第七届“飞思卡尔”杯全国大学生智能汽车竞赛

技 术 报 告

学　　校：　山 东 大 学

队伍名称：　风火流星

参赛队员：　赵茂行

　　朱　明

　　王　黎

带队教师：　李振华

　　刘成云

关于技术报告和研究论文使用授权的说明

本人完全了解第七届“飞思卡尔”杯全国大学生智能汽车竞赛关保留、使用技术报告和研究论文的规定，即：参赛作品著作权归参赛者本人，比赛组委会和飞思卡尔半导体公司可以在相关主页上收录并公开参赛作品的设计方案、技术报告以及参赛模型车的视频、图像资料，并将相关内容编纂收录在组委会出版论文集中。

参赛队员签名：_____

带队教师签名：_____

日　　期：_____

摘 要

本文以第七届飞思卡尔杯智能汽车竞赛为背景详细介绍了智能车系统的软硬件结构和开发流程。采用 1:10 的仿真车模，以飞思卡尔半导体公司生产的 Kinetis 系列 32 位处理器 PK10N512 为核心控制器，在 CodeWarrior 10.1 和 IAR6.306 IDE 开发环境中进行软件开发，使赛车在跑道上沿着跑道以最快的速度行驶。整个系统涉及车模机械结构的设计与调整、系统架构及硬件设计、软件算法和系统调试等多个方面。机械结构介绍了车模轮胎的优化，前轮的定位，舵机调整及摄像头的安放位置调整等环节；硬件方面介绍了单片机的选型，各个硬件模块的设计，其中主要介绍了电源管理模块、摄像头采集模块和电机的 H 桥驱动；软件方面则重点介绍了主要资源模块的初始化和 PID 算法；最后具体介绍了几个智能车系统的调试工具，重点对 SD 卡上位机和下位机进行了详细的介绍。

关键词：K10，H 桥，PID，边沿算法

ABSTRACT

This paper introduces the hardware and software structure of smart car system and development process In the 7th freescale cup competition for intelligent vehicle. Using 1:10 simulation models, with Freescale Semiconductor company's 32-bit MCU PK10N512 of Kinetis family as the core, do the software development in the CodeWarrior 10.1 and IAR 6.306 IDE development environment and make car on the track with the fastest speed. The whole system involves the mechanical structure design and adjustment, system architecture and design of hardware, software algorithm and debugging systems, etc. The front tire's optimization, the front position, steering gear adjustment and the camera position are introduced in the mechanical structure; The MCU hardware selection and hardware design of each module are introduced in the hardware design, in which the power management module, camera acquisition module and motor derive with H bridge are mainly introduced; In the software, the main resource module initialization and PID algorithm are the focus; Finally, several debugging tools of smart car are introduced in detail, focusing on the SD card on PC and MCU.

Key words: K10, Camera, H bridge, PID, Edge algorithm

目 录

第一章　绪论

1.1 课题背景及意义

随着现代科技的发展，汽车的拥有量越来越多，随之而来的汽车拥堵，车祸频发等一系列安全问题引起了人们的重视。因此，人们对汽车的智能化要求越来越高，汽车生产商也推出越来越智能的汽车，来满足各种各样的市场需求，汽车的电子化和智能化已经成为了行业发展的必然趋势，自 2006 年以来每年举行的全国大学生智能汽车竞赛就是在此背景下举行的。

全国大学生智能车竞赛以迅猛发展的的汽车电子为背景，至今已经发展到了第七届，是一项涵盖控制、模式识别、传感、电子、电气、计算机和机械等多个学科交叉的科技创意性比赛，对学生的知识融合和实践动手能力的培养，具有良好的长期的推动作用，并且本项赛事对进一步深化高等工程教育改革，培养本科生获取知识、应用知识的能力及创新意识，培养本科生从事科学、技术研究能力同样具有重要的意义，对于高校相关学科学术领域学术水平的提高也有一定的帮助。

本小组就是搭建基于视觉导航的智能汽车系统，通过摄像头对赛道图像的有效采集，然后再用控制器进一步处理信息，做出相应的决策，控制小车寻迹。这种视觉寻迹系统以其灵活，信息量大等优势成为了未来的寻迹发展方向，在将来智能汽车电子应用上非常有前景，同时在智能机器人、导弹寻迹等方面也有很大的应用空间，在不久的将来将会有其更广大的发展空间。

1.2 整体思路和总体介绍

本文分章节介绍了智能车系统的各个模块，包括智能车机械机构的设计及调整、智能车系统架构及硬件设计、智能车软件设计方案和系统调试。

本智能车系统采用飞思卡尔公司的 PK10N512 芯片为核心控制器，配合有相应的硬件及驱动电路，组成一个自动控制系统，由摄像头传感器、信息处理、控制算法和执行机构组成。

在小车上安装摄像头负责采集赛道信息，电机传动齿轮上装有光电编码器用来采集车轮的速度，并将信号传到核心控制单元处理后，由 PWM 模块产生相应的 PWM 波形驱动舵机和电机，控制智能车转向，前进和制动。

在控制算法上，采用位置式 PD 和增量式 PID 控制舵机和电机，窗口算法采集处理跑道，控制转速和转向。

第二章　智能车机械设计及调整

模型车的机械结构和组装形式是整个车身的基础，经过大量的实验和过后总结的经验可以看出来，机械结构的好坏对智能车的运行有直接的影响，可以说机械结构决定了智能车的上限速度，而软件算法的优化就是使车速不断接近这个上限速度，一辆软件十分优秀的小车，需要有精细的机械结构才能够不影响智能车整体性能的发挥，由此可以看出机械结构的重要性，为此我们小组不遗余力地对车模的的机械结构部分做了较大的改进。

2.1 轮胎优化调整

轮胎是车身与跑道直接接触的部分，直接影响智能车附着力（俗称抓地力）与过弯性能。在我们长时间的实际调车过程中，发现轮胎的摩擦力大小可很大程度上弥补舵机拐弯的不及时，大大改善车身拐弯效果。

在调车过程中我们发现新轮胎由于有中间分模面的缘故，并没有完全接触到跑道因此摩擦力相应小些，为此我们事先进行人为前期磨损，使其达到好的附着性能。另外，在高速过弯时，地面对轮胎侧向力很大，易使轮胎脱离轮毂，这时可以考虑将轮胎粘在轮毂上。

2.2 车轮的定位和舵机调整

现代汽车在正常行驶过程中，为了使汽车直线行驶稳定，转向轻便，转向后能自动回正，减少轮胎和转向系零件的磨损等，在转向轮、转向节和前轴之间须形成一定的相对安装位置，叫车轮定位，其主要的参数有：前轮前束，前轮外倾和后轮外倾。

2.2.1 前轮前束

Toe 角度（束角）是描述从车的正上方看，车轮的前端和车辆纵线的夹角。车轮前端向内倾（内八字），称为 Toe-in，车轮前端向外倾（外八字），称为 Toe-out。前轮在滚动时，其惯性力会自然将轮胎向内偏斜，如果前束适当，轮胎滚动时的偏斜方向就会抵消，轮胎内外侧磨损的现象会减少。不同的 Toe 角度会改变车辆的转向反应和直道行驶的稳定性。可以通过改变前万向节拉杆的长度来改变 Toe 角度。经过多次试验，选择了效果相对较好的内八。

2.2.2 前轮外倾

前轮外倾角是指通过车轮中心的汽车横向平面与车轮平面的交线与地面垂线之间的夹角，实际汽车一般采用正外倾，即从正面看车轮呈现“八”字。主要目的是为承载车辆车轮磨损均匀，并有一定的回正力矩，可以通过调整前上桥的拉杆的长度来改变前轮的外倾角度。

2.2.3 后轮外倾

后轮外倾影响车辆的稳定性和抓地力，通过调整后悬挂长柄万向节上的拉杆的长度来改变后轮的外倾角；也可通过改变后悬挂长柄万向节车身一端的固定位置来获得不同的外倾角。

2.2.4 舵机的调整

我们采用舵机直立放置的方式，采用机床加工的舵机支架，并将舵机安放于车模前部中间位置，以保证车模转向的机械对称性。舵机连杆我们使用雕刻机加工而成，采用标准化的舵机尺寸和连接方式，适当加长舵机力臂，并保证力矩作用在水平方向上而减小其他方向的损失，使舵机输出力臂增加，即让舵机转动一小角度前轮转动大角度，但这种做法一定程度上是以减小力矩为代价的，所以力臂不能加的太长，为此经过反复的调试，选取了恰当长度的连接杆，同时保证了较快的反应速度和足够大的输出力矩。

2.3 重心调整与车上设备的安放

在电机和电池相同的情况下，车体重量对于车的加减速性能有着至关重要的影响，因此在车身支架及设备布置时应该尽量减少不必要的配置，使车总重尽量减少。

车体重心高低主要影响车身运动中的稳定性，对于平顺性也有一定影响。重心调整主要目标是在过坡的前提下尽量降重心。对于 A 车，我们增加前轮垫片，使整体车身降低，同时严格控制摄像头和安装架的重量，摄像头采用较轻的 CMOS 黑白摄像头，安装架采用碳纤维杆，并将摄像头安放于车体中心附近，在重心降低的同时也后移了重心，使车模拐弯的时候更平稳减少了侧翻的可能，也减小了拐弯的负担。

