sample}的分布整体向较小值方向移动，这进一步表明多线程技术可以有效的提高系统的实时性。

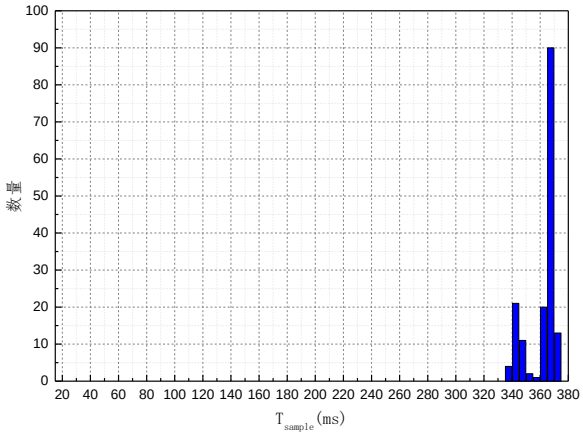

图 3.22 $N=0$时T_{sample}统计直方图

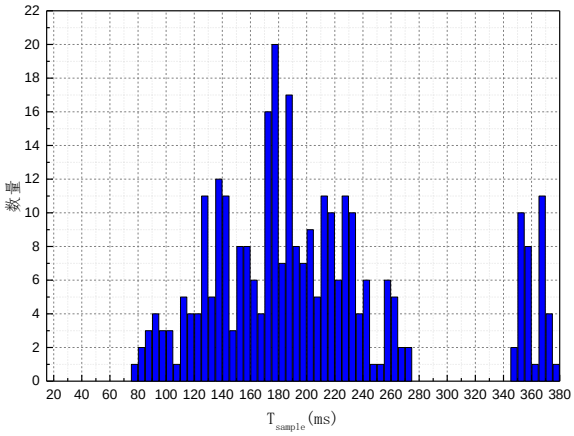

图 3.23 $N=1$时T_{sample}统计直方图

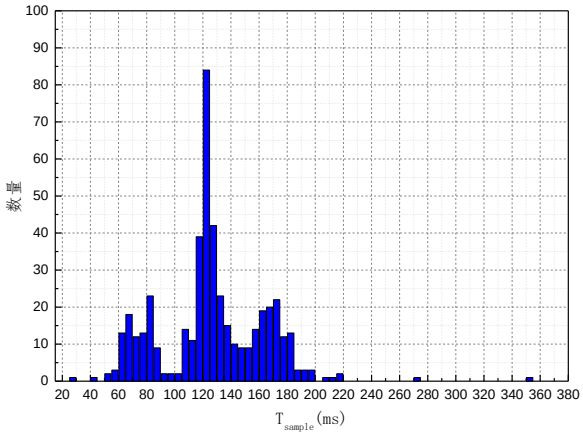

图 3.24 $N=2$时T_{sample}统计直方图

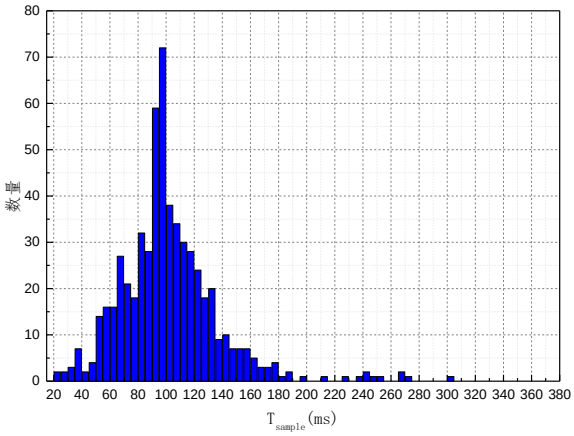

图 3.25 $N=3$时T_{sample}统计直方图

上文测试中T_{sample}的缩短其实主要得益于线程数的增加，以上测试并没有能很好的反应出本章提出的任务调度策略的优势，因此需要进一步与不使用任务调度（即同时创建多个线程并行执行，不对其时序关系做控制）下系统采样间隔时间进行对比。考虑$N=1$及$N=3$这两个最有代表性的情况，不进行任务调度时序控制情况下T_{sample}的统计直方图见图 3.26 及图 3.27。

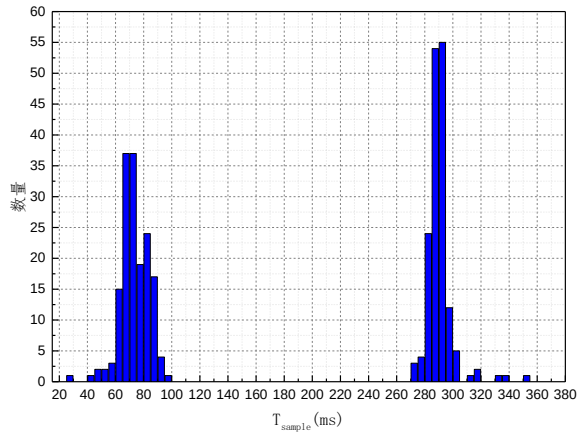

图 3.26 $N=1$时T_{sample}统计直方图（不采用任务调度策略）

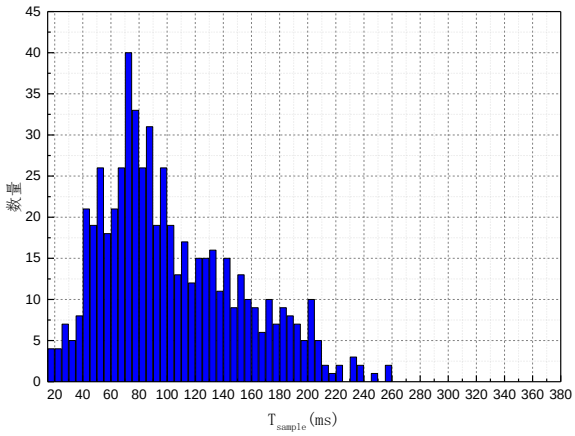

图 3.27 $N=3$时T_{sample}统计直方图（不采用任务调度策略）

对比图 3.23 及图 3.26 可以很明显的看到在不采用任务调度时下T_{sample}的分布极为不均匀，这是由于主线程与从线程开始采样的时间点距离太近导致的，在采用了多线程任务调度策略后很好的解决了这一问题；不过在线程数较多时，图 3.25 与图 3.27 间差异并不明显；为进一步更好的对比是否采用任务调度策略的区别可从另一角度进行考虑。正如本章第一节的分析中提到的，提高系统实时性的核心要求就是尽量缩短T_{sample}，若$T_{sample} > T_{sample(limit)}$对于交通标志识别来说就有可能造成漏检，故我们可以用T_{sample}的累积概率曲线来更好的衡量系统实时性，而统计直方图其实就是概率密度曲线在小样本下的表现形式，因此可用归一化后的累积频率曲线来作为累积概率曲线的近似，结果见图 3.28 及图 3.29。

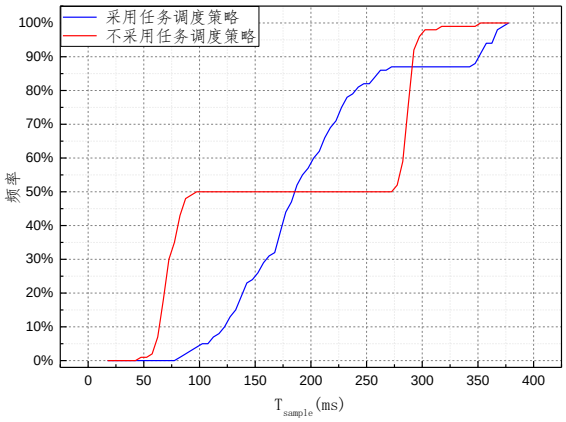

图 3.28 $N=1$时T_{sample}累积频率曲线

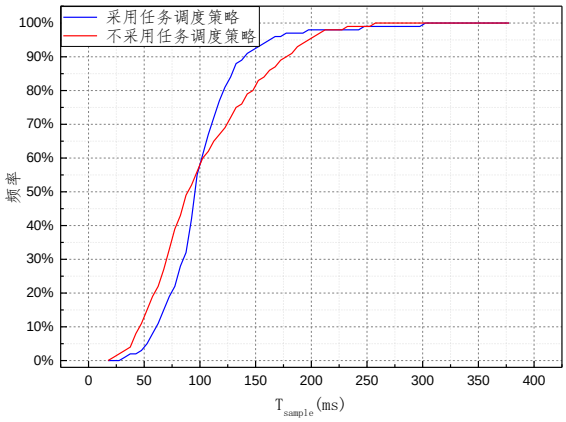

图 3.29 $N=3$时T_{sample}累积频率曲线

从图中可以看到当$N=1$时在采用任务调度策略的情况下 80%的$T_{sample}<250\text{ms}$，而在不采用任务调度策略的情况下 80%的$T_{sample}<290\text{ms}$；当$N=3$时在采用任务调度策略的情况下 80%的$T_{sample}<120\text{ms}$，而在不采用任务调度策略的情况下 80%的$T_{sample}<150\text{ms}$。由此可见采用本章的任务调度策略与不进行调度控制相比的确获得了更佳的系统实时性。

不过图 3.29 中两条曲线差距不大，且图 3.27 不采用任务调度策略的统计直方图中T_{sample}也没有表现出分布不均匀，这与理论期望结果有所偏差，造成这一现象的原因是由于摄像头采集图像需要时间，实际测试表明原型样机采集 1920*1080 的图像约需 80ms，且连续采集与随机采集所需时间并不相同。为保证线程安全程序中对采集图像操作使用互斥锁（Mutex）进行了保护，任何时候只有一个线程能进行图像采集，其余线程均会进入等待状态。线程开始时间是从申请互斥量算起的，此时各线程的执行时间就会不同，因此几个线程间自然会错开一段时间；而完成处理任务所需时间也只有 370ms 左右，理想线程调度算法给出的$T_{sample}\approx 90\text{ms}$与摄像头图像采集时间很接近，这就使得不使用任务调度策略自然形成的T_{sample}分布也很接近理想分布。为排除这一因素的干扰，将采集图像改为读取已有图片进行处理，并重复运行 10 次交通标志定位算法以延长处理任务所需时间，此时T_{sample}统计直方图见图 3.30~图 3.33，累积频率曲线见图 3.34 及图 3.35。

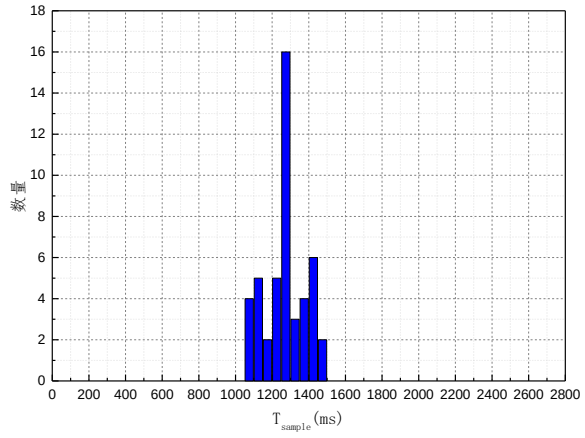

图 3.30 $N=1$处理已有图片时T_{sample}统计直方图

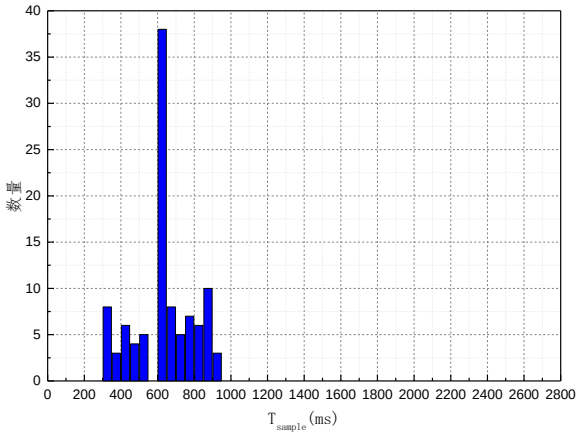

图 3.31 $N=3$处理已有图片时T_{sample}统计直方图

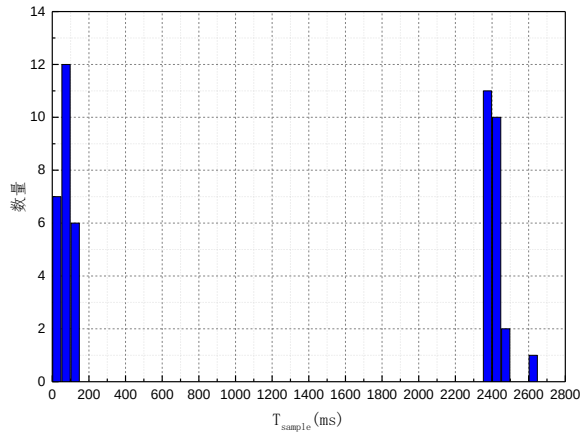

图 3.32 $N=1$处理已有图片时T_{sample}统计直方图（不采用任务调度策略）

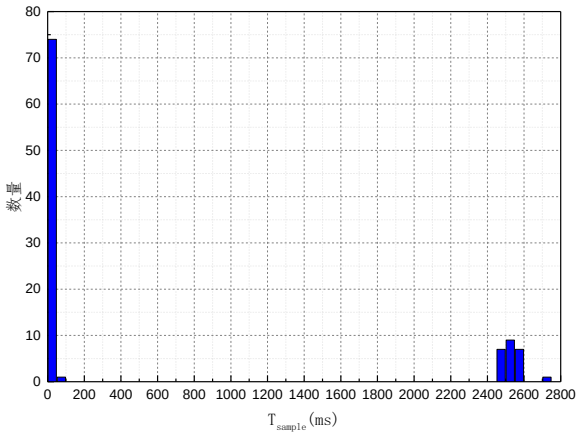

图 3.33 $N=3$处理已有图片时T_{sample}统计直方图（不采用任务调度策略）

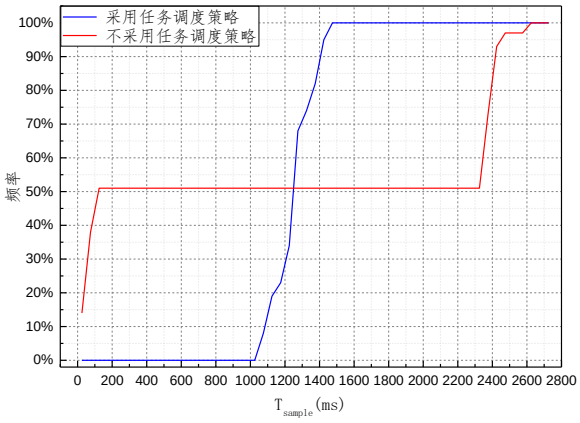

图 3.34 $N=1$处理已有图片时T_{sample}累积频率曲线

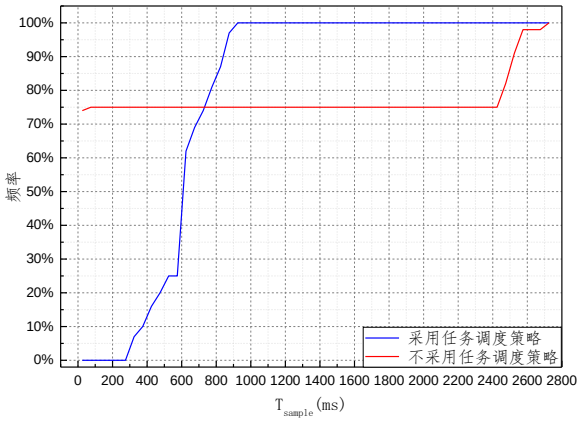

图 3.35 $N=3$处理已有图片时T_{sample}累积频率曲线

从以上各图中可以明显看到在处理已有图片时使用多线程任务调度策略可以极大的降低$T_{sample(max)}$进而提高系统实时性，当$N=1$时在采用任务调度策略的情况下 80%的$T_{sample}<1350\text{ms}$，而在不采用任务调度策略的情况下 80%的$T_{sample}<2400\text{ms}$；当$N=3$时在采用任务调度策略的情况下 80%的$T_{sample}<750\text{ms}$，而在不采用任务调度策略的情况下 80%的$T_{sample}<2450\text{ms}$。

综上所述，本节的测试验证充分说明了本章提出的多线程任务调度策略能有效的缩短采样间隔时间，特别是当执行的算法所需时间相对较长时采用此调度策略的优势更为明显。

3.5 本章小结

本章首先对交通标志检测识别系统的实时性要求进行了分析，指出可以用两次采样的间隔时间作为此类系统是否满足实时性约束及实时性好坏的定量判断条件。在此基础上对此问题进行建模分析，提出了基于模型的理想任务调度算法且证明了此算法是本问题的最优解，针对理想任务调度算法实际不能实现的问题，将此系统抽象为一控制系统，对其控制器（即任务调度器）与参数估计器分别进行设计与仿真验证。在核心的任务调度器设计上分析比较了下界限幅调度策略与虚拟采样调度策略这两种调度策略，针对其各自的优缺点提出了一种混合切换调度策略，并通过仿真验证了此策略的有效性；参数估计器的设计上使用了一种高效的易于编程实现的动态更新策略。最后本章给出了前文提出的混合切换调度策略及动态更新参数估计器在 POSIX 标准下的程序框架实现，并在嵌入式原型样机上实际测试验证了此多线程任务调度策略，通过多个维度的对比分析充分说明了此任务调度策略能够有效的缩短采样时间间隔进而提高系统实时性。本章研究的多线程任务调度策略不仅适用于交通标志识别，对于类似的检测识别类应用均有一定参考价值。

4 仿真验证平台及嵌入式原型的搭建与系统集成测试

本文第二、三两章分别对基于颜色分割的交通标志定位算法及混合切换多线程任务调度策略进行了论述，其中已经涉及了在仿真验证平台及嵌入式原型样机上进行的一些测试验证，本章将对相关平台的详细情况、搭建过程与软硬件设计方法进行具体介绍，并将在实际道路上对嵌入式原型样机进行集成测试，以此验证本文的圆形禁令标志标志定位算法及混合切换多线程任务调度策略在实际嵌入式系统与真实道路环境中的有效性。

4.1 平台介绍及核心元件的选取

本文涉及的平台一共有 3 个，在第二章中对数据集中的视频片段进行处理以比较各算法效果时使用的平台是基于 Windows 系统的，GUI（Graphic User Interface, 用户图形界面）前端界面采用 Qt 实现，下文将其简称为算法验证平台；在第三章中对各任务调度算法进行仿真比较时使用的平台是基于 Linux 系统的，使用科学计算软件 Octave 完成，此外一些性能基准测试，如 2.2.3 、2.3.3 节中的算法所需时间测试，3.2 节中线程数量与执行时间关系测试等均在此平台上完成，下文将此平台称为性能测试平台；最后一个平台是嵌入式原型样机，基于 Intel Joule 开发板与 Linux 系统，下文简称为原型样机。

4.1.1 算法验证平台及性能测试平台介绍

这两个平台使用的都是成熟的商用硬件，算法验证平台是一台 ThinkPad 笔记本电脑，性能测试平台是自行组装的服务器，均不涉及平台本身的搭建问题，直接在其上进行软件开发即可，此处仅列出其基本硬件配置、操作系统及 IDE 等以供参考，见表 4.1 及表 4.2。

表 4.1 算法验证平台配置表

项目	配置信息
型号	ThinkPad T440s
CPU	Intel Core i7-4600U @ 2.10GHz (TurboBoost 最高频率 2.69GHz)
内存	12.0GB DDR3L-1600MHz
操作系统	Windows 10 64 位专业版 Build 15063.786
IDE	Microsoft Visual Studio Community 2015

表 4.2 性能测试平台配置表

项目	配置信息
CPU	双路 Intel Xeon E5-2660 @ 2.20GHz
内存	32.0GB DDR3-1333MHz ECC REG 服务器内存
操作系统	Ubuntu Server 14.04.5 LTS (Kernel version: 4.4.0)
C/C++编译器	GCC 4.8.4

4.1.2 原型样机平台选型

目前主流嵌入式平台按 CPU 架构可分为 x86 平台、ARM 平台、MIPS 平台等，其中 ARM 平台最为普遍。由于目前基本所有的 CPU 及 SDRAM 芯片均采用 BGA 或其他无引脚封装，自行设计硬件电路 PCB 打样及焊接成本较高，且还涉及 Bootloader、操作系统的移植及 BSP（Board Support Package，板级支持包）的开发，工作量较大；为快速搭建原型样机以验证本文算法，选用成熟的嵌入式单板计算机是一个更佳的选择。

4.1.2.1 主流嵌入式平台选型比较

目前最常用的嵌入式单板计算机是由英国树莓派基金会开发的树莓派（Raspberry Pi），由于其价格低廉易于使用得到了全世界许多高校及爱好者的支持，目前其最新版本为 2016 年 2 月发布的 Raspberry Pi 3 Model B（不计随后于 2017 年发布的计算模块及低价版本 Zero），其外观图片见图 4.1。树莓派是 ARM 架构单板计算机的代表，不过除 ARM 架构外 Intel 也推出了多款基于 x86-64 架构的嵌入式开发板，如 Edison、Baytrail、Galileo、Curie 等，其中发布于 2016 年 9 月的 Joule 平台在体积与性能上都较合适车载安全预警系统使用，其外观图片见图 4.2。

图 4.1 Raspberry Pi 3 Model B 外观图

图 4.2 Intel Joule 570x 外观图

本节将对这两款单板计算机进行比较测试，从而选型确定原型样机的嵌入式平台，这两款开发板的基本参数对比见表 4.3，从中可以看到 Intel Joule 平台在各项性能指标上均明显优于树莓派。

表 4.3 Raspberry Pi 3 Model B 与 Intel Joule 570x 参数对比表

项目	Raspberry Pi 3 Model B	Intel Joule 570x
CPU 架构	ARMv8-A (RISC)	x86-64 (CISC)
CPU 核心	ARM Cortex-A53	Intel Goldmont
CPU 型号	Broadcom BCM2837	Intel Atom T5700
CPU 主频	1.20GHz	1.70GHz (TurboBoost to 2.40GHz)
CPU 核心数	4	4
GPU 主频	400MHz	450MHz (Turbo to 650MHz)
SDRAM	1.0GB	4.0GB
eMMC	无（使用 TF 卡）	16.0GB
视频输出	HDMI 1.3	HDMI 1.4b
USB 接口	4 * USB2.0	1 * USB3.0
无线网络	802.11 n	802.11 n / ac
蓝牙	Bluetooth 4.1	Bluetooth 4.2

为更准确的定量衡量这两个平台的计算性能，使用 sysbench 这款 Linux 下常用的开源基准性能测试软件[92]对其进行测试，使用的 sysbench 版本为 1.1.0-08d4b38，由源码直接编译得到。测试命令为：

```
sysbench --test=cpu --num-threads=1 --cpu-max-prime=20000 run
```

此命令使用单线程计算小于 20000 的质数数量，可用于评估系统的单核整数运算能力。为同时对比嵌入式平台与通用计算机的性能，在上文所述性能测试平台及腾讯 1 核 1GB 云服务器上同时使用相同版本的 sysbench 进行测试。测试结果见图 4.3，其中 CPU 速度单位为 event/s (eps)，其值越大代表 CPU 运算速度越快。从测试结果中可以看到 Intel Joule 开发板的性能大幅优于树莓派，约是其 14 倍左右，而根据表 4.3 二者的 CPU 主频并没有这么大差距，实际性能的差异更多来自于 CPU 架构和指令集；从测试结果中还可

以看到 Intel Joule 平台的性能与通用计算机相比仍有一些差距，约为性能测试平台的 44%左右。对于车载安全预警系统而言，需要处理来自摄像头的图像并作出决策判断，其计算量相对较大，故原型样机选择 Intel Joule 平台更为合适。

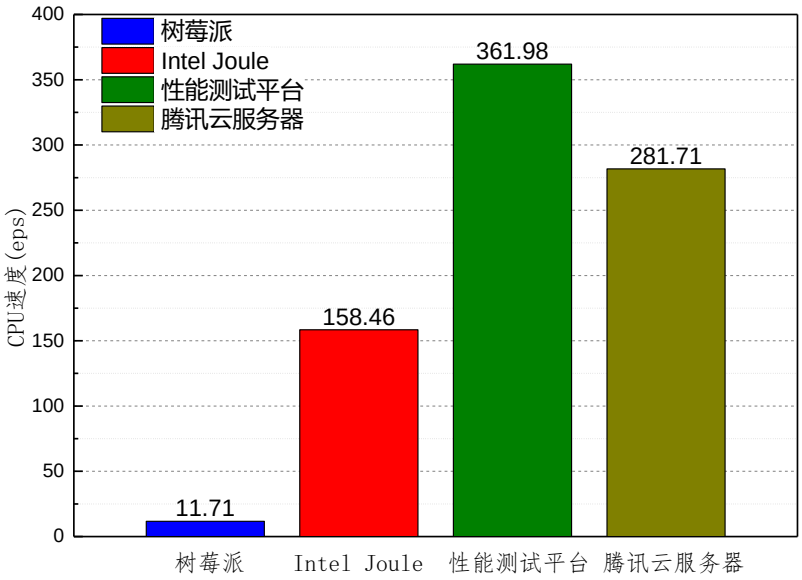

图 4.3 各平台 CPU 基准性能测试对比图

4.1.2.2 Intel Joule 平台介绍

上一节中通过参数对比及实际性能测试选择了采用 Intel Joule 平台搭建系统原型样机，此处对此平台的基本情况做一简单介绍，更详细的说明可参考其数据手册[93]。Intel 将 Joule 模块称为 SoM（System on Module），可以理解为一个模块中集成了整个完整系统，上文图 4.2 的外观图其实是 Joule 模块与其扩展板结合在一起的形态，独立的 Juole 模块是其中银色部分，大小仅 24mm*48mm，其中集成了 CPU、4GB LPDDR4 SDRAM、16GB eMMC，已经构成了一台完整计算机的最小系统。Joule 模块通过两个 100 pin 的高速板对板连接器与扩展板连接，扩展板起到引出各接口（USB、HDMI 等）并为 Joule 模块提供 5V、4V 及 3.3V 稳压电源的作用。