经过以上的步骤改装后的摄像头 A 车如图 2.1 所示。

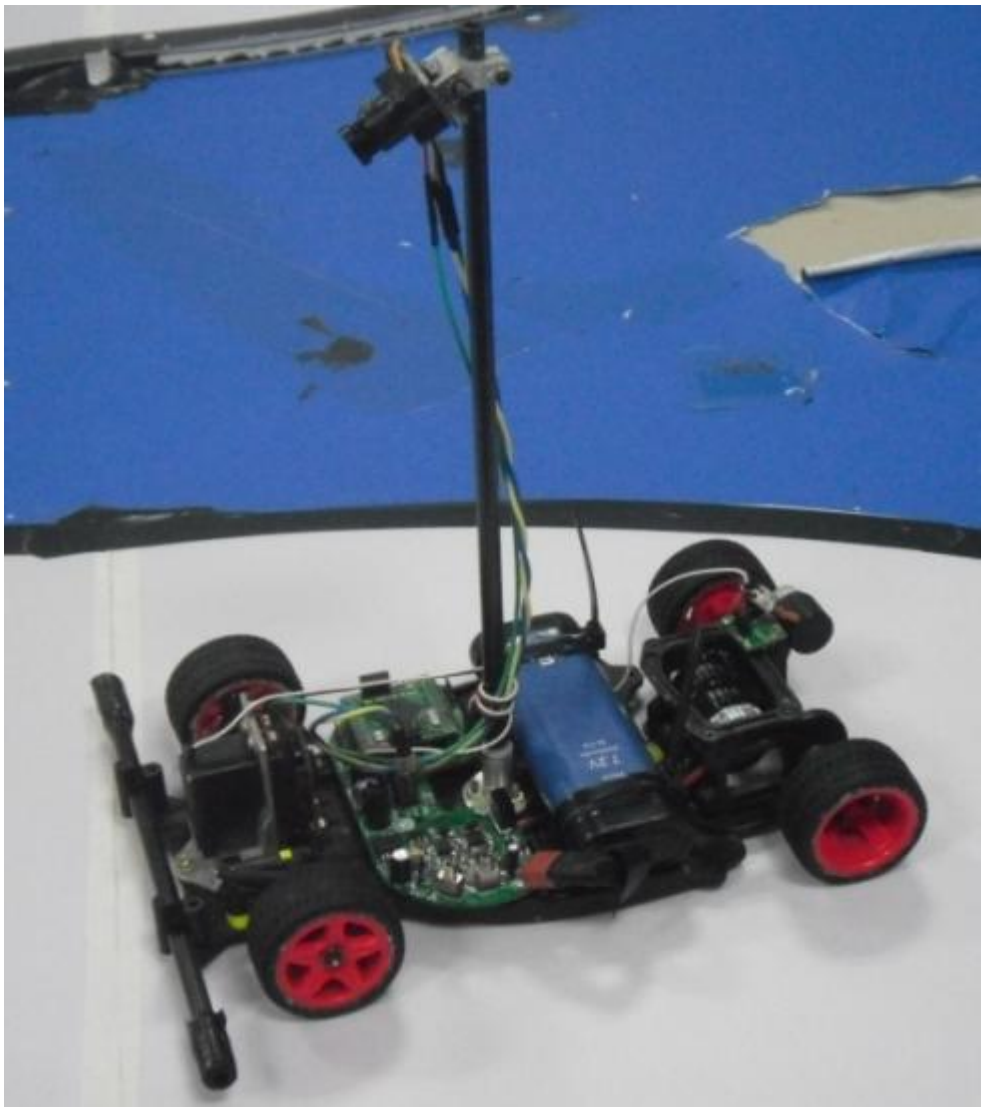

图 2.1 车模原型照

第三章　智能车系统架构及硬件设计

对智能车来说，硬件系统的可靠性是必须的，为此我们小组花费了大量时间在硬件电路设计和功能论证上，最后确定了比较可靠的硬件电路，以下是智能车系统的完整架构。

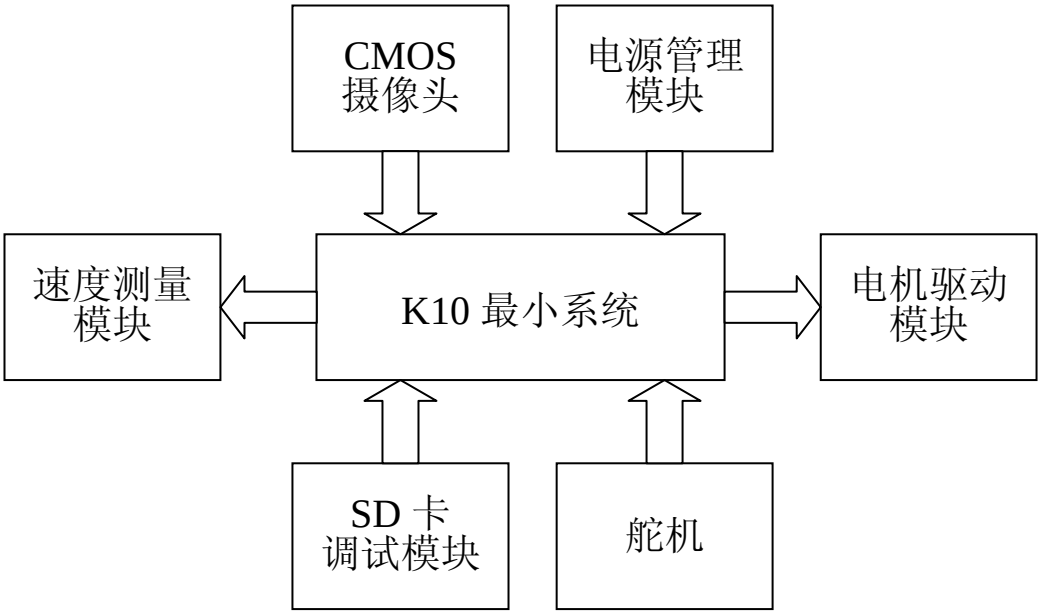

图 3.1 智能车系统结构图

本系统是以 K10N512 处理器为核心，设计在较严格循线的基础上能以最优化的速度完成一圈的智能车系统。实现功能算法的前提，要先设计出功能模块的电路，进行调试后再用算法实现。视频信号是通过摄像头采集模块摄取赛道图像并以 NTSC 制式模拟信号输出比较器进行硬件二值化，由单片机系统的中断功能和计数功能对视频图像二值化并进行视频同步信号分离，进一步处理获得图像信息。依据处理过的图像信息，通过光电编码器来检测车速形成闭环，利用适当的算法，通过 PWM 控制电机的速度及功率和舵机的转向。

3.1 K10 最小系统

K10N512 是基于 ARM® CortexTM-M4 具有超强可扩展性的低功耗、混合信号微控制器。第一阶段产品由五个微控制器系列组成，包含超过两百种器件，在引脚、外设和软件上可兼容。每个系列提供了不同的性能，存储器和外设特性。通过通用外设、存储器映射和封装的一致性来实现系列内和各系列间的便捷移植。Kinetis 微控制器基于飞

思卡尔创新的 90 纳米薄膜存储器（TFS）闪存技术，具有独特的 Flex 存储器（可配置的内嵌 EEPROM）。 Kinetis 微控制器系列融合了最新的低功耗革新技术，具有高性能、高精度的混合信号能力，宽广的互连性，人机接口和安全外设。飞思卡尔公司以及其他大量的 ARM 第三方应用商提供对 Kinetis 微控制器的应用支持。其基本特性为：

- 电压范围 1.71V - 3.6V
- 温度范围 (TA) -40 to 105° C
- 32 位 ARM Cortex-M4 内核
- 支持 DSP 指令
- 嵌套向量中断控制器 (NVIC)
- 异步唤醒中断控制器 (AWIC)
- 调试和跟踪
- 2 引脚串口调试 (SWD)
- IEEE 1149.1 JTAG 调试
- IEEE 1149.7 简洁 JTAG
- 端口跟踪接口单元 (TPIU）
- 闪存片和断点单元 (FPB)
- 数据检测和跟踪单元 (DW)
- 指令跟踪宏单元 (ITM)

3.2 电源管理模块的设计

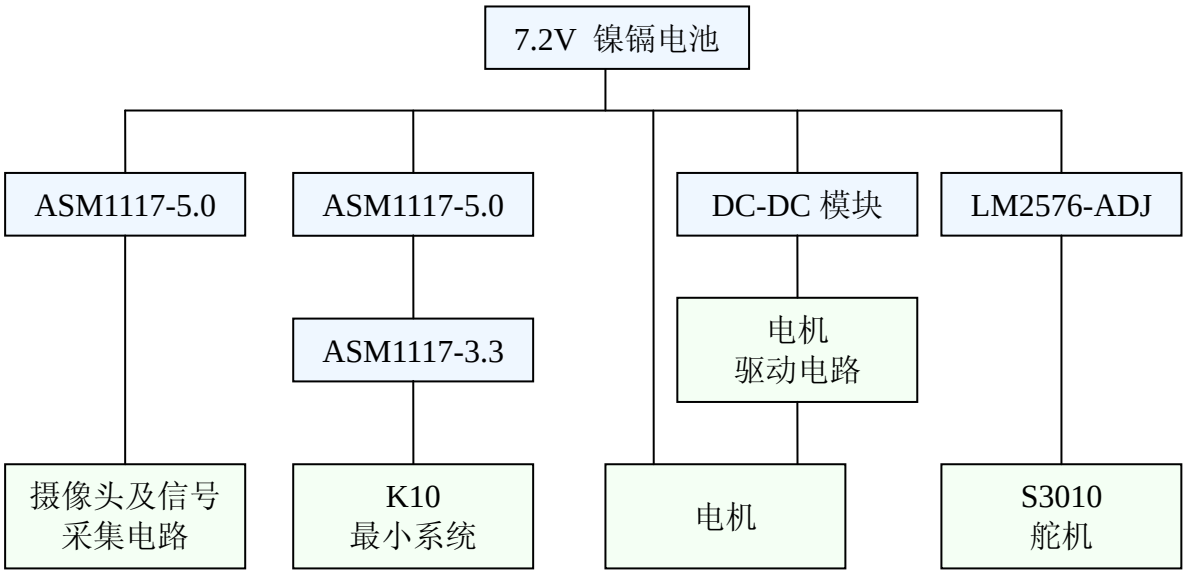

图 3.2 电源管理系统框架图

稳定的电源对于一个控制系统来说至关重要，关系到系统能否正常工作，因此在设计智能车系统时为各个模块配置了合适、稳定的电源并且在电路设计上尽量避免不同电源和相同电源不同模块之间的干扰，保证了整个系统的稳定运行。

智能车系统的总的电源供应来自 7.2V 大容量镍镉电池，摄像头以及大多数芯片均为 5V 供电，单片机最小系统以及 SD 卡调试模块需要 3.3V 低压，伺服电机工作电压为 6V 左右，电机直接电池供电，但驱动电路需要 12V 的电源，因此智能车电压调节电路框架如图 3.2 所示。

3.2.1 5V 稳压电路

5V 电源模块用于为单片机系统、传感器模块和 LM1881 等芯片供电。经过比较，由于在电机驱动时电池压降较大为提高系统稳定性，必须使用低压差稳压芯片。为此我们选用了低压差线性稳压芯片 AMS1117-5.0 为 5V 工作的芯片供电。它们的纹波电压小，能对负载的变化迅速做出反应，适合为各个模块供电，故用如下图 3.3 所示。

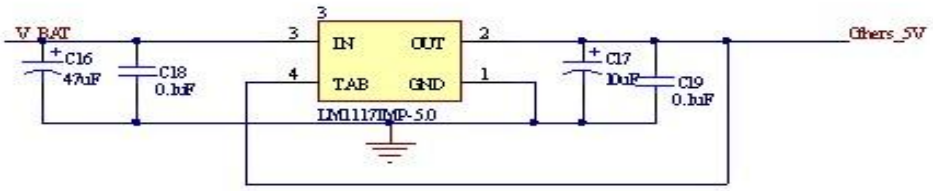

图 3.3 5V 稳压电路

3.2.2 3.3V 稳压电路

K10 最小系统及 SD 卡为 3.3V 供电。因为单片机是整个系统的核心，因此其电源的稳定性尤为重要，固这里先用 ASM1117-5.0 产生 5V，再将该 5V 输入 ASM1117-3.3 产生 3.3V 电源。这样既可以有效抑制电源纹波，又可以减小 1117-3.3 的功耗，保证其工作的稳定性。具体电路如图 3.4 所示。