Joule 模块的 CPU 采用 Goldmont 架构的 Atom T5700 处理器，其基本参数在表 4.3 中已给出，这款 CPU 是 Intel 专门为嵌入式领域设计的，在保证其相对较高性能的同时将其 TDP（Thermal Design Power，热设计功耗）降低至 4W 左右。Atom T5700 支持 SSE、SSE2、SSE3、SSE4、SSSE3 等矢量指令集，不过缺乏对 AVX、AVX2 及 FMA3 矢量指令集的支持，后续开发中在编译 OpenCV 时需要根据 CPU 指令集支持情况进行针对性优

化。Joule 模块的视频输出采用 Micro-HDMI 接口，支持 HDMI 1.4 标准的 4k 高清输出，原型样机中无需这么高的分辨率，使用了一块 7 英寸 800*480 分辨的液晶屏作为显示屏。

4.1.3 摄像头的选取及介绍

由于本文第二章所做的研究是基于图像的，对于原型样机来说在确定了其基本平台后最重要外部元件就是摄像头，Joule 模块同时支持 USB 摄像头及 MIPI CSI-2 接口摄像头，不过由于 MIPI 接口摄像头一般是集成于板级设计中（如手机摄像头），独立摄像头模块基本都是 USB 接口的，故原型样机设计中也采用了成熟的 USB 摄像头。通过比较最终选择了深圳金宏达视电子科技公司的 KS2A17 摄像头模块，这是一款基于 OV2710 COMS 传感器芯片的 USB 2.0 摄像头模块，尺寸为 38mm * 38mm，可方便的更换不同焦距的镜头，其外观图片见图 4.4。

图 4.4 KS2A17 摄像头模块外观图

OV2710 芯片是 OmniVision 公司推出的一款全高清（1080P）COMS 传感器[94]，其核心参数见表 4.4，本文第二章建立中国道路交通标志公开数据集时采用的视频采集设备参数见前文表 2.1，其中使用的 COMS 传感器芯片型号为 OV4689，此处同时列出其核心参数[95]作为对比，见表 4.5。对比 OV2710 及 OV4689 的技术参数可以看到，OV4689 的输出分辨率及帧率更高，不过在感光元件尺寸（场面尺寸及像素尺寸）、信噪比、动态范围、光照敏感度等指标上 OV2710 均更好一些，则说明其成像质量将优于 OV4689；事实上 OV2710 的这些指标在小型消费级 COMS 传感器中属于较为优秀的，这有助于获得噪点更少质量更高的图片，很适合车载安全预警系统原型样机使用。

表 4.4 OV2710 COMS 传感器芯片核心参数表

项目	参数信息
最高输出分辨率	1920 * 1080
最高输出帧率	30fps @ 1080P, 60fps @ 720P, 120fps @ VGA, 240fps @ QVGA
场面尺寸(Lens size)	1/2.7″
像素尺寸(Pixel size)	3μm * 3μm
最大信噪比(S/N)	40dB
动态范围	69dB @ 8x gain
光照敏感度	3700mv/lux-sec

表 4.5 OV4689 COMS 传感器芯片核心参数表

项目	参数信息
最高输出分辨率	2688 * 1520
最高输出帧率	90fps @ 1520P, 120fps @ 1080P, 180fps @ 720P, 330fps @ 380P
场面尺寸(Lens size)	1/3.0″
像素尺寸(Pixel size)	2μm * 2μm
最大信噪比(S/N)	38.3dB
动态范围	64.6dB @ 1x gain
光照敏感度	1900mv/lux-sec

4.2 系统软硬件开发核心技术介绍

本节将依次对算法验证平台及原型样机设计开发过程中的一些核心步骤及技术进行阐述，并对摄像头最佳输出分辨率进行分析优化。

4.2.1 算法验证平台软件开发技术

如前文所述，算法验证平台是基于笔记本电脑 Windows 系统下的，其作用在于使用给定的图像处理算法对数据集中的视频或图片进行处理以此评估算法效果，基于 Windows 的程序更便于调试，有助于快速验证算法的有效性及调整算法参数等，因此本研究中选择了在 Windows 下开发算法验证平台软件，本文第二章中的所有处理结果图也都由算法验证平台软件产生。此软件同样以开源形式发布，可从以下 Github 链接处获取：

https://github.com/g199209/TSR

软件基于 OpenCV 3.2 及 Qt 5.7 开发，IDE 环境为 Microsoft Visual Studio Community 2015；Qt 与 VS 环境的集成采用 QtPackage 插件实现[96]；Qt 与 OpenCV 配合使用时图片的交互相对较为复杂，需要实现 cv::Mat 与 QImage 类之间数据的转换。程序中通过继承 QWidget 类实现了一个自定义 Qt 控件 ImageViewer，其中封装了转换函数 QPixmap cvMatToQPixmap(const cv::Mat &inMat)，具体实现方法详见源代码。

为保证 GUI 主窗体能实时响应用户操作，不能在主线程中进行长时间耗时操作以避免阻塞 UI 主线程，因此程序中采用了异步非阻塞机制来保证程序的可用性，下面将对此机制的实现方法做一简要说明。图像处理算法被封装为一个独立的 TSR 类，继承自 QThread 类，这是 Qt 中多线程机制平台无关的抽象封装，TSR 类会作为一个独立的线程持续运行，这就保证了图像处理算法的执行不会阻塞 UI 主线程；在 TSR 类中通过 Qt 经典的信号-槽（Signals & Slots）机制实现异步通知，信号槽机制是 Qt 处理事件的核心机制，基本思想就是设计模式中的观察者模式，在此机制下通过 connect()函数捆绑后调用信号函数会自动触发槽函数，以此实现线程间异步事件同步的目的，其通用抽象原理示意图及本软件中的实际实现分别见图 4.5 及图 4.6。在完成图像处理后 TSR 类的实例通过调用 ImageReady 信号通知主线程（即 MainWin 类，主窗口的实现）的 UpdateImage 槽；MainWin:: UpdateImage()函数通过读取保存在全局变量中的算法处理结果更新显示在主界面上的状态（包括图像及文本等）。

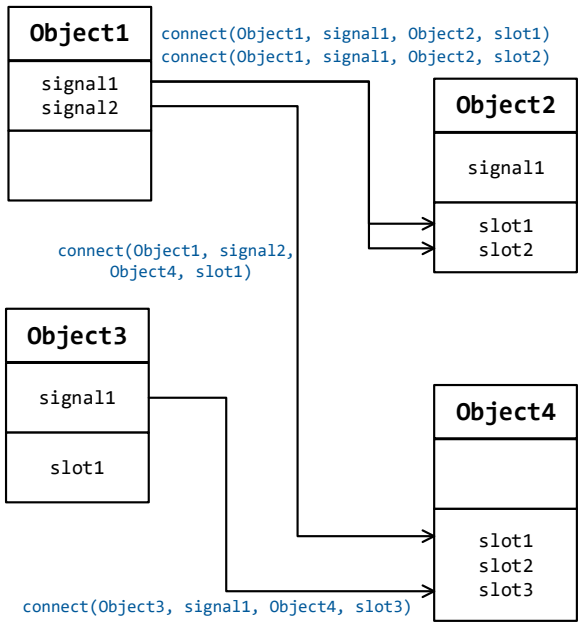

图 4.5 信号槽机制通用原理示意图

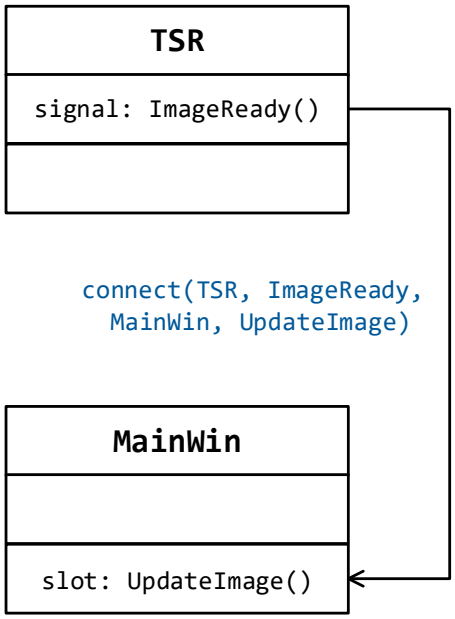

图 4.6 算法验证软件中信号槽示意图

以上流程机制实现了 TSR 类完成图像处理算法后异步更新主界面的目的，而 UI 主界面也需要在检测到用户特定的操作事件（如点击开始处理按钮）后通知 TSR 类开始执行图像处理算法。TSR 类内部使用了状态机来监听这一事件并管理整个算法处理流程，其状态转移图见图 4.7，其中 ReadImg 状态表示读取待处理图像，这是整个算法处理流程的开始；Step1~Stepn 这若干个状态代表算法处理的若干步骤；Update 状态发送上文提到 ImageReady 信号以更新主界面显示；Idle 代表空闲状态，此状态下会延时一段时间并持续停留在 Idle 状态下，不会主动切换到其它状态。

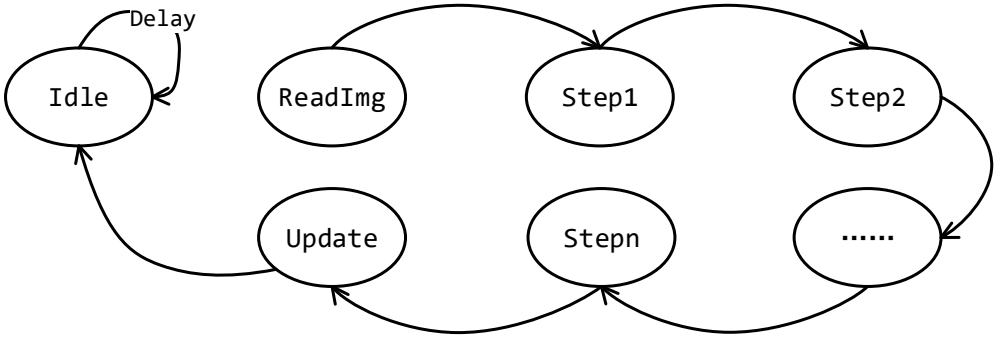

图 4.7 算法验证软件 TSR 类内部状态转移图

表示当前状态的枚举变量封装在一个名为 TSRParam_t 的结构体类型中，此结构体内还包含一些算法参数(如 2.4.2 节中 Hough 圆变换所需的r_{min}, r_{max}, $Dist_{min}$, Th_{Hough}等)；当 UI 主线程检测到用户操作事件需要开始一次新的算法处理流程后会修改一个 TSRParam_t 类型的全局变量，将其中的当前状态改为 ReadImg，并写入其余算法参数；TSR 类内部在按照图 4.7 每进行一次状态切换前都会先更新读取此全局变量，通过此机制 UI 主线程可以强制重置或触发算法处理流程，一次实现异步通信的目的。图 4.7 的状态转移图中将算法主体部分分拆成了很多子状态（Step1~Stepn），这样设计的目的是为了提高 UI 界面响应速度；由于程序中只会在完成某一状态切换到下一状态前更新 TSRParam_t 全局变量，若所有算法均对应一个状态 TSR 线程在执行处理算法时将不会响应 UI 主线程重置当前状态为 ReadImg 的请求，这就会导致用户界面响应的延迟；将耗时较长的处理算法合理分拆为多个子状态可以很好的解决这一问题，在子状态间发生状态转移时均会更新 TSRParam_t 全局变量，若 UI 主线程对其进行了修改则会放弃此次处理重新由 ReadImg 状态开始执行新一次处理流程。需要说明的是，实际程序代码中需