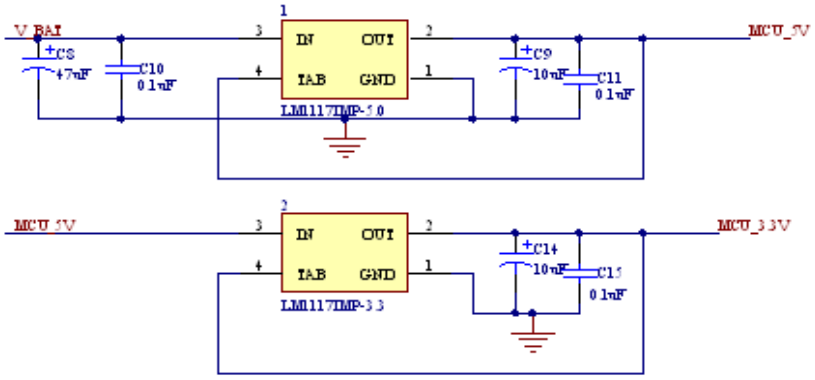

图 3.4 3.3V 稳压电路

3.2.3 12V 稳压电路

驱动电路采用 12V 的电源供电，考虑到 PCB 板的面积和布线的难易，我们经过比较采用了 5V 转 12V 稳压模块，该模块方便，且价格相对来说较便宜在电子市场上很容易买得到，只需外加简单去耦电容就可工作，最后经过我们实际检验该电路比较稳定，相对可靠。

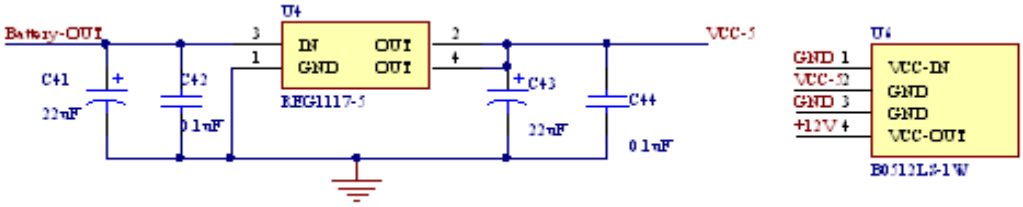

图 3.5 12V 稳压电路

3.2.4 6V 稳压电路

S3010 舵机的额定供电电压为 6V，但经测试发现，使其供电电压略高于 6V 可提升舵机的响应速度。而用一般的线性稳压电源需要一定的压差，在 7.2V 电池供电时，很难将输出电压提高。固这里使用开关电源 LM2576-ADJ 为舵机供电，调节电阻 R5、R6 比值，得到 6.5 – 6.7V 的电压为舵机供电。其电路如图 3.7 所示。

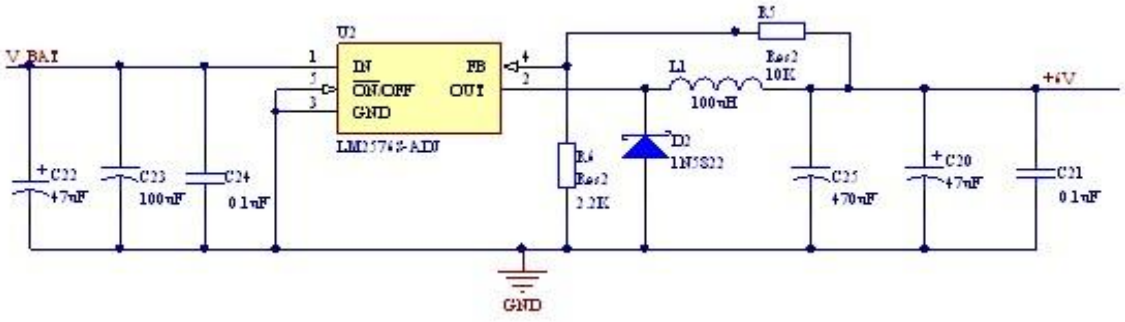

图 3.6 6V 稳压电路

3.3 摄像头采集模块

目前市场的模拟摄像头有两大种：CCD 和 CMOS。CCD 摄像头的优点是动态性能好，即使车在高速行驶时也可得到较为清晰的图像，其缺点主要有两点：耗电量大、重量大。CMOS 摄像头恰好相反，动态性能较差，高速时容易出现图像模糊的现象，但体积小，耗电小。

由于它们的特性不一致，所以在架设方式的选择上会有所差别。CCD 摄像头更适合做“低摄像头”，因为 CCD 摄像头质量大，如果架设过高可能会使重心太高；同时在实践中发现，“低摄像头”的动态模糊现象较“高摄像头”更为明显，而 CCD 高动态性能这

一特点恰好可以弥补这一缺陷。与 CCD 相反，CMOS 摄像头由于其质量轻，动态性能差的特点，更适合于“高摄像头”。

我们选择的就是 CMOS“高摄像头”的方案。主要原因是，虽然低摄像头具有更大的前瞻，但图像畸变过于严重，我们现有的技术还无法很好的处理这种畸变，导致远处的图像不仅得不到很好的利用，反而会带来各种各样的干扰，所以最终为了保证稳定与可靠性，选择了 CMOS“高摄像头”的方案。

我们所选摄像头的型号为 OV5116，输出信号为 NTSC 制式。其主要工作原理是：按一定的分辨率，以隔行扫描的方式逐点扫描，当扫描到某点时，就通过图像传感芯片将该点处图像的灰度转换成与灰度成一一对应关系的电压值，然后将此电压值通过视频信号端输出。摄像头连续地扫描图像上的一行，就输出一段连续的电压视频信号，该电压信号的高低起伏正反映了该行图像的灰度变化情况。当扫描完一行，视频信号端就输出一低于最低视频信号电压的电平，并保持一段时间，这个低电平脉冲叫做行同步脉冲，它是扫描换行的标志。然后，跳过一行后，开始扫描新的一行，直到扫描完该场的视频信号，接着就会出现一段场消隐区。此区中有若干个复合消隐脉冲，其中有个比较宽的脉冲称为场同步脉冲，它是扫描换场的标志。场同步脉冲标志着新的一场的到来，不过，场消隐区恰好跨在上一场的结尾部分和下一场的开始部分，得等场消隐区过去，下一场的视频信号才真正到来。摄像头每秒扫描 30 幅图像，每幅又分奇、偶两场，先奇场后偶场，故每秒扫描 60 场图像。奇场时只扫描图像中的奇数行，偶场时则只扫描偶数行。

在方案中，我们使用了 LM1881 视频分离芯片来辅助采样视频。LM1881 提取摄像头信号的行同步脉冲、消隐脉冲和场同步脉冲，并将它们转换成数字式电平直接输给单片机的外部中断引脚， LM1881 视频分离电路如图 3.7。

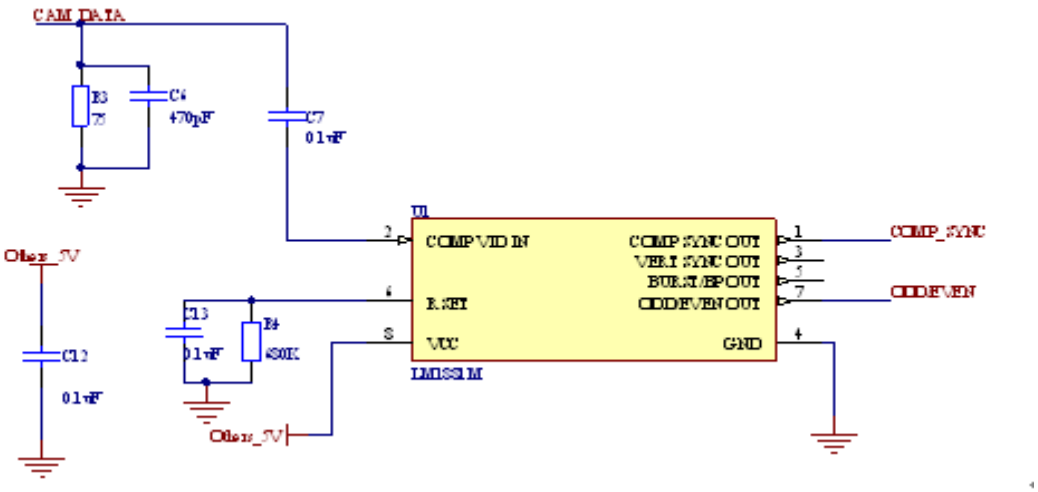

图 3.7 LM1881 电路

在图像采集方面使用了硬件二值化，相比于以前的 AD 采集方案，无论是在 CPU 资源利用上，采集时间上还是在图像分辨率上都大大提高了。具体实现是使用高速轨道轨运放 AD8032 先对原始信号进行放大，再将放大后的信号输入由 AD8032 搭成的迟滞比较器上，与设定的阈值进行比较，从而将摄像头输出的模拟信号转化为数字电平。并将