要采用互斥锁来保证对 TSRParam_t 全局变量的读写操作不会发生线程冲突。

以上线程同步通信机制是算法验证平台软件的核心技术，通过此机制不但保证了 GUI 主窗体不会没有响应，也很好的实现了 UI 界面框架与算法代码的分离与解耦，使得程序整体结构更为清晰也更易于添加新功能，保证了后续的可维护性。诸如界面设计、保存读取配置文件、打开保存图片视频文件等其余功能的实现方法较为常规此处不再说明，可参考实际源代码。

本节最后给出算法验证平台软件界面截图，见图 4.8，其中 1 号区域为菜单栏；2 号区域为视频播放控制区域；3 号区域算法选择及参数设置面板，通过多个层叠面板可对处理流程各步骤采用的算法进行选择并设置其参数，此界面设计具有很好的扩展柔性；4 号区域为底部状态栏，可显示当前处理的是视频中的哪一帧；5 号区域为处理结果的显示界面，可同时灵活显示多张图片。

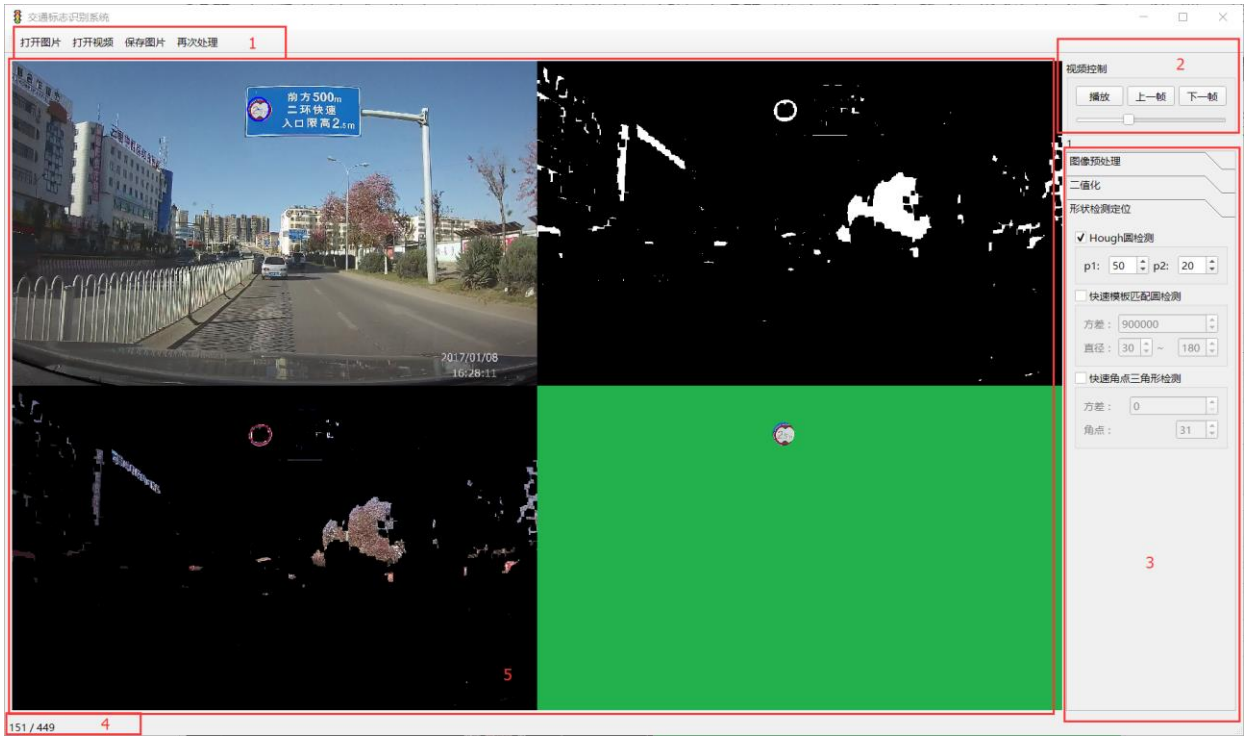

图 4.8 算法验证平台软件界面截图

4.2.2 原型样机硬件结构设计

本文选用的嵌入式原型样机平台为 Intel Joule 模块，配合其扩展板后硬件部分已基本完整无需再另行设计硬件电路，仅需要通过扩展接口连接各模块组件即可，结构框图见图 4.9，由于 Joule 模块本身只有一个 USB 接口，需要使用一个 USB Hub 对其进行扩展；键盘仅在调试时需要连接使用；红色的连接线代表供电线，原型样机由 7.4V 2S 锂

电池组供电；散热风扇用于为 Joule 核心板提供良好的散热，根据 Intel Joule 模块热设计相关文档[97]在 25℃环境温度下安装散热风扇最大功率可达到 7W 左右，若不安装散热风扇最大功率仅能达到 2.5W 左右，此时会对 CPU 核心频率进行限制进而影响系统性能。

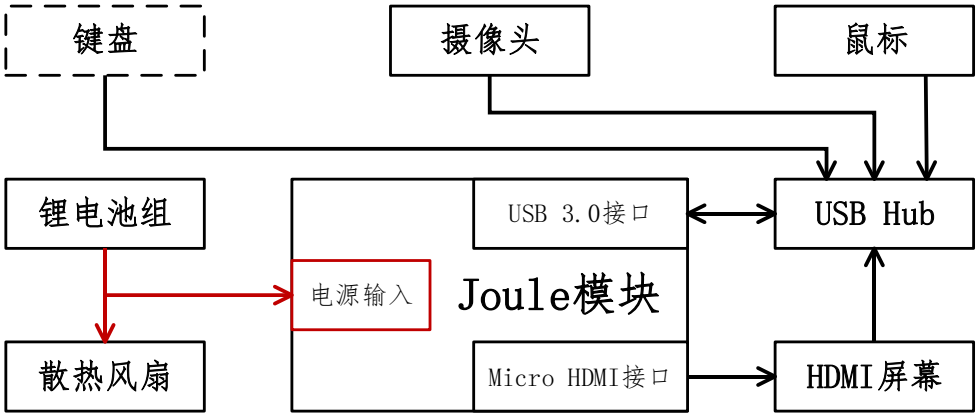

图 4.9 原型样机结构框图

从图 4.9 的系统结构框图中可以看到整个原型样机由多个独立模块组成，为很好的将这些模块组合到一起形成一台完整的原型样机需要设计一个合适的固定结构，此处采用机械设计软件 Solidworks 设计了一块底板，设计图及 3D 效果渲染图分别见图 4.10 及图 4.11，其中凸起的位置用于使用铜柱固定 Joule 扩展板及 HDMI 液晶显示屏；下方凸台用于固定散热风扇；右上方则用于固定 USB Hub。

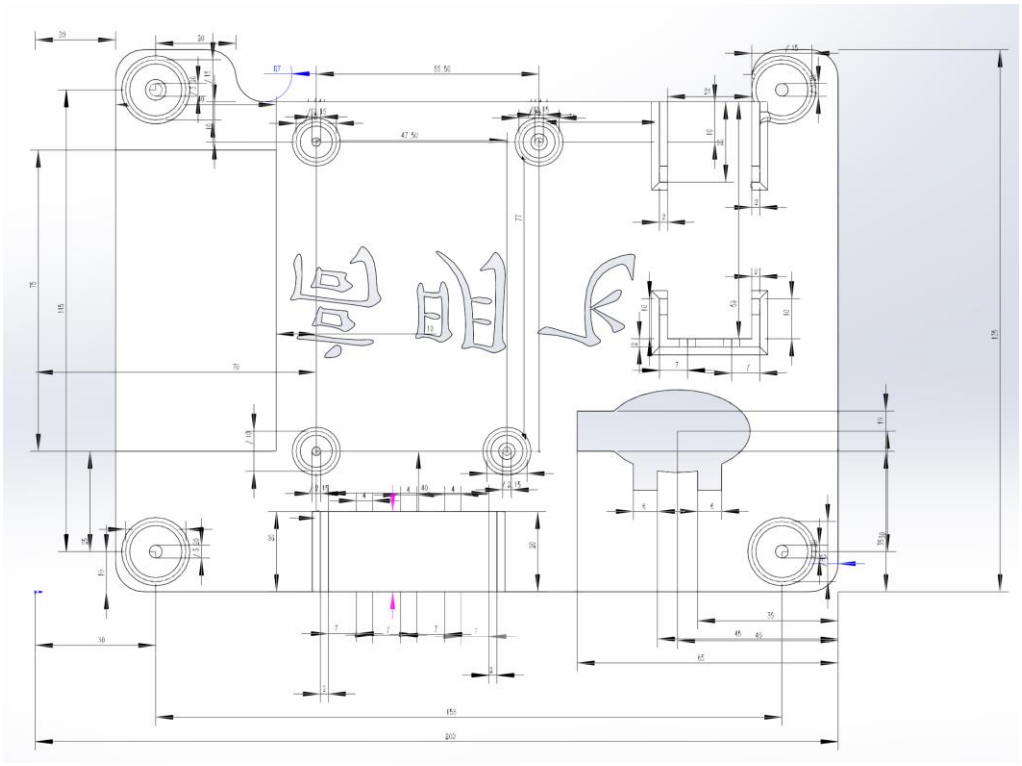

图 4.10 原型样机底板设计图（俯视图）

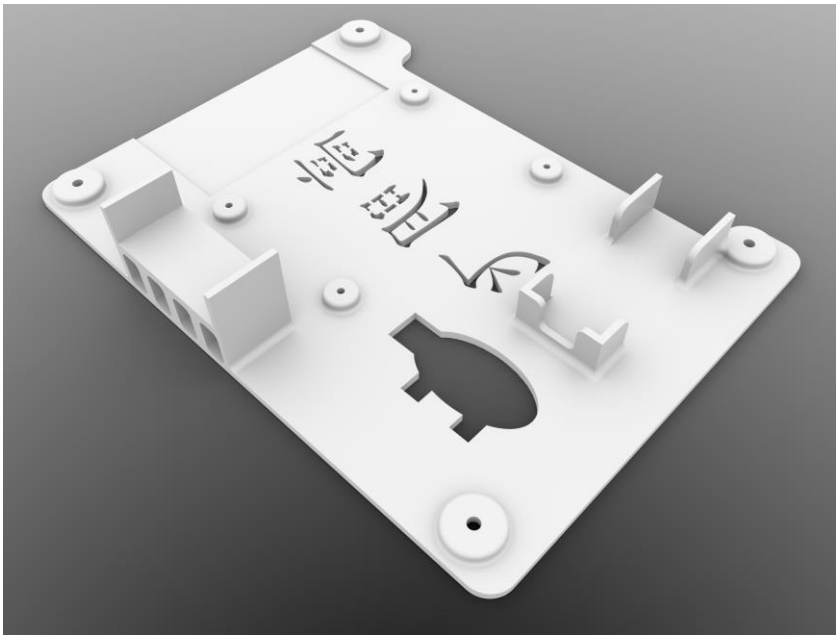

图 4.11 原型样机底板 3D 效果渲染图

考虑到 3D 打印具有成型速度快、加工精度高等诸多优点，我们选用了 SLA（StereoLithography Apparatus，光固化成型） 3D 打印技术来加工此底板，所用材料为 DSM（帝斯曼）公司的 Somos Imagine 8000 低粘度光固化树脂，此材料加工得到的产品表面光滑度较好，性能类似于常用的 ABS 工程塑料。组装中间过程及组装完成后的照片分别见图 4.12 及图 4.13，其中 USB Hub、散热风扇与图中左下角的控制开关及电源接口使用 506 快速固化胶粘合于底板上；Joule 扩展板与液晶屏间的 HDMI 连接使用的是定制的 FPC（Flexible Printed Circuit，柔性电路板）软排线，与标准线缆相比其长度更短且占用空间更小。

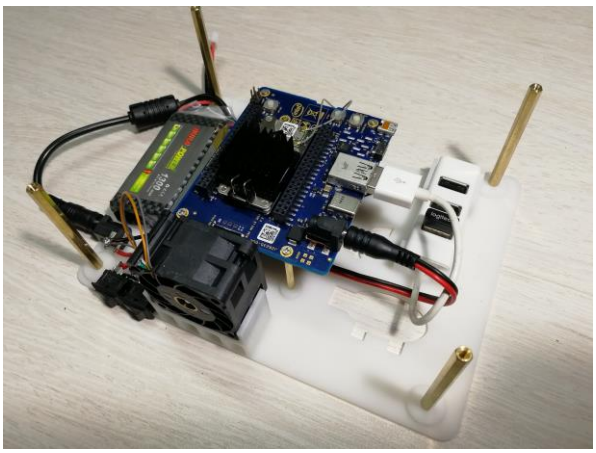

图 4.12 原型样机组装过程照片

图 4.13 原型样机成品照片

4.2.3 原型样机软件开发技术

Joule 模块所用的 Atom T5700 CPU 是基于 Intel 经典的 x86-64 架构的，理论上说任何可在 PC 机上运行的系统均可在此模块上运行，然而考虑到其属于嵌入式系统，性能与通用 PC 相比仍有一定差距（参考图 4.3 的对比结果），更好的选择是采用专门为嵌入式应用场景优化的操作系统。目前主流的嵌入式操作系统基本都是基于 Linux 内核的，Intel 公司针对 Joule 模块提供了 2 个不同的 Linux 发行版可供选择，分别是 Ubuntu-Core 16.04 及 Ref-IoT-OS，其中 Ubuntu-Core 是 Ubuntu 系统针对嵌入式物联网应用的优化版本；Ref-IoT-OS 则是 Intel 基于 Yocto 项目[98]开发的一个定制版 Linux 系统。然而实际测试表明 Ubuntu-Core 16.04 不包含 Joule 模块的无线网卡驱动，且其基本开发环境不全均需要自行编译安装；而 Ref-IoT-OS 由于是 Intel 专门为物联网开发制作的操作系统，其驱动与开发环境集成都十分完整，故最终选择了使用 Ref-IoT-OS 作为基础操作系统，采用的发行版本为 1.0+snapshot-20170316，Linux 内核版本为 4.4.41。

Ref-IoT-OS 中未包含 OpenCV 库，需要自行由源码编译安装，这里使用了 OpenCV 3.2，OpenCV 中部分代码可使用矢量指令集进行优化，在编译 OpenCV 的过程中通过向 CMake 传入配置参数来指定使用那些矢量指令集，Atom T5700 仅支持 SSE 系列矢量指令集而缺乏对 AVX 等高级指令集的支持，因此 CMake 配置命令为：

```
cmake -D WITH_IPP=ON -D WITH_TBB=OFF -D BUILD_TBB=ON -D WITH_CUDA=OFF -D
WITH_OPENCL=OFF -D ENABLE_AVX=OFF -D ENABLE_AVX2=OFF -D ENABLE_FMA3=OFF -D
ENABLE_POPCNT=ON -D ENABLE_SSE=ON -D ENABLE_SSE2=ON -D ENABLE_SSE3=ON -D
ENABLE_SSE41=ON -D ENABLE_SSE42=ON -D ENABLE_SSSE3=ON ../
```

由于 Atom T5700 与性能测试平台同为 Linux 系统及 x86-64 架构，无需使用嵌入式开发中常用的交叉编译器，直接在性能测试平台上采用本地 GCC 编译器进行编译即可，编译得到的程序在两个平台上均可正常运行。根据惯例同时支持了 Debug 调试模式及 Release 发布模式的编译生成，与 VS 平台不同，在 GCC 及 CMake 中没有默认的 Debug 及 Release 模式的配置，其区别仅在于编译参数不同，Debug 模式下优化级别较低且附加了较多调试参数（如生成调试信息、生成性能分析测试代码等），Release 模式下优化级别较高且不生成任何调试信息，在正式使用中均采用 Release 模式。

Debug 模式核心编译参数为：

```
-pthread -O0 -Wall -g -ggdb -pg
```

Release 模式核心编译参数为：

```
-pthread -O3 -DNDEBUG
```

Ref-IoT-OS 采用了 Xfce 桌面环境，这是一款基于 GTK+图形库的轻量级桌面环境，因此系统中自带 GTK+库，这是目前唯一一个主要使用 C 语言开发的主流 GUI 库；由于编译 Qt 库相对较为复杂，此处选用了 GTK+ 3.0 作为原型样机 GUI 界面的基础图形库。原型样机上所需的用户界面与算法验证平台相比要简单得多，考虑到正常使用中无需进行用户交互只需展示出实时检测结果即可，因此整个界面仅包含一个可容纳图像的控件并全屏显示。软件设计的核心在于 OpenCV 中 Mat 类图片需要先转换为 GTK+中使用的 GdkPixbuf 结构体之后才能正常显示出来，此转换可通过 gdk_pixbuf_new_from_data()函数实现。原型样机上 GUI 界面运行时效果见图 4.14，界面上显示的是摄像头实时拍摄得到的图片及检出结果，由于嵌入式系统中不方便截图，此处使用的是实际拍摄的照片。

图 4.14 原型样机上 GUI 界面照片

4.2.4 摄像头最佳输出分辨率分析优化

KS2A17 摄像头模块最高可输出 1920*1080 分辨率的视频，不过也可通过配置其 ISP 降低输出分辨率。较高的输出分辨率下显然更有利于交通标志的检测定位好，然而也会造成处理时间过长反而影响系统实时性；高分辨率的意义在于其图像更为精细，可用交通标志区域所占像素面积来衡量，不妨将其用A_{sign}表示，当A_{sign}大于某阈值时方能检出目标交通标志，因此可以通过测试不同分辨率及不同距离下A_{sign}的值来确定摄像头最佳输出分辨率。为进行此测试选择了浙江大学玉泉校区内一条长直道路，其卫星图片见图 4.15，其中 A、B 两点存在固定交通标志，1~8 为测试点，测试点与交通标志见距离见表 4.6。在每个测试点处拍摄若干张不同分辨率的图片，通过计算图片中 A、B 两个交通标志所占像素面积后求平均值即可得到某距离某分辨率下的A_{sign}值，结果见图 4.16 及图 4.17，图中使用指数函数进行了曲线拟合。

图 4.15 不同分辨率测试采样地点卫星地图

表 4.6 不同分辨率测试中采样点与交通标志间距离表

距离(m)	1	2	3	4	5	6	7	8
A	108	68	45	20	6.5	\	\	\
B	180	139	116	91.5	78	48	35.5	9

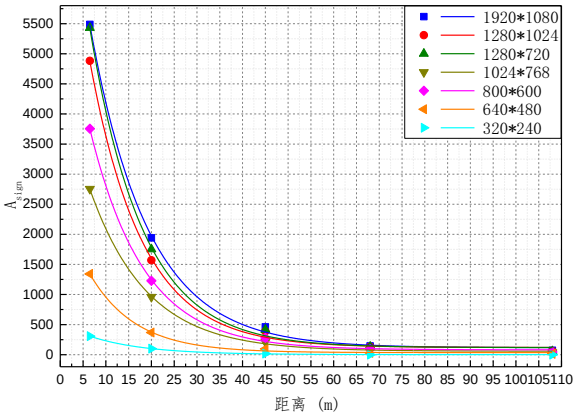

图 4.16 A 点交通标志A_{sign}与距离关系图

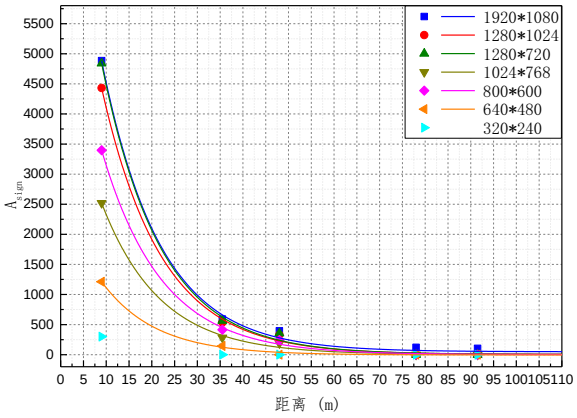

图 4.17 B 点交通标志A_{sign}与距离关系图

通过在拍摄得到的图片上运行本文的交通标志定位算法，当$A_{sign} > 2000$时方能正确定位目标交通标志；图 4.16 与图 4.17 表现出了较好的一致性，根据图中的拟合曲线可确定在各分辨率下$A_{sign} = 2000$对应的识别距离，见表 4.7，640*480 及 320*240 分辨率下 5m 距离内依然无法正常识别，因此下文将不再讨论这两个分辨率。

表 4.7 不同分辨率下$A_{sign}=2000$对应的识别距离表

分辨率	1920*1080	1280*1024	1280*720	1024*768	800*600
识别距离(m)	20	17	19	12	15

其实表 4.7 给出的距离就是“3.1 系统实时性要求分析”一节中定义的有效识别采样区域的长度，按照中国大部分城市道路最高限速 80km/h（高速公路下限速可达到 120km/h，然而高速公路上的交通标志也更大）可计算出对应的最长采样间隔时间$T_{sample(limit)}$，进而结合测试得到的各分辨率下算法执行所需时间可计算出在有效识别采样区域内有效采样帧数，结果见表 4.8。从中可以看到分辨率为 800*600 时有效采样帧数最多，此分辨率下系统可取得最好的实时性，因此摄像头的最佳输出分辨率为 800*600。仔细分析图 4.16 与图 4.17，由于所得曲线近似为指数函数，在距离较远时A_{sign}的变化其实很小，表 4.7 中也可以看到 1920*1080 与 800*600 相比有效识别采样区域仅仅提高了 5m，而根据表 4.8 其算法处理时间却是 800*600 的 4 倍左右，因此较高的分辨率（如 1920*1080）并没有明显的优势。

表 4.8 不同分辨率下$T_{sample(limit)}$、算法所需时间及有效帧数对照表

分辨率	1920*1080	1280*1024	1280*720	1024*768	800*600
$T_{sample(limit)}$(ms)	900	765	855	540	675
处理时间(ms)	301	207	146	119	80
有效帧数(帧)	3.0	3.7	5.9	4.5	8.4

另一个比较有意思的结论是，1280*1024 是高于 1280*720 的，其算法处理时间也要更长，然而其识别距离却反而更短，这是由于不同分辨率下图像范围其实是不相同的，可计算A_{sign}占总像素面积A_{image}的比例，以测试点 4 处拍摄的 A 点交通标志为例计算结果见表 4.9，从中可以看到 800*600 分辨率下交通标志占比是最大的，而 1280*1024 分辨率的交通标志占比是低于 1280*720 的，这也从另一个角度论证了 800*600 为何是最佳分辨率。

表 4.9 不同分辨率下交通标志区域面积占图片总面积比例表

分辨率	1920*1080	1280*1024	1280*720	1024*768	800*600
A_{sign}/A_{image}	0.94 ‰	1.20 ‰	1.90 ‰	1.22 ‰	2.56 ‰

4.3 真实道路环境系统集成测试

本文第二、三章中在论述基于颜色分割的交通标志定位算法及混合切换多线程任务调度策略时已经通过仿真分别对其进行了单元测试验证，本节将对完整的算法在原型样机上实现效果进行集成测试。

4.3.1 原型样机优化配置条件

为保证算法在嵌入式系统上的实时性及检测定位效果，嵌入式原型样机的实现中进行了一些针对性优化，具体包括两点：

(1) 根据“2.5 算法在数据集上测试验证”一节中的结果，在逆光情况下若采用摄像头默认的动态测光算法很有可能导致交通标志区域过暗进而影响颜色分割效果，为解决此问题此处采用了曝光锁定策略，根据天气光照情况确定此时合适的曝光值并固定其值不变，这样就可以在一定程度上抑制天空、太阳等过亮区域的影响；不过目前是通过人为手动调节确定合适曝光值的，此方法缺乏通用性，针对不同天气光照情况均需要手动调整；后续研究中可以考虑采用点测光策略自动根据路面区域亮度确定曝光值。

(2) 根据“4.2.4 摄像头最佳输出分辨率分析优化”一节中的分析，当摄像头输出分辨率取 800*600 时有效识别距离仅为 15m 左右，此结论是在采用 4mm 广角镜头进行测试时得到的，为延长有效识别距离原型样机中将镜头更换为了 12mm 长焦镜头，此时有效识别距离在天气光照良好时可提升至 40m 左右；然而采用长焦镜头也有严重副作用，长焦镜头下视角区域很窄，手持测试时尚可通过将摄像头对准交通标志来解决此问题，然而实际应用中车载安全预警系统是固定在车辆上的，若摄像头可视角度太窄很有可能造成漏检，故后续研究中还需要对此问题进行进一步研究评估确定更合适的镜头焦距。

另外由于原型样机的摄像头型号及分辨率与数据集中的不同，Hough 圆检测中部分算法参数需要进行调整，具体见表 4.10，表中未提及的参数与上文介绍的相同。

表 4.10 实际原型样机集成测试中 Hough 圆检测算法参数取值表

r_{min}	r_{max}	$Dist_{min}$	Th_{Hough}
7	300	150	22

最后考虑到 Intel Joule 模块采用的 Atom T5700 CPU 一共有 4 个核心，为取得最佳的 CPU 利用率一共创建了 4 个图像处理线程，此外实际程序中还存在 GUI 线程、Log 统计记录线程等其它几个线程，此时在系统运行过程中 CPU 及内存使用情况见图 4.18（此图是通过 SSH 方式远程登录到原型样机上运行 htop 资源监测工具得到的），其中"/home/root/Desktop/ADAS"即为交通标志检测程序，在 htop 的输出中每一项代表一个线程或进程，从中可以看到在系统运行过程中 CPU 利用率已经达到了很高的水平。