该电平分别送入 K10 的两个中断引脚上，通过采集信号的上升/下降沿来采集图像。其具体电路入股 3.8 所示。

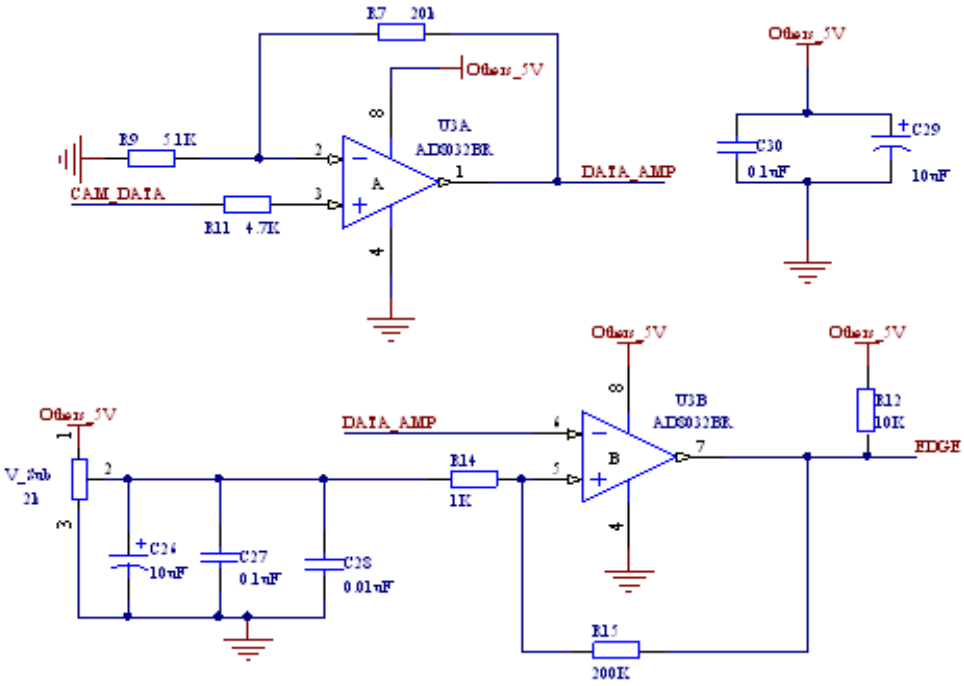

图 3.8 硬件二值化电路

3.4 电机驱动模块

驱动电路对于竞速比赛的重要性是不言而喻的，较好的加速与制动能力对小车平均速度的提高有很大帮助。

电机的速度与施加的电压成正比，输出转矩则与电流成正比。对直流电机的控制是一个挑战，因为必须在工作期间改变直流电机的速度。一般的，直流电机高效运行的最常见方法是施加一个 PWM（脉宽调制）方波，其通-断比率对应于所需速度。电机起到一个低通滤波器作用，将 PWM 信号转换为有效直流电平。PWM 驱动信号很常用，因为使用微处理器的控制器很容易产生 PWM 信号。虽然用精确的脉冲宽度可以调节电机的速度，实际应用中的 PWM 频率却是可变的，应对其进行优化，以防止电机颤抖，发出耳朵听得到的噪声。如要使直流电机反转，必须转换电机中电流的方向。

我们自己用 IR 公司的 MOSFET 管 IRLR7843 和 MOSFET 驱动芯片 IR2104 搭建了 H 桥驱动电路。IRLR7843 的最小内部只有 3.3 毫欧姆，最大允许连续源极电流 161A，超额符合要求，驱动能力很强，IR2104 外接自举电路，输出驱动电压在 10 到 20V，具有刹车使能功能，具有较好的保护和恰当的死区，外接电路简单。其具体原理图如图 3.9 所示。

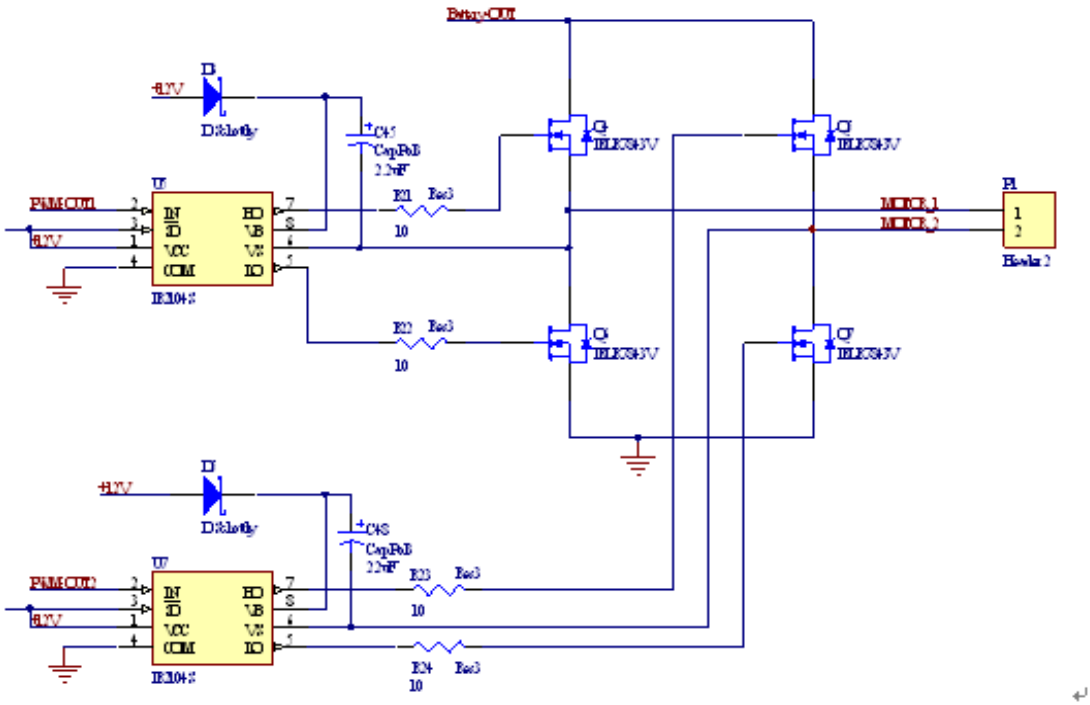

图 3.9 电机驱动电路

3.5 速度反馈模块

为了使得赛车能够平稳地沿着赛道运行，需要控制车速，使赛车在急转弯时速度不至过快而冲出赛道。通过控制驱动电机上的平均电压可以控制车速，但是如果开环控制电机转速，会受很多因素影响，例如电池电压、电机传动摩擦力、道路摩擦力和前轮转向角度等。这些因素会造成赛车运行不稳定。通过速度检测，对车模速度进行闭环反馈 PID 控制，即可消除上述各种因素的影响，使得车模运行得更稳定。

车速检测的方式有很多种，例如用测速发电机、转角编码盘、反射式光电检测、透射式光电检测和霍尔传感器检测。本次设计采用了日本 NIDEC NEMICON 公司的光电编码器，其内部是由光电对管加光栅码盘构成的，结构简单，稳定可靠，如图 3.10 所示，输出为 100 线，1600mv 的脉冲，需要外加信号调理电路整形成可靠的方波，调理电路就是用比较器 LM393 和外加电阻电容构成的，如图 3.11 所示，经过测试，在高速和低速情况下波形均稳定可靠，满足要求。

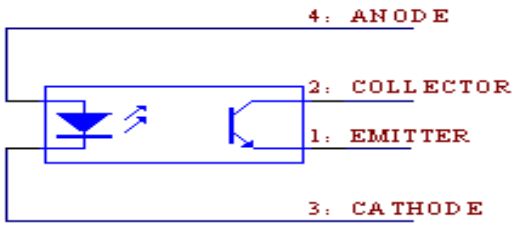

图 3.10 电编码器

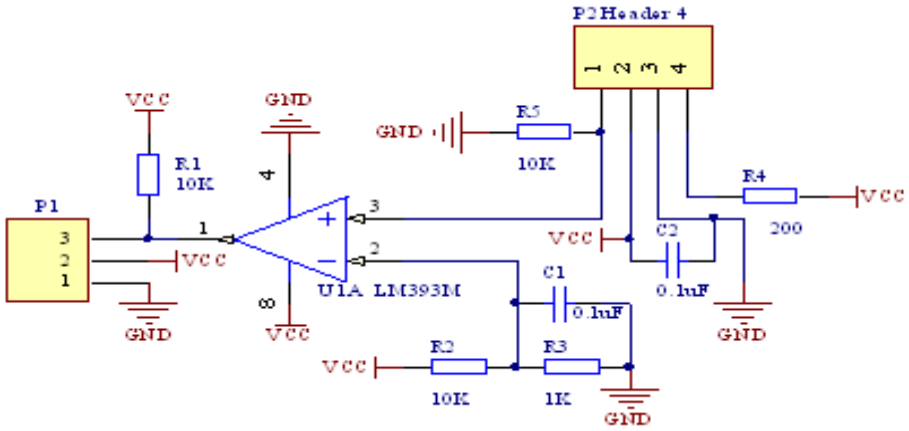

图 3.11 测速电路

3.6 调试模块

为了能够分析小车在赛道上行驶时实际看到的图像和其他一些像速度，舵机拐角等重要参数，开发了 SD 卡设备，通过 SPI 与单片机通信，实时保存重要数据，跑下一圈后，通过上位机查看图像数据，以便分析改进。为了保证该接口与 NRF2401 无线调试模块的接口兼容，方便以后扩展，这里将 SD 卡接口设置为 8 针接口。其电路如图 3.12 所示。

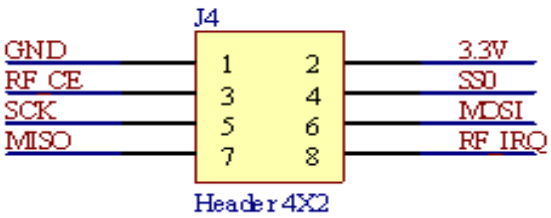

图 3.12 SD 卡调试接口

第四章　智能车软件设计方案

4.1 控制系统整体程序架构

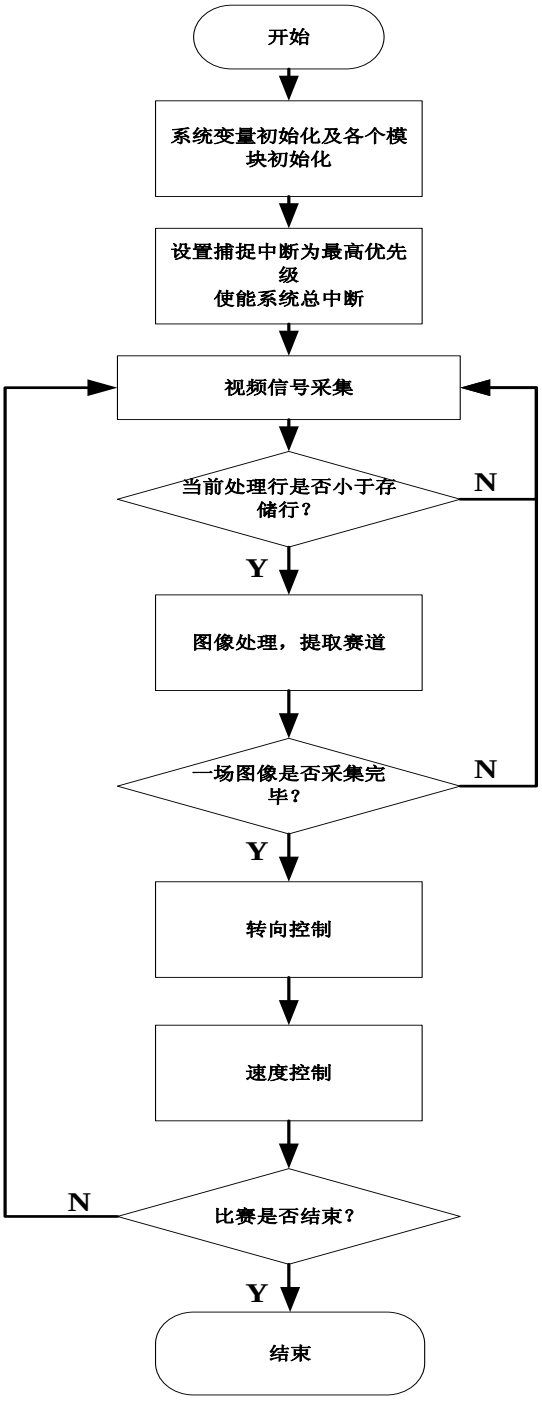

图 4.1 系统程序框架

程序框架如图 4.1，底层函数都以模块化形式编写，并且尽量进行了最优化的调整，使整体程序高效稳定。

4.2 系统初始化

4.2.1 超频设置

我们的 K10 系统板由 16M 外部晶振作为外部时钟，对其进行倍频处理已得到系统时钟。官方推荐的 K10 最大内核时钟为 100MH，但为了提高处理速度，增加算法的灵活性，我们进行了超频处理，将内核时钟设置为了 120MHz，同时将总线时钟设置为 60MHz，FLASH 时钟设置为 30MHz。