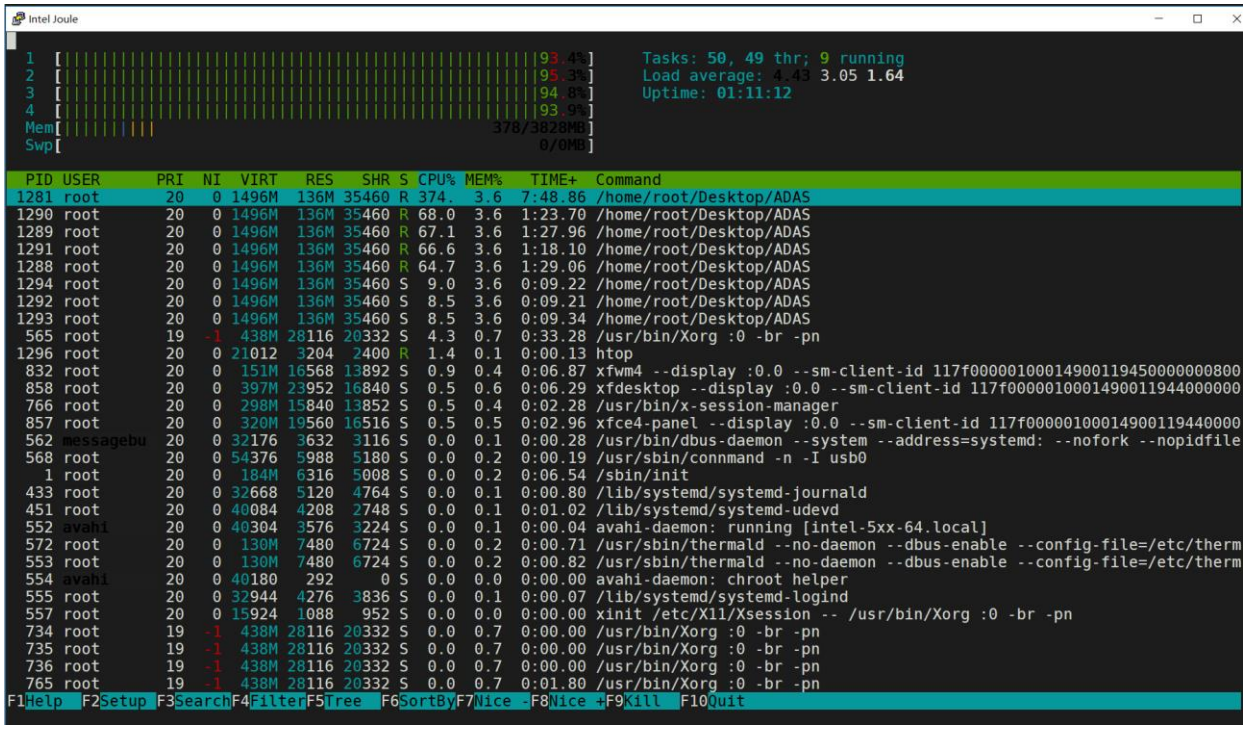

图 4.18 原型样机运行过程中 CPU 及内存使用情况图

4.3.2 校园环境静态测试

浙江大学玉泉校区内存在大量交通标志，因此选择在校园道路上内对原型样机上交通标志定位算法的检出率进行测试，重点评估在天气光照良好且运动速度较慢时的检测效果。测试时间为下午 15 点左右，天气晴且光照良好，一共在 25 个点处进行了测试，测试点分布见图 4.19，由于交通标志具有聚集性，25 个测试点处一共存在 55 个红色圆形禁令标志；测试时手持原型样机及摄像头从正面以正常速度走过交通标志，由于人的运动速度很慢故将此测试称为静态测试；在这一过程中若能正确定位交通标志即认为成功检出。

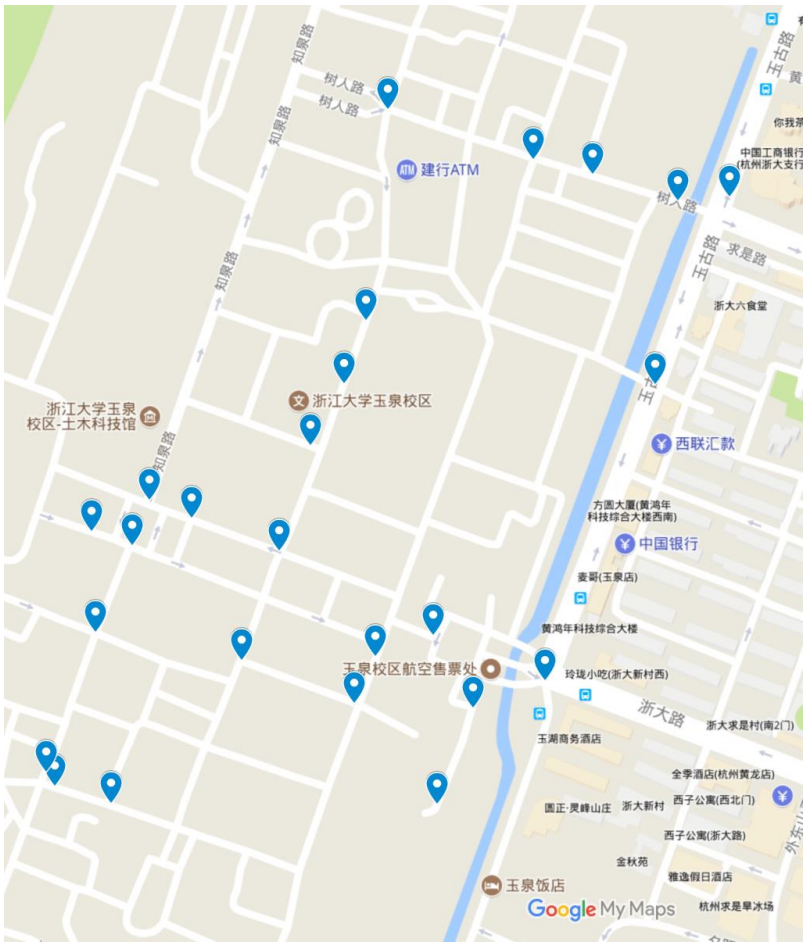

图 4.19 校园环境静态测试中测试点分布图

某测试点的实际照片及原型样机检出结果见图 4.20 及图 4.21，由于在太阳光下原型样机屏幕反光严重图 4.21 拍摄效果不佳，不过仍能看出此时正确检出了两个交通标志的存在；由于右侧禁止冲坡标志角度有所倾斜，Hough 圆检测作出的圆形没有和交通标志本身完全重合。对 55 个交通标志的测试结果表明，本文的方法可以成功检出其中 50 个标志，检出率为 91%相对较高，这是由于测试时天气光照较为理想且在步行状态下能够灵活调整摄像头位置的缘故。未能检出的标志可分为两类，一类是标志褪色过于严重且红色边缘不完整，此类标志会导致颜色分割方法失效进而无法检出；另一类主要由于距离太远导致无法检出，此类标志主要是位于玉古路（参考图 4.19）上的几个交通标志，出于安全考虑无法手持设备从正面靠近位于机动车道上方的交通标志。

图 4.20 静态测试中某测试点照片

图 4.21 静态测试中某测试点检出结果

测试过程中也对天气光照良好时系统的最远检出距离进行了估计，对于图 4.22 所示几个交通标志，拍照点处基本就是系统的检出极限距离，此时交通标志及测试点位置见图 4.23 所示卫星照片，在地图上可量得二者间的距离约为 45m 左右。

图 4.22 静态测试中最远检出距离照片

图 4.23 静态测试中最远检出距离卫星图

不过需要指出的是，尽管天气良好时静态检出率较为理想，然而也存在一定的误检情况，如在图 4.24 及图 4.25 中共享单车的红色车轮会被检测为交通标志，考虑到在实际完整的系统中检测定位仅仅只是交通标志识别的第一步，后续还会对检出的区域进行模式分类判断具体是哪种交通标志，一定限度的误检是可以通过后续分类过程加以解决的，因此在交通标志检测定位环节检出率比漏检率更为重要。

图 4.24 静态测试中误检情况测试点照片

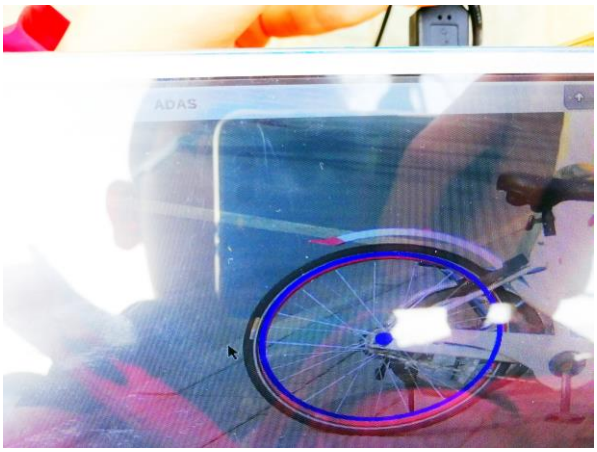

图 4.25 静态测试中误检情况检出结果

4.3.3 城市道路动态测试

以上校园环境静态测试主要用于评估系统在天气光照较为理想且图像稳定时的最高检出率，测试结果较为理想，可取得 91%的检出率，然而实际车载安全预警系统是固定安装于车辆上的，车辆运动速度较快时图像会存在抖动、模糊等情况，且天气光照也不可能随时都是很理想的状态；为评估实际应用中系统的表现需要在实际车辆上进行测试，此测试将重点评估在车辆运动过程中及天气光照不佳时的检测效果，由于车辆处于运动过程中故将此测试称为动态测试。为便于进行测试使用泡沫制作了一个简易固定装置，且同时将摄像头使用铜柱固定于原型样机上，通过此简易装置可方便的通过手持将原型样机固定于车辆上，此简易固定装置及测试过程中的照片见图 4.26 及图 4.27。

图 4.26 原型样机简易固定装置图

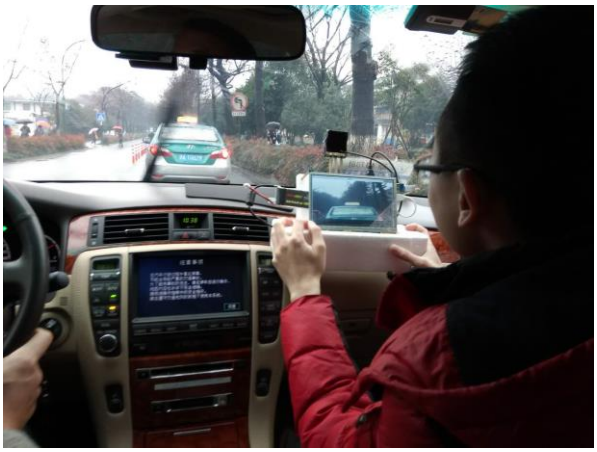

图 4.27 进行动态测试过程中照片

城市道路动态测试时间为早上 10 点~11 点之间，天气状况为小雨，此时雨刮、玻璃上的雨滴及反光等均会造成干扰，这对检测定位算法的鲁棒性提出了较高的要求，此测试可以反映本文算法在实际车载情况及天气不佳时的表现。测试路段为浙大玉泉校区至龙井村往返共计 15km 左右的城市道路，具体路线见图 4.28，行车路线为 A->B->C。