4.2.2 摄像头初始化

摄像头的信号在连接到比较器电路的同时，也连接到 LM1881 上以产生行同步信号和场同步信号，这两个信号分别接到单片机的 PTD0 和 PTC17 两个 IO 口上，并将其设置为中断功能；对于比较器输出的二值信号，接到单片机的 PTA5 和 PTE26 引脚上，同样是通过中断功能对信号进行采集。初始化代码如下所示。

```
/*****************************************************************************
* IntPoint_Init
*  初始化上升(PTE26)/下降(PTA5)沿中断
*****************************************************************************/
void IntPoint_Init(void)
{
   PORTA_PCR5 = PORT_PCR_MUX(1) | PORT_PCR_PE_MASK;
   PORTE_PCR26 = PORT_PCR_MUX(1) | PORT_PCR_PE_MASK;
/*  使能下拉电阻，PTA5 为下降沿中断,PTE26 为上升沿中断  */
   (void)EnableInt_Kinetis(87);                    /*  开启对应的中断  */
   (void)EnableInt_Kinetis(91);
}
/*****************************************************************************
* IntLine_Init
*  初始化行中断(PTD0)
*****************************************************************************/
void IntLine_Init(void)
{
   PORTD_PCR0 = PORT_PCR_MUX(1) | PORT_PCR_PE_MASK;
/*  使能下拉电阻，该中断将被设置为上升沿触发,其设置过程在使能中断的同时完成  */
```

```
  (void)EnableInt_Kinetis(90);                    /* 开启对应的中断 */
}
/*****************************************************************************
* IntField_Init
* 初始化场中断(PTC17)
*****************************************************************************/
void IntField_Init(void)
{
  PORTC_PCR17 = PORT_PCR_MUX(1) | PORT_PCR_PE_MASK;
/* 使能下拉电阻，该中断将被设置为任意沿触发,其设置过程在使能中断的同时完成 */
  (void)EnableInt_Kinetis(89);                    /* 开启对应的中断 */
}
```

4.2.3 PWM 初始化

为了控制转向舵机与电机的正反转，需要 3 路 PWM 输出。这里将 PTD4 引脚和 PTD6 引脚分别设置为 FTM0_CH4 功能和 FTM0_CH6 功能，同时将 FTM0 设置为边沿对齐 PWM 模式，以控制电机的正反转；将 PTA12 设置为 FTM1_CH0 功能，以输出 50Hz 的 PWM 波控制转向舵机。其具体初始化代码如下：

```
/*****************************************************************************
* Motor_PWM_Init
* 初始化电机正反转对应的 PWM 通道(PTD4,FTM0_CH4 对应正转,PTD6,FTM0_CH6
* 对应反转)
* (PWM 时钟设为 7.5MHz,其频率与总线时钟相关)
* 入口参数   pwm_limit:电机 PWM 的最大值
*****************************************************************************/
void Motor_PWM_Init(INT16U pwm_limit)
{
  SIM_SCGC6 |= SIM_SCGC6_FTM0_MASK;       /* 使能 FTM0 的时钟 */

  FTM0_MODE |= FTM_MODE_WPDIS_MASK;   /* 禁用写保护 */
  FTM0_SC &= ~FTM_SC_CLKS_MASK;             /* 先将 CLK[1:0]设为 0:0 */
  /* !!! 很多寄存器只有在这种状态下才能写入 !!! */

  FTM0_COMBINE = 0;
  /* DECAPEN=0,双边沿捕捉禁止,COMBINE=0,不级联 */
```

```
    FTM0_QDCTRL &= ~FTM_QDCTRL_QUADEN_MASK;   /*  设置 QUADEN = 0 */

    FTM0_C4SC = FTM_CnSC_MSB_MASK | FTM_CnSC_ELSB_MASK;
    /*  边沿对齐 PWM 模式(MSB = 1, ELSB:ELSA = 10) */
    FTM0_C6SC = FTM_CnSC_MSB_MASK | FTM_CnSC_ELSB_MASK;

    FTM0_MODE &= ~1;                                  /* FTM0 使能  */
    FTM0_OUTMASK &= ~(FTM_OUTMASK_CH4OM_MASK
                      | FTM_OUTMASK_CH6OM_MASK);
                                                      /*  通道 4、6 输出  */

    FTM0_MOD = pwm_limit - 1;                         /*  计数终值  */
    FTM0_CNTIN = 0;                                   /*  计数初值  */
    FTM1_CNT = 0;
    FTM0_C4V = 0;
    FTM0_C6V = 0;

    FTM0_SC = (FTM_SC_CLKS(1) | FTM_SC_PS(3)) & (~FTM_SC_CPWMS_MASK);
    /*  选择 BusClock 时钟,8 分频,CPWMS = 0 */

    PORTD_PCR4 = PORT_PCR_MUX(4);
    /*  将 PTD4 设为 FTM0_CH4,输出正转对应的 PWM */
    PORTD_PCR6 = PORT_PCR_MUX(4);
    /*  将 PTD6 设为 FTM0_CH6,输出反转对应的 PWM */
}
/*****************************************************************************
* Servo_PWM_Init
*  初始化舵机的 PWM 通道(PTA12,FTM1_CH0)
* (PWM 周期 50Hz,其初始化参数与总线时钟相关)
*  入口参数    pwm:舵机 PWM 的初始值
*****************************************************************************/
void Servo_PWM_Init(INT16U pwm)
{
    SIM_SCGC6 |= SIM_SCGC6_FTM1_MASK;          /*  使能 FTM1 时钟  */

    FTM1_MODE |= FTM_MODE_WPDIS_MASK;     /*  写保护禁止  */
```

```
  /*  设置通道 0,工作在左边沿对其 PWM 模式(MSB = 1, ELSB:ELSA = 10) */
  FTM1_C0SC = FTM_CnSC_MSB_MASK | FTM_CnSC_ELSB_MASK;
  左边沿对齐 PWM 模式(MSB = 1, ELSB:ELSA = 10) */

  FTM1_SC = (FTM_SC_CLKS(1) | FTM_SC_PS(5)) & (~FTM_SC_CPWMS_MASK);
  /*  选择 BusClock 时钟,32 分频后,CPWMS=0,递增计数  */

  FTM1_MODE &= ~1;                                          /* FTM1 使能  */
  FTM1_OUTMASK &= ~FTM_OUTMASK_CH0OM_MASK; /*  通道 0 输出  */
  FTM1_QDCTRL &= ~FTM_QDCTRL_QUADEN_MASK;     /*  禁止正交解码模式  */
  FTM1_COMBINE = 0;
  /* DECAPEN=0,双边沿捕捉禁止,COMBINE=0,不级联  */

  FTM1_CNTIN = 0;                                  /* FTM1 计数器初始值为 0 */
  FTM1_MOD = 37499;
  /*  结束值,周期为（MOD-CNTIN+1）*时钟周期  = 20ms */
  FTM1_C0V = ((pwm) * 3L) >> 2L;                  /*  设置占空比  */
  FTM1_CNT = 0;

  PORTA_PCR12 = PORT_PCR_MUX(3);
  /*  设置引脚 PTA12 引脚为 FTM1_CH0 功能  */
}
```

4.2.4 测速模块初始化

本系统测速方案采用了光电编码器测速，在固定周期(5ms)里读取脉冲数。定时采用了 PIT 定时器，使系统每 5ms 发生一次中断；脉冲采集使用了 Lptmr 计数器，将其设置为脉冲累计模式，并使能了数字滤波功能，以防止电磁干扰带来的测速不稳定。其具体初始化代码如下。

```
/*****************************************************************************
* IntTick_Init
*  初始化系统时钟中断
*****************************************************************************/
void IntTick_Init(void)
{
```

```
    SIM_SCGC6 |= SIM_SCGC6_PIT_MASK;    /*  使能 PIT 的时钟  */

    (void)EnableInt_Kinetis(68);                /*  开启对应的中断  */
    PIT_MCR = 0;                                /*  开启时钟,在 DEBUG 时使能 PIT */
    PIT_LDVAL0 = CAR_Tick_ms * 1000L * CAR_Fbus;        /*  计数初值  */
    PIT_TCTRL0 = PIT_TCTRL_TIE_MASK;                    /*  使能 PIT 中断  */
    PIT_TCTRL0 |= PIT_TCTRL_TEN_MASK;                   /*  开启计数器  */
}
/*************************************************************************
* SpeedCounter_Init
*  初始化测速计数器(加入滤波功能,支持的最大频率为 500k)
*************************************************************************/
void SpeedCounter_Init(void)
{
    PORTE_PCR17 = PORT_PCR_MUX(6) | PORT_PCR_PE_MASK;
/* PTE17 设为脉冲输入功能,并使能下拉电阻  */

    SIM_SCGC5 |= SIM_SCGC5_LPTIMER_MASK;      /*  使能 Lptmr 的时钟  */
    LPTMR0_CSR &= ~LPTMR_CSR_TEN_MASK;      /*  设置前先关闭计数器  */

    LPTMR0_CMR = LPTMR_CMR_COMPARE(5000);
    LPTMR0_PSR = LPTMR_PSR_PCS(0x3);
/*  选择 glitch filter 的时钟为外部晶振，同时使能 glitch filter */
    LPTMR0_PSR |= LPTMR_PSR_PRESCALE(3);
/*  设置输入时钟必须大于 8 个基准时钟，16M 晶振时,对应脉宽约 0.5us */

    LPTMR0_CSR = LPTMR_CSR_TMS_MASK | LPTMR_CSR_TPS(3);
    /*  选择脉冲计数模式,通道 3 输入*/
    LPTMR0_CSR |= LPTMR_CSR_TEN_MASK;         /*  开启计数器  */
}
```

4.3 视频采集与图像处理

4.3.1 视频信号采集与处理的时间安排

由于采用了硬件二值化方案，图像黑白比较分明，赛道信息比较清晰，单片机采集

到的原始信息如图 4.2。

图 4.2 二值化图像

因为采用的模拟摄像头每秒钟会传送 60 场画面到控制器，所以要求控制器要妥善处理数据采集和数据处理之间的时间安排，保证尽可能多的采集数据并在一场时间内处理完毕（一般为 16.67ms）。

考虑到数据的采集主要在中断中完成，且是采用隔行扫描的方式，软件的图像数据处理中设置了两个标志位，一位为数据采集标志位，担当数据采集的数据指针，另一位为数据处理标志位，担当数据处理的数据指针。实现了图像采集与图像处理同步进行，通过软件测量表明在数据采集完成后仅需不到一行的时间就可完成数据处理任务，图像数据处理软件部分流程图如图 4.2 所示。

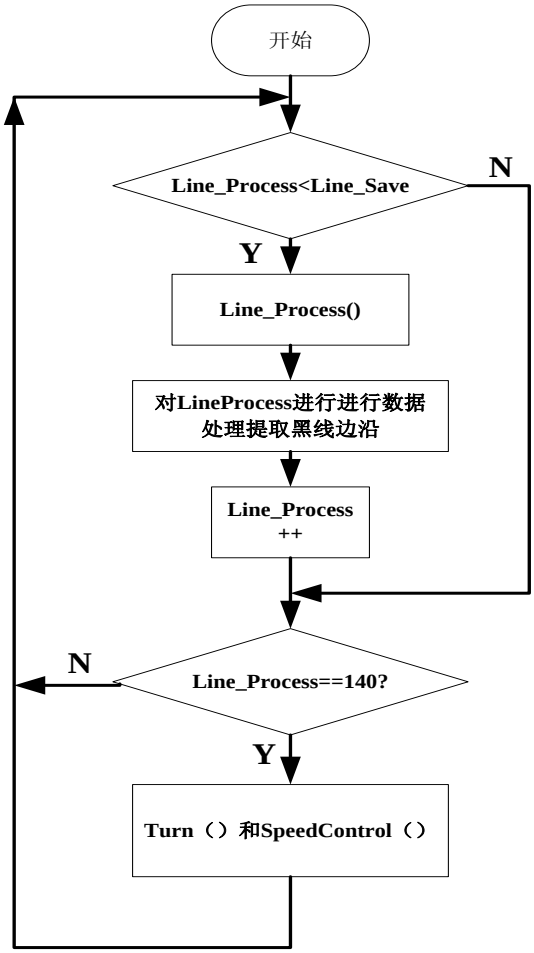

图 4.3 数据处理流程图

在图 4.3 中，Line_Save 变量负责数据的采集工作，表示已经采集的行数，它的变化有视频的场中断信号和行中断信号负责，场中断信号将其清零，而行中断信号负责累加。Line_Process 变量负责数据的处理工作，表示数据待处理的行数，它也由场中断信号负责清零，但它的累加由软件来完成，通过 Line_Process 与 Line_Save 之间的关系来实现数据采集与数据处理的同步。

4.3.2 轨迹信息提取

对轨迹的信息提取采用双边缘跟踪法，同时跟踪左右双边缘的变化。搜索时采用窗口算法，分三步完成数据，其总体流程入图 4. 4 所示。

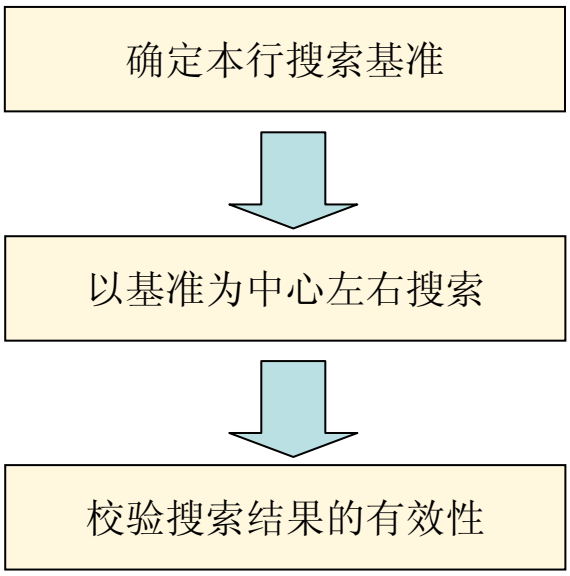

图 4.4 信息提取程序流程图

首先确定本行的搜索基准，其原则如下，如果本场已找到两个连续的有效行，那么就以上一行的边沿坐标作为搜索基准；否则，以上一场的坐标作为搜索基准。

确定搜索基准后，建立一个窗口，在窗口的左右搜索符合条件的跳变，如果没有找到，认为本行无效；如果找到一个有效跳变，将其记为本行的赛道边沿；如果找到多个有效跳变，则选与搜索基准最接近的一个为本行赛道边沿。

找到赛道边沿后，利用预计的赛道宽度对其进行进一步判定，其规则如下：如果两侧边沿均找到，则两边沿只差（既赛道宽度）与预期宽度的差值不能太大，否则判为无效；如果仅一边有效，则该边沿距离图像边缘的差值必须小于最大可能宽度。

4.4 控制策略

4.4.1 经典 PID 控制介绍

PID 控制是工程实际中应用最为广泛的调节器控制规律。问世至今 70 多年来，它以其结构简单、稳定性好、工作可靠、调整方便而成为工业控制的主要技术之一。单位反馈的 PID 控制原理框图如图 4.5：

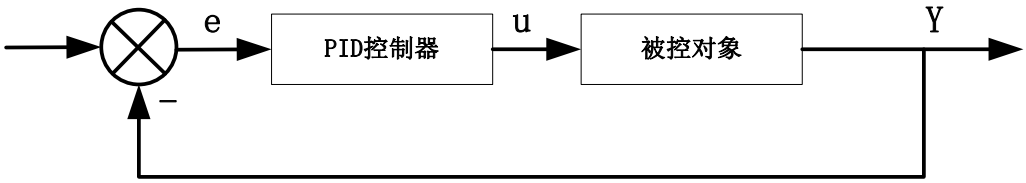

图 4.5 单位反馈的 PID 原理图

单位反馈 e 代表理想输入与实际输出的误差，这个误差信号被送到控制器，控制器算出误差信号的积分值和微分值，并将它们与原误差信号进行线性组合，得到输出量 u。

计算公式如公式 4-1 所示：

$$u = k_p e + k_i \int e dt + k_d \frac{de}{dt} \quad (4\text{-}1)$$

其中 k_p，k_i，k_d 分别称为比例系数、积分系数、微分系数。u 接着被送到了执行机构，这样就获得了新的输出信号 u ,这个新的输出信号被再次送到感应器以发现新的误差信号，这个过程就这样周而复始地进行。

PID 各个参数作用基本介绍：

增大微分项系数可以加快动态系统响应，但容易引起震荡。一般增大比例系数能够减小上升时间，但不能消除稳态误差。增大积分系数能够消除稳态误差，但会使瞬时响应变差。增大微分系数能够增强系统的稳定特性，减小超调，并且改善瞬时响应。

对连续系统中的积分项和微分项在计算机上的实现，是将上式转换成差分方程，由此实现数字 PID 调节器。

位置式 PID 控制算法：

用矩形数值积分代替上式中的积分项，对导数项用后向差分逼近，得到数字 PID 控制器的基本算式（位置算式）如公式 4-2：

$$u_n = k_p (e_n + \frac{1}{T_i} \sum_{k=1}^{n} e_k T + T_d \frac{e_n - e_{n-1}}{T}) \quad (4\text{-}2)$$

其中 T 是采样时间，k_p，T_i，T_d 为三个待调参数，在实际代码实现算法时，处理成形式如公式 4-3：

$$\mathrm{Pr}eU = k_p * error + k_i * Integral + k_d * derror \quad (4\text{-}3)$$

增量式 PID 控制算法

对位置式加以变换，可以得到 PID 算法的另一种实现形式（增量式），如公式 4-4：

$$\Delta u_n = u_n - u_{n-1} = k_p[(e_n - e_{n-1}) + \frac{1}{T_i} e_n + \frac{T_d}{T}(e_n - 2e_{n-1} + e_{n-2})] \quad (4\text{-}4)$$

我们在实现代码时，处理成形式如公式 4-5 所示：

$$\mathrm{Pr}eU += (k_p * d\_error + k_i * error + k_d * dd\_error) \quad (4\text{-}5)$$

运用 PID 控制的关键是调整三个比例系数，即参数整定。PID 整定的方法有两大类：一是理论计算整定法。它主要是依据系统的数学模型，经过理论计算确定控制器参数。由于智能车整个系统是机电高耦合的分布参数系统，并且要考虑赛道具体环境，要建立精确的智能车运动控制数学模型有一定难度，而且我们对车身机械结构经常进行不断修正，模型参数变化较频繁，可操作性不强；二是工程整定方法，它主要依赖工程经验，直接在控制系统的试验中进行，而且方法简单[14]。所以实际调车过程中，采用了第二种方法。

4.4.2 经典 PID 算法应用在智能车的速度控制上

通过光电编码器获取的当前速度值来调整电机的 PWM 占空比，可以实现对于速度的闭环控制。这样做改变了通过直接设置 PWM 占空比调整电机转速的开环控制方法，通过对速度的闭环控制，去掉电源电压和车身重量对车速的影响，采取了最可靠的方法，保证赛车各段速度较为稳定。

电机控制主要要求提高电机的响应速度和调速准确性，故选用 PID 参数时选取较大的 P 参数，而积分参数 I 对车速控制有惯性，影响反应速度，而积分参数 I 过大会使速度波动增加，影响车辆的稳定运行，所以选择了非常小的积分参数。经过多次试验，小车最终采用增量式 PID 控制，效果比较理想，小车在赛道上调速平稳，响应较快。在直道上车速平稳，入弯、出弯时加减速响应较快，比较好的完成速度的控制。具体的在单片机上实现代码跟据公式 4-3 得如下代码：

```
/*****************************************************************************
* Speed_PID
*  增量式速度 PID 控制
*****************************************************************************/
static void Speed_PID(void)
{
   INT32S Pwm_Delta;
/*简化 PID 算式(实际上就是增量式 PID 合并同类项得到的)*/
   Pwm_Delta=(KP_Speed+KI_Speed+KD_Speed)*((INT32S)SpeedErr);
   Pwm_Delta-=(KP_Speed+2*KD_Speed)*((INT32S)His_SpeedErr);
   Pwm_Delta+=KD_Speed*((INT32S)His2_SpeedErr);
   MotorPWM += (INT16S)(Pwm_Delta/20L);
/*设定电机 PWM*/
   SetMotorPWM(MotorPWM);
}
```