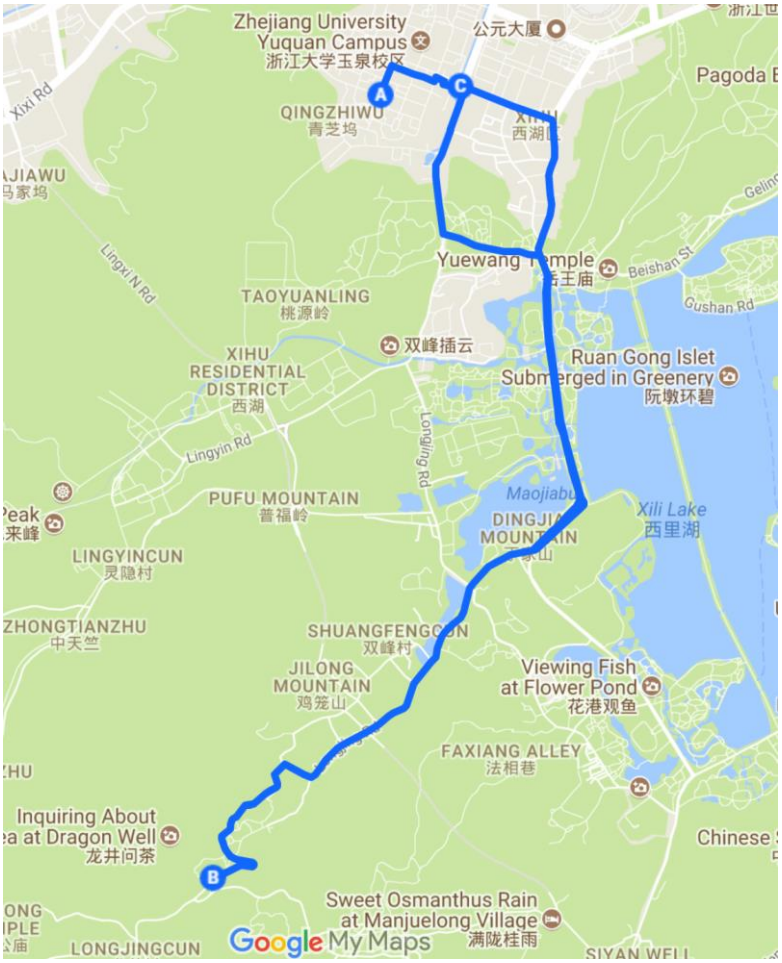

图 4.28 城市道路动态测试中行车路线图

原型样机程序中可保存检测到存在交通标志时的视频图像帧，一些正确检测结果见图 4.29，图中各图片均是由原型样机自动保存记录的实际实时检测结果图。由于行车过程中不方便记录一共存在多少个交通标志此处未能得出准确的检出率，估计检出率约为 60%~70%左右，与校园环境静态测试中 91%的检出率相比有较大下降，这主要是由于下雨情况下光照发生变化导致颜色分割策略阈值与晴天存在较大差异导致的；另外在阴雨天气下交通标志的最远检出距离无法达到上一节中天气光照良好时的 40 余米，此时最远检出距离有较大下降，缩短到仅有 20m~30m 左右。此结果说明基于本文颜色分割策略构建的交通标志定位算法在天气光照变化下的鲁棒性有所欠缺，当外部环境不佳时的检测定位效果依然存在很大的提升空间。

图 4.29 城市道路动态测试中部分正确检测结果图

此外从测试结果中发现颜色分割后进行 Hough 圆检测的经典流程对于遮挡有一定的处理适应能力，图 4.30 所示情况中两个交通标志被树枝挡住了一部分，然而在车辆驶过交通标志的过程中不同帧内可分别实现对这两个交通标志的正确检出定位。

图 4.30 城市道路动态测试中遮挡情况下正确检测结果图

与校园环境静态测试中一样，城市道路动态测试中也存在一些误检情况，典型情景见图 4.31，这类误检主要是由于车灯、交通信号灯、其余车辆（特别是公交车）上的装饰性图片引起的，事实上此时的确构成了红色的圆形区域，故仅从检测定位这一步来看并没有很好的解决误检的方法，此类误检需要靠下一步模式分类去除。

图 4.31 城市道路动态测试中部分误检结果图

测试中最高车速为 70km/h 左右，此时依然能正确检出交通标志（未检出的标志不是由于车速过快导致的），这说明采用多线程技术并配合本文第三章的混合切换多线程任务调度策略可保证算法在嵌入式原型样机中的实时性满足交通标志检测的要求；这也就意味着本文提出的方法对于实际的基于小型嵌入式设备搭建的车载安全预警系统而言也具有一定的实用价值，能满足此类资源有限系统较高的实时性要求。

4.4 本章小结

本章首先对本文涉及的算法验证平台、性能测试平台及嵌入式原型样机的基本情况进行了介绍，并对原型样机平台及摄像头的选型进行了分析讨论，最终通过比较选择了采用 Intel Joule 模块及 KS2A17 摄像头模块搭建嵌入式原型样机。在此基础上介绍了算法验证平台及原型样机软硬件开发过程，对算法验证平台的线程间同步通信机制等核心技术实现方法进行了阐述，并通过实验测试确定了摄像头最佳输出分辨率，根据测试结果对于 KS2A17 摄像头模块来说输出分辨率为 800*600 时系统的实时性最好。最后本章使用原型样机在浙大校园环境及杭州城市道路上分别进行了静态及动态集成测试，测试结果表明本文基于颜色分割的交通标志定位算法及混合多线程任务调度策略在原型样机上可以有效实现红色圆形禁令标志的实时检出，特别是在天气良好且图像稳定时检出率很高，不过在天气不佳情况下检出率及最远检出距离均会有一定降低；且两种情况下都存在一定的误检情况。综上，本章通过实际测试证明了本文研究的方法在嵌入式实时系统中的可用性与有效性，不过城市道路中的测试结果也说明本文的方法尚存在一定缺陷需要在将来的研究中进一步改善。

5 总结与展望

5.1 研究工作总结

嵌入式车载安全预警系统对于提高驾驶安全性有很大帮助，而交通标志检测作为其中的一个重要基础技术尚存在很多需要深入研究的技术问题，本文对基于颜色分割的交通标志定位算法及混合切换多线程任务调度策略这两项嵌入式交通标志检测中的关键技术进行了研究，并完成了算法验证平台软件的开发及嵌入式原型样机的搭建。本文主要进行的工作及研究成果如下：

(1) 对交通标志定位中常用的 RGB 阈值分割法、HSI 颜色空间转换法及 SVF 分割方法的基本原理进行了研究并对其分割效果进行了评估，重点对两种不同的 HSI 颜色空间转换公式的差异性及执行效率进行了对比，比较结果表明分段函数形式的转换公式执行效率远高于经典的反余弦函数转换公式；在此基础上针对已有颜色分割方法对特定颜色分割不准确的缺陷提出了一种用于分割红色及黄色的混合颜色分割策略，此策略是以上多种方法的结合且完全基于线性分类器，测试验证结果表明其分割效果及算法执行效率均较为理想，对于红色及黄色分割来说比目前已有方法更有优势；最后采用 Hough 变换实现了对红色圆形禁令标志的检测与定位，并在数据集上进行了测试验证，结果表明本文采用的基于颜色分割的交通标志定位算法在天气光照良好时具有相对较高的检出率，不过在夜间、逆光等不利光照情况下算法检出率不高。

(2) 对交通标志检测识别问题的实时性要求进行定量分析并提出以采样间隔时间作为此类采样检测系统的实时性量度，在此基础上通过建模分析得出了理论最优的多线程理想任务调度算法；针对此算法实际不能实现的问题分别设计了混合切换调度策略及动态更新参数估计策略，并通过控制系统模型数值仿真及实际原型样机上的测试验证了本文的任务调度策略的确能优化采样间隔时间分布情况进而提高系统实时性；最后给出了此混合采样多线程任务调度策略在 POSIX 标准下程序框架的实现方法。

(3) 建立了中国道路交通标志数据集并将其公开发布，这是公开领域目前唯一的中国道路交通标志数据集，此数据集也弥补了国外同类公开数据集中视频片段较少的缺陷。

(4) 开发了算法仿真验证软件，此软件可通过图形界面窗口打开图片及视频并运行特定的图像处理算法对其进行处理，算法的选择及其参数设置均能在图形界面中方便的完成；除此之外通过合理的软件架构设计实现了界面与算法的解耦，以此保证主界面能及

时响应用户请求并使得程序具有较好的可扩展性。

(5) 采用 Intel Joule 模块搭建了嵌入式原型样机，在此原型样机上对基于颜色的交通标志定位算法及混合切换多线程任务调度策略进行了集成测试验证，测试结果表明本文的方法可在天气光照良好时较为准确的定位红色圆形交通标志，不过若天气光照情况不佳检出率会有所下降；除此之外还对摄像头最佳输出分辨率进行了研究，测试结果表明对于本文使用的 KS2A17 摄像头模块来说输出分辨率为 800*600 时系统的实时性最好。

5.2 后续研究展望

由于时间有限本文虽然进行了一定的工作也取得了一些成果，然而还存在很多不完善的地方需要在今后的研究中进一步进行更为深入的探索。

(1) 本文提出的混合颜色分割方法在光照良好时具有较好的分割效果，然而在夜间、逆光等情况下鲁棒性不佳，此时颜色分割结果不准确导致后续基于颜色分割的形状定位方法无法进行；虽然可通过调整算法阈值改善特定情况下颜色分割效果，然而很难找到一组同时适用于各情况下的阈值，后续研究可考虑加入自适应算法判断此时的场景调用合适的阈值进行处理。

(2) 交通标志定位识别中本文的研究尚缺乏完整性，由于时间关系仅对红色及黄色的分割及圆形交通标志的定位方法进行了研究，后续研究中尚需要对其余类型交通标志的检测定位及后续的模式分类识别问题进行研究。

(3) 目前人工智能（AI）、深度学习等领域的研究极为火热，且人工智能芯片等基础硬件也在迅速发展，能否采用这些方法在嵌入式平台上实现交通标志的实时检测识别及其它更多的车载安全预警应用也是很值得研究的方向；此类方法需要大量数据的支撑，本文建立的交通标志数据集对后续这方面的研究提供了很好的基础。

(4) 本文原型样机使用的平台是 Intel Joule 模块，不过遗憾的是由于 Joule 模块价格过高等原因，与树莓派蓬勃的发展相比其市场反响并不好，因此 Intel 公司已经于 2017 年底停止 Joule 模块的生产及相关软件（操作系统、工具包等）的开发维护，同时也将放弃 Galileo 及 Edison 产品线；故后续研究中需要对其余嵌入式平台的性能进行评估选型以替换 Intel Joule 模块。

参考文献

[1] 中华人民共和国国家统计局. 中国统计年鉴—2016[M]. 北京: 中国统计出版社, 2016.

[2] 孔令铮. 交通事故致因中的人为因素分析[J]. 中国安全科学学报, 2013(01):28-34.

[3] Williams M. PROMETHEUS-The European research programme for optimising the road transport system in Europe: IEE Colloquium on Driver Information, 1988[C]. IET.

[4] Wikipedia. Eureka Prometheus Project[EB/OL]. https://en.wikipedia.org/wiki/Eureka_Prometheus_Project.

[5] BMW Group. G12 Driver Assistance Systems, BV-72[R].2015.

[6] 宋健, 王伟玮, 李亮, 等. 汽车安全技术的研究现状和展望[J]. 汽车安全与节能学报, 2010,1(2):98-106.

[7] 小蚁科技. 小蚁智能行车记录仪[EB/OL]. https://www.xiaoyi.com/yicarcam/yicarcam.html.

[8] Ruskin. 小蚁智能行车记录仪深度体验评测[EB/OL]. http://news.mydrivers.com/1/472/472503_all.htm.

[9] Ambarella. A12A Advanced HD Automotive Camera SoC[R].2016.

[10] Lowe D G. Distinctive image features from scale-invariant keypoints[J]. International journal of computer vision, 2004,60(2):91-110.

[11] Lowe D G. Object recognition from local scale-invariant features: The proceedings of the seventh IEEE international conference on Computer vision, 1999[C]. IEEE.

[12] Viola P, Jones M. Rapid object detection using a boosted cascade of simple features: Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), 2001[C]. IEEE.

[13] Lake B M, Salakhutdinov R, Tenenbaum J B. Human-level concept learning through probabilistic program induction[J]. Science, 2015,350(6266):1332-1338.

[14] Sun Y, Wang X, Tang X. Deep learning face representation from predicting 10,000 classes: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2014[C]. IEEE.

[15] Howard A R. Traffic sign recognition[J]. Canadian Good Roads Association Proc, 1964.

[16] 周欣. 圆形和三角形交通标志分割与识别算法研究[D]. 华东理工大学, 2013.

[17] 何春. 基于特征的交通标志图像识别的应用研究[D]. 广东工业大学, 2013.

[18] Zaklouta F, Stanciulescu B. Real-time traffic sign recognition in three stages[J]. Robotics and autonomous systems, 2014,62(1):16-24.

[19] TIAN Q, LIU C, DU X. Research on Method of Traffic Signs Recognition Based on Hu Invariant Moments and Zernike Invariant Moment[J]. Journal of Zhejiang Sci-Tech University, 2012,2:19.

[20] Wang B, Kong B, Ding D, et al. A novel traffic sign recognition algorithm based on sparse representation and dictionary learning[J]. Journal of Intelligent & Fuzzy Systems, 2017,32(5):3775-3784.

[21] Abedin Z, Dhar P, Hossenand M K, et al. Traffic sign detection and recognition using fuzzy segmentation approach and artificial neural network classifier respectively: International Conference on Electrical, Computer and Communication Engineering (ECCE), 2017[C]. IEEE.

[22] 张潘潘. 道路交通标志检测与识别算法的研究[D]. 山东大学控制科学与工程, 2012.