4.4.3 经典 PID 算法应用在智能车的转向控制上

转向舵机的控制采用了位置式 PD 控制。在控制中关键部分为目标位置的确定。

我们采用图像的加权平均偏差加图像斜率的控制方法。加权平均偏差通过每一行跑道的中心坐标的加权平均值减去图像中心获得。对于权值的选取，远端权值越大，但远端权值并不能太大，一是远端图像的可靠性不如近端，二是对于太急的弯道，过早贴内反而不利于转向，所以我们选择远端斜率略大于近端。对于斜率，我们使用最小二乘法分别计算左右两端斜率，然后求平均值作为图像斜率，加入到控制中。斜率反应了图像的变化趋势，有利于提前转向，并且有利于 S 弯道时的路径控制。

对于 PD 系数的选择，系数太大直道容易震荡，系数太小，弯道不够贴内。所以我们使用根据有效行数动态调节 PD 系数的算法：当弯道时，有效行较少，将比例系数增大；直道时，有效行较少，将比例系数减小。这样可同时兼顾弯道和直道的路径。

4.4.4 具体细节控制

控制策略 1——十字弯识别：

十字弯处，交叉处为全白色，这使得后面的边沿很难提取出来。为了可靠提取出全部信息，需要对十字弯进行识别。识别方法是利用交叉处图像全白的特点，检测到连续多行图像全白既认为是十字弯。其具体代码如下。

```
if(Num_Point == 1)
{ /*  由于有消隐区的存在,全白时图像有且仅有一对边沿  */
   if(
       v_point_up[Line_Process][0] < LEFT_Image + 3 &&
       v_point_down[Line_Process][0] > RIGHT_Image – 3
      )
   { /*  此时图像中唯一的一对边沿一定在视野的最边上  */
      Cnt_Cross++;
   }
   else
   {
      Cnt_Cross = 0;
   }
}
else
{
   Cnt_Cross = 0;
}
```

控制策略 2——D 形弯加速策略：

在实际调车过程中，发现由于采用大角度的转弯减速策略，赛车在过 D 形弯时，D 形弯后半圆，车的速度非常的慢，而一旦减速后再加起速来又需要一段时间。这样就使赛车过 D 形弯时，用时非常多。考虑到 D 形弯时一旦进弯车的转角比较稳定，在看到直道前可以提前适当加速，从而减少过弯时间。

根据 D 形弯的具体特点——长直道入弯，再到长直道，判断长直道的方法是：以当前采集到得信息为参考点，如果往前第十幅图像采集到的是弯道，在前 5 幅图像时是弯

道，当前图像又是弯道，则认为是进入 D 形弯。适当选择图像就可以准确判断 D 形弯，在实际调车过程中证明，这种判断非常准确。D 形弯如图 4.6 所示：

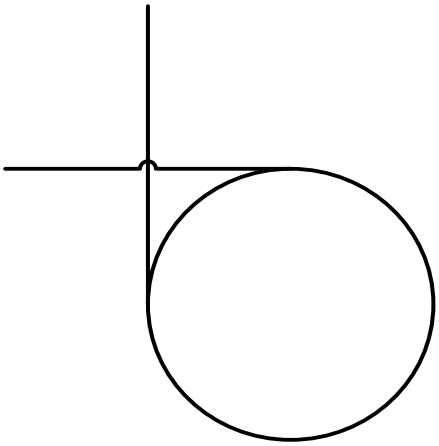

图 4.6 D 形弯

控制策略 3——起跑线检测：

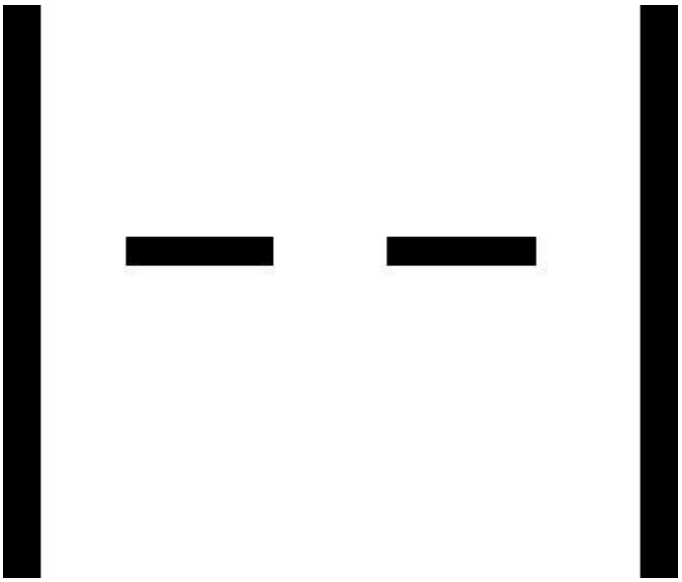

图 4.7 起跑线

如图 4.7 起跑线是黑、白、黑、白、黑、白、黑分布的，根据这一特点我们在程序里设置了 start_line[]数组，用于反映图像中黑点的分布情况。具体方法是纵向数点，将某一纵列中黑点的个数存放在 start_line[]数组中，然后检测 start_line[]数组中黑点的横向分布情况，如果满足黑、白、黑、白、黑、白、黑的分布，并且每一块的长度都在允许的范围内，则认为是起跑线。在实际调试过程中发现，只要选择合适的阈值，这种检测方法比较准确。

第五章 系统调试

5.1 编译开发环境

对于 Kinetis 系列处理器的编程开发，主要有两种开发工具：CodeWarrior 10.1(以下简称 CW10.1)和 IAR For ARM(以下简称 IAR)。

CW10.1 编译器是基于 Eclipse 的集成开发环境，它提供了高度可视化操作及自动创建复杂嵌入式系统应用的功能，编辑界面十分友好，可以对程序的阅读与开发带来很多便利。但正是因为它有如此强大的功能，所以运行时需要占用 PC 机更多的资源，包括软件启动在内的很多操作都执行的很慢，而且不具有变量实时查看功能，不利于系统的在线调试。

而 IAR 编译器编辑界面相对简单，虽然在操作的可视化程度上较 CW10.1 略逊一筹，但它消耗 PC 机的资源较少，同时它具有的变量实时查看功能，会对调试但来很多便利。

所以我们将两个开发环境结合起来，建立了一个同时兼容两种编译器的工程模版，在程序开发阶段使用 CW10.1，而在调试阶段使用 IAR，以提高开发效率。

5.2 SD 卡模块

智能车在高速运行的时候每个 16.67ms 控制周期内的各个参数量都是不一样的，显然要想知道各个参量的变化情况，甚至只是想知道当前的速度这都是不现实的，所以我们用 SD 卡把摄像头的采集到的赛道图像和一些重要参数实时存储下来是一种很好的方案，大大方便了智能车的调试过程。

5.2.1 SD 卡介绍

SD 卡(Secure Digital Memory Card)是一种基于半导体快闪存的新一代记忆设备。由日本松下、东芝及美国 SanDisk 公司于 1999 年 8 月共同开发研制，其大小犹如一张邮票，重量只有 2g，却拥有高记忆容量、快速数据传输率、极大的移动灵活性以及很好的安全性。SD 卡的数据存储管理可以类似于硬盘的磁盘管理系统，以 FAT 的格式来存储数据。SD 卡的接口支持 SD 模式和 SPI 模式，主机系统可以选择其中任一模式。SPI 模式允许简单通用的 SPI 通道接口，这种模式相对于 SD 模式的不足之处是降低了速度。由于飞思卡尔系列单片机拥有 SPI 接口，所以使用 SD 卡的 SPI 模式。

5.2.2 SPI 总线的介绍

SPI（Serial Peripheral Interface，串行外围设备接口总线）总线技术是 MOTOROLA

公司推出的一种同步串行总线接口，它是目前单片机应用系统中最常用的几种串行扩展接口之一。SPI 总线主要通过三根线进行数据传输：同步时钟线 SCK, 主机输入/从机输出数据线 MISO、主机输出/从机输入数据线 MOSI,另外还有一条低电平有效的从机片选择线 CS。SPI 系统的片选信号以及同步时钟脉冲由主机提供。SPI 总线模式的数据是以字节为单位进行传输的，每字节为 8 位，每个命令或者数据块都是字节对齐的(8 个时钟的整数倍)。主机与 SD 卡的各种通信都由主机控制，主机在对 SD 卡进行任何操作前都必须先要拉低 SD 卡的片选信号 CS(CardSelect) ，然后由主机向 SD 卡发送命令，SD 卡对主机发送的任何命令都要进行响应，不同的命令会有不同的响应格式(1 字节或 2 字节响应)。SD 卡除了对命令响应外，在执行写操作时，还要对主机发送的每个数据块进行响应(向主机发送一个特殊的数据响应标志)。

5.2.3 SD 卡的软件操作

SD 卡与单片机的硬件链接如图 3.14 所示，选用 PM 口的 SPI 模式，将其与 SD 卡连在一块。

SD 卡的软件设计主要包括两部分内容：SD 卡的上电初始化过程和对 SD 卡的读写操作。对 SD 卡初始化步骤如图 5.1 所示：

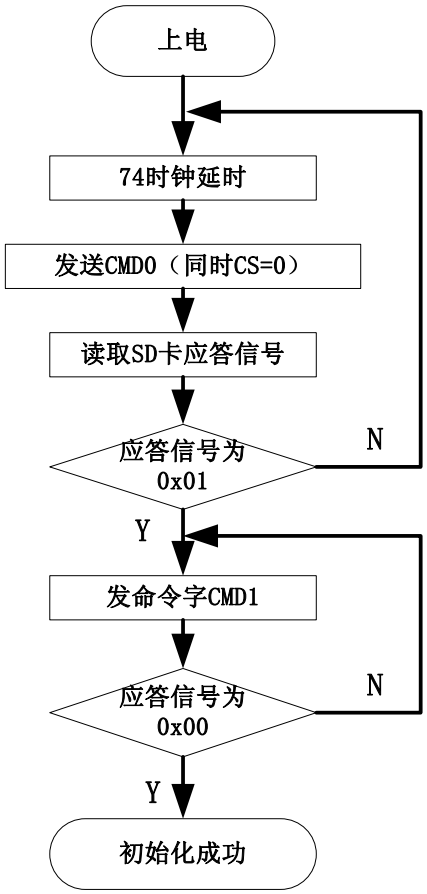

图 5.1 SD 卡初始化流程图

SD 卡上电后，主机必须先向 SD 卡发送 74 个时钟周期，以完成 SD 卡上电过程。

SD 卡上电后会自动进入 SD 总线模式，并在 SD 总线模式下向 SD 卡发送复位命令(CMD0)，若此时片选信号 CS 处于低电平态，则 SD 卡进入 SPI 总线模式，否则 SD 卡工作在 SD 总线模式。SD 卡进入 SPI 工作模式会发出应答信号，若主机读到的应答信号为 01,即表明 SD 卡已进入 SPI 模式,此时主机即可不断地向 SD 卡发送命令字(CMD1)并读取 SD 卡的应答信号，直到应答信号为 00，以表明 SD 卡已完成初始化过程，准备好接受下一命令。