[23] Sompoch P, Xiaoyangv C, Michiro K. Automatic Recognition and Location of Road Signs from terrestrial color Imagery[J]. Bangkok, Thailad, Geoinformatics& DMGIS, 2001,2001:238-247.

[24] Betke M, Makris N C. Fast object recognition in noisy images using simulated annealing: Proceedings of IEEE International Conference on Computer Vision, 1995[C]. IEEE.

[25] De La Escalera A, Moreno L E, Salichs M A, et al. Road traffic sign detection and classification[J]. IEEE transactions on industrial electronics, 1997,44(6):848-859.

[26] Sathiya S, Balasubramanian M, Palanivel S. Pattern recognition based detection recognition of traffic sign using SVM[J]. Int J Eng Technol, 2014,6(2):1147-1157.

[27] Xu Y, Wang Q, Wei Z, et al. Traffic sign recognition based on weighted ELM and AdaBoost[J]. Electronics Letters, 2016,52(24):1988-1990.

[28] Ritter W, Stein F, Janssen R. Traffic sign recognition using colour information[J]. Mathematical and computer modelling, 1995,22(4):149-161.

[29] CireşAn D, Meier U, Masci J, et al. Multi-column deep neural network for traffic sign classification[J]. Neural Networks, 2012,32:333-338.

[30] Jin J, Fu K, Zhang C. Traffic sign recognition with hinge loss trained convolutional neural networks[J]. IEEE Transactions on Intelligent Transportation Systems, 2014,15(5):1991-2000.

[31] Zeng Y, Xu X, Shen D, et al. Traffic Sign Recognition Using Kernel Extreme Learning Machines With Deep Perceptual Features[J]. IEEE Transactions on Intelligent Transportation Systems, 2016.

[32] Gim J, Hwang M, Ko B C, et al. Real-Time Speed-Limit Sign Detection and Recognition Using Spatial Pyramid Feature and Boosted Random Forest: International Conference Image Analysis and Recognition, 2015[C]. Springer.

[33] Ellahyani A, El Ansari M, El Jaafari I. Traffic sign detection and recognition based on random forests[J]. Applied Soft Computing, 2016,46:805-815.

[34] Mogelmose A, Trivedi M M, Moeslund T B. Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey[J]. IEEE Transactions on Intelligent Transportation Systems, 2012,13(4):1484-1497.

[35] 仲崇峰. 基于ARM的道路交通标志识别系统的研究[D]. 长春理工大学电子科学与技术, 2014.

[36] Han Y, Oruklu E. Real-time traffic sign recognition based on zynq fpga and arm socs: IEEE International Conference on Electro/Information Technology (EIT), 2014[C]. IEEE.

[37] Ganapathi K, Madumbu V, Rajendran R, et al. Design and implementation of an automatic traffic sign recognition system on TI OMAP-L138: IEEE International Conference on Industrial Technology (ICIT), 2013[C]. IEEE.

[38] Berkaya S K, Gunduz H, Ozsen O, et al. On circular traffic sign detection and recognition[J]. Expert Systems with Applications, 2016,48:67-75.

[39] Hamdi S, Faeidh H, Farhat W, et al. Real-Time Implementation of Light-Independent Traffic Sign Recognition Approach[M]//Real-Time Modelling and Processing for Communication Systems. Springer, 2018:257-282.

[40] Fu K, Gu I Y, Ödblom A. Traffic sign recognition using salient region features: A novel learning-based coarse-to-fine scheme: IEEE Intelligent Vehicles Symposium (IV), 2015[C]. IEEE.

[41] Zaklouta F, Stanciulescu B. Real-time traffic sign recognition in three stages[J]. Robotics and autonomous systems, 2014,62(1):16-24.

[42] Lee E, Lee S, Hwang Y, et al. Hardware implementation of fast traffic sign recognition for intelligent vehicle system: International SoC Design Conference (ISOCC), 2016[C]. IEEE.

[43] Asakura T, Aoyagi Y, Hirose O K. Real-time recognition of road traffic sign in moving scene image using new image filter: SICE 2000. Proceedings of the 39th SICE Annual Conference. International Session Papers (IEEE Cat. No.00TH8545), 2000[C].

[44] Gómez-Moreno H, Maldonado-Bascón S, Gil-Jiménez P, et al. Goal evaluation of segmentation algorithms for traffic sign recognition[J]. IEEE Transactions on Intelligent Transportation Systems, 2010,11(4):917-930.

[45] Fleyeh H. Shadow and highlight invariant colour segmentation algorithm for traffic signs: IEEE Conference on Cybernetics and Intelligent Systems, 2006[C]. IEEE.

[46] Lillo-Castellano J M, Mora-Jiménez I, Figuera-Pozuelo C, et al. Traffic sign segmentation and classification using statistical learning methods[J]. Neurocomputing, 2015,153:286-299.

[47] Marinas J, Salgado L, Camplani M. Multi-resolution model-based traffic sign detection and tracking.: Real-Time Image and Video Processing, 2012[C].

[48] 常卫. 基于机器视觉的染色品色差检测系统的关键技术研究[D]. 浙江理工大学, 2012.
[49] Giosan I, Nedevschi S, Pocol C. Shape improvement of traffic pedestrian hypotheses by means of stereo-vision and superpixels: IEEE International Conference on Intelligent Computer Communication and Processing (ICCP), 2015[C]. IEEE.
[50] Wang S, Pan H, Zhang C, et al. RGB-D image-based detection of stairs, pedestrian crosswalks and traffic signs[J]. Journal of Visual Communication and Image Representation, 2014,25(2):263-272.
[51] Youn J, Kim G H, Chong K. Automatic Extraction of Direction Information from Road Sign Images Obtained by a Mobile Mapping System[J]. 2014.
[52] Yao Z, Yi W. Curvature aided Hough transform for circle detection[J]. Expert Systems with Applications, 2016,51:26-33.
[53] Qin J, ZHANG X. Hierarchical traffic sign recognition system based on improved shape context[J]. Computer Engineering and Design, 2014,1:34.
[54] Eickeler S, Valdenegro M, Werner T, et al. Future computer vision algorithms for traffic sign recognition systems[M]//Advanced Microsystems for Automotive Applications 2015. Springer, 2016:69-77.
[55] 何江萍. 三角形交通标志的检测方法研究[D]. 兰州大学, 2009.
[56] Salhi A, Minaoui B, Fakir M. Robust Automatic Traffic Signs Detection using fast polygonal approximation of digital curves: International Conference on Multimedia Computing and Systems (ICMCS), 2014[C]. IEEE.
[57] 张静, 何明一, 戴玉超, 等. 综合颜色和形状的圆形交通标志检测方法[J]. 计算机工程与应用, 2011(02):233-236.
[58] Gowardhan M Y, Hablani R. Traffic sign detection and analysis[J]. Traffic, 2016,3(02).
[59] Qingsong X, Juan S, Tiantian L. A detection and recognition method for prohibition traffic signs: International Conference on Image Analysis and Signal Processing (IASP), 2010[C]. IEEE.
[60] Schmidt T, Liu G, Dömer R. Exploiting thread and data level parallelism for ultimate parallel SystemC simulation: 54th ACM/EDAC/IEEE Design Automation Conference (DAC), 2017[C]. IEEE.
[61] Hrbacek R, Sekanina L. Towards highly optimized cartesian genetic programming: From sequential via simd and thread to massive parallel implementation: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, 2014[C]. ACM.
[62] Wikipedia. Dhrystone[EB/OL]. https://en.wikipedia.org/wiki/Dhrystone.
[63] Wikipedia. Instructions per second[EB/OL]. https://en.wikipedia.org/wiki/Instructions_per_second.
[64] Wikipedia. List of ARM microarchitectures[EB/OL]. https://en.wikipedia.org/wiki/List_of_ARM_microarchitectures.
[65] Kleihorst R, Abbo A, Schueler B, et al. Camera mote with a high-performance parallel processor for real-time frame-based video processing: First ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC), 2007[C]. IEEE.
[66] 刘晓东. 嵌入式处理器中多媒体加速单元的研究[D]. 浙江大学, 2012.
[67] Gibson K B, Nguyen T Q, Yoon H. Improvements in parallel SIMD implementation of single image defogging: International SoC Design Conference (ISOCC), 2016[C]. IEEE.
[68] Wulf W, Levin R, Pierson C. Overview of the Hydra operating system development: ACM SIGOPS Operating Systems Review, 1975[C]. ACM.
[69] Marçais G, Kingsford C. A fast, lock-free approach for efficient parallel counting of occurrences of k-mers[J]. Bioinformatics, 2011,27(6):764-770.
[70] Akbari M, Rashidi H. A multi-objectives scheduling algorithm based on cuckoo optimization for task allocation problem at compile time in heterogeneous systems[J]. Expert Systems with Applications, 2016,60:234-248.
[71] Xu Y, Li K, Hu J, et al. A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues[J]. Information Sciences, 2014,270:255-287.

[72] Ergu D, Kou G, Peng Y, et al. The analytic hierarchy process: task scheduling and resource allocation in cloud computing environment[J]. The Journal of Supercomputing, 2013:1-14.
[73] Houben S, Stallkamp J, Salmen J, et al. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark: International Joint Conference on Neural Networks (IJCNN), 2013[C]. IEEE.
[74] Larsson F, Felsberg M. Using Fourier Descriptors and Spatial Models for Traffic Sign Recognition.: SCIA, 2011[C]. Springer.
[75] Timofte R, Prisacariu V A, Gool L V, et al. Combining traffic sign detection with 3D tracking towards better driver assistance[M]//Emerging topics in computer vision and its applications. World Scientific, 2012:425-446.
[76] 崔迪潇. 公开数据 DATA BASE[EB/OL]. http://www.dixiaocui.com/research/data-base/.
[77] Wikipedia. Color filter array[EB/OL]. https://en.wikipedia.org/wiki/Color_filter_array.
[78] Wikipedia. Foveon X3 sensor[EB/OL]. https://en.wikipedia.org/wiki/Foveon_X3_sensor.
[79] Deglint J, Kazemzadeh F, Cho D, et al. Numerical demultiplexing of color image sensor measurements via non-linear random forest modeling[J]. Scientific reports, 2016,6:28665.
[80] 常卢峰. 车载辅助系统中交通标志检测与识别技术研究[D]. 中南大学, 2010.
[81] Gonzalez R C, Woods R E. Processing[Z]. Prentice-Hall, 2002.
[82] Chien B, Cheng M. A color image segmentation approach based on fuzzy similarity measure: Proceedings of the 2002 IEEE International Conference on Fuzzy Systems, 2002[C]. IEEE.
[83] OpenCV. Miscellaneous Image Transformations[EB/OL]. https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html.
[84] 刘华波. RGB 与 HSI 颜色模型的转换方法对比研究[Z]. 2008.
[85] Maldonado-Bascon S, Lafuente-Arroyo S, Gil-Jimenez P, et al. Road-sign detection and recognition based on support vector machines[J]. IEEE transactions on intelligent transportation systems, 2007,8(2):264-278.
[86] Deshpande J V. On continuity of a partial order[J]. Proceedings of the American Mathematical Society, 1968,19(2):383-386.
[87] Soille P. Morphological image analysis: principles and applications[M]. Springer Science & Business Media, 2013.
[88] 毛星云, 冷雪飞, 王碧辉. OpenCV3 编程入门[J]. 北 京: 电 子 工, 2015.
[89] Chai Y, Wei S J, Li X C. The Multi-Scale Hough Transform Lane Detection Method Based on the Algorithm of Otsu and Canny: Advanced Materials Research, 2014[C]. Trans Tech Publ.
[90] OpenCV. Hough Circle Transform[EB/OL]. https://docs.opencv.org/3.3.1/d4/d70/tutorial_hough_circle.html.
[91] IEEE. ANSI/IEEE 1003.1-1988 IEEE Standard Portable Operating System Interface for Computer Environments[S]. 1988.
[92] Akopytov. sysbench[EB/OL]. https://github.com/akopytov/sysbench.
[93] Intel. Intel Joule Module Datasheet[R].2017.
[94] OmniVision. OV2710-1E full HD (1080p) product brief[R].2017.
[95] OmniVision. OV4689 4MP product brief[R].2017.
[96] Havendv. QtPackage[EB/OL]. https://marketplace.visualstudio.com/items?itemName=havendv.QtPackage.
[97] Intel. Intel Joule Developer Kit Thermal Management Overview[R].2016.
[98] Linux Foundation. Yocto Project[EB/OL]. https://www.yoctoproject.org/.

作者简介

高明飞，男，2015 年获上海交通大学化学化工学院应用化学专业学士学位，现为浙江大学控制科学与工程学院硕士研究生。主要研究方向为嵌入式系统、电机控制等。