此后，系统便可读取 SD 卡的各寄存器，并进行读写等操作，每次读写数据都是按照扇区操作的，每次操作 512 字节。通过不断的测试，在 SPI 时钟为 16M 时 SD 卡的读写正常，并且没有影响单片机正常工作。具体流程图如图 5.2 所示：

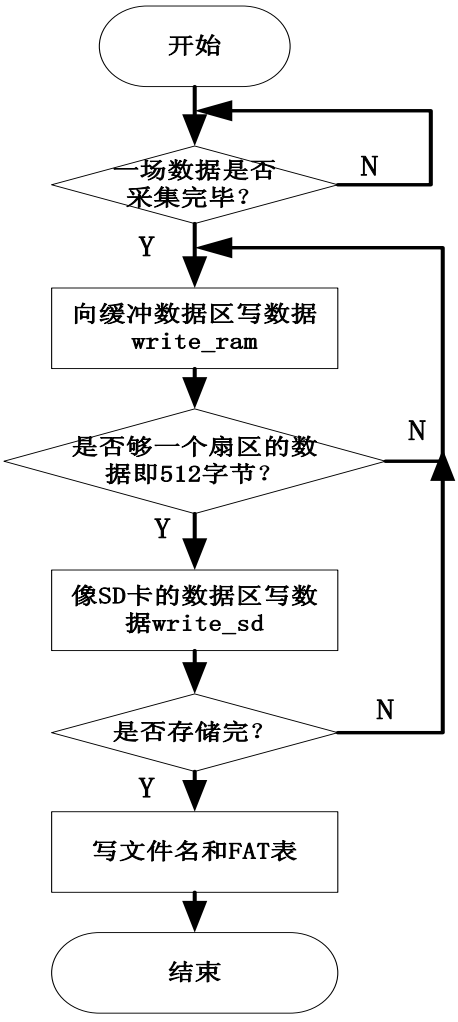

图 5.2 SD 卡读写流程图

5.2.3 上位机软件调试

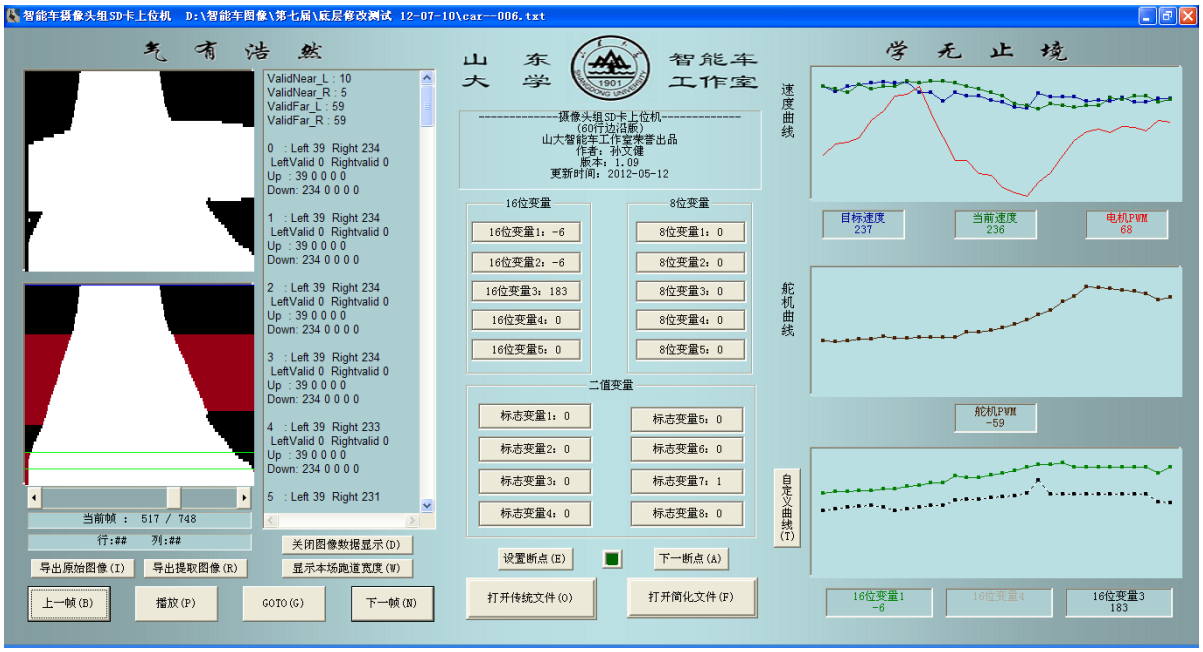

图 5.3 上位机界面

如图 5.3 所示上位机软件由 VC++编写，实现了在上位机上对 SD 卡中数据的读取和显示，将采集到的原始数据、提取的赛道、速度曲线等信息显示出来，大大方便了对智能车的调试和分析，对小车上摄像头实际看到的图像有了直观的认识，进而较好以车的角度分析赛道编写算法，这种调试方法使智能车的调试效率大大提高。

第六章 模型车的主要技术参数

赛车基本参数	长	27.5cm
	宽	22.0cm
	高	35.0cm
车重		1.3kg
功耗	空载	8.0W
	带载	〉15W
电容总容量		<1500uF
传感器	CMOS摄像头	1个
	100线光电编码器	1个
	陀螺仪ENC-03	1个
除了车模原有的驱动电机、舵机之外伺服电机个数		0个
赛道信息检测	视野范围（近/远）	16/146cm
	精度(近/远)	3/11.5mm
	频率	60Hz

结 论

本文详细介绍了基于 CMOS 摄像头寻迹的智能车系统方案。以飞思卡尔半导体公司生产的 Kinetis 系列 32 位处理器 PK10N512 为核心控制器，采用基于摄像头的图像采样获取赛道图像信息，通过窗口算法提取赛道，求出小车与赛道的位置偏差，采用 PID 方式对舵机转向进行反馈控制。通过速度传感器对小车形成速度闭环的 PID 控制。

文中分章节分别介绍了小车车模的改装、机械结构的调整和优化，赛车各个硬件模块的工作原理、设计思路和改进方案，并且详细的叙述了整个智能车系统的开发工具，软件和各个调试模块的设计方法和使用方法。

在硬件方面包括了电源分模块供电、光电编码器测速和 H 桥驱动电路等部分，软件方面包括边沿检测算法，窗口算法，舵机及电机 PID 调节和 SD 卡上位机下位机等部分。

总结整个设计过程，不仅使我们得到了对已有知识进行实践的机会，更培养了一定的科研能力，拓宽了知识面，同时我们小组在智能车制作的过程中学会了发现问题，分析问题和解决问题的能力，我们大家互相配合，锻炼了团队精神。展望未来，智能车技术在以飞思卡尔杯智能汽车竞赛为背景下必然会有广泛的应用。

参考文献

[1] jicheng0622 博客. http://blog.chinaaet.com/jihceng0622

[2] 黄开胜，金华民，蒋狄南.韩国智能模型车技术方案分析[J].电子产品世界，2006,3:150~152

[3] 孙浩，程磊，黄卫华.基于 HCS12 的小车智能控制系统设计及机械调整方法[J].单片机与嵌入式系统应用，2007,3:51~57

[4] 孙涵，任明武，唐振民等.基于机器视觉的智能车辆导航综述[J].公路交通科技，2005,22(5):132~135

[5] Freescale Semiconductor. MC9S12XS256RMV1 Datasheet[S]

[6] 孙同景，陈桂友.Freescale 9S12 十六位单片机原理及嵌入式开发技术[M].北京：机械工业出版社，2008.5

[7] 薛涛，宫辉，曾鸣等.单片机与嵌入式系统开发方法[M].北京：清华大学出版社，2009.10

[8] 黄萍莉，岳军.图像传感器 CCD 技术[J].信息记录材料，2005,6(1):50~54

[9] 宋敏，郤新凯，郑亚茹.CCD 与 CMOS 图像传感器探测性能比较.半导体光电，2005,26(1):5~9

[10] 卓晴，王璇，王磊.基于面阵 CCD 的赛道参数检测方法[J].电子产品世界，2006,4:141~143

[11] 葛亚明，刘涛，王宗义.视频同步分离芯片 LM1881 及其应用[J].应用科技，2004,31(9):20~22

[12] 卓晴，黄开胜，邵贝贝.学做智能车——挑战“飞思卡尔”杯[M].北京：北京航空航天大学出版社，2007,3:35~40

[13] Brown M A, Blackwell K T, Khalak H G..Multi-Scale Edge Detection and Feature Binding: An Integrated Approach[J]. Pattern recognition,1998, 31(10), 1479-1490.

[14] 刘进，齐晓慧，李永科.基于视觉的智能车模糊 PID 控制算法[J].兵工自动化，2008, 27(10)

[15] 李正军.计算机控制系统[M].北京：机械工业出版社，2005.1

[16] 唐建文.智能小车控制系统的设计与实现[D].广东：广东工业大学硕士论文，2008

[17] 孙鑫，余安萍.VC++深入详解[M].北京：电子工业出版社，2006.6

[18] 邓剑，杨晓非，廖俊卿.FAT 文件系统原理及实现[J].计算机与数字工程，2005,33(9):105~108

附录

附录 A：主控程序源代码

```
/*********************************************************************************************

* main.c

*  定义程序的入口(main 函数)及系统的主要运行流程
*  定义摄像头采样行分配表

*********************************************************************************************/

#include "../Platform/BaseTypeDefine.h"
#include "../Platform/Platform.h"
#include "../Common_Def.h"

#include "../Math/Math_Car.h"
#include "../System_Init/System_Init.h"
#include "../Hardware_Interface/Interrupt.h"
#include "../Hardware_Interface/HardwareOperation.h"
#include "../Original_Process/OriginalProcess.h"
#include "../Make_Decision/MakeDecision.h"
#include "../Make_Decision/SpeedControl.h"

LineConfig_t Tab_LineConfig[NUM_Line] =
{
   30,34,38,42,46,50,54,58,62,66,
   70,74,78,82,86,90,94,98,102,106,
   110,114,118,122,126,130,134,138,142,146,
   150,154,158,162,166,170,174,178,182,186,
   190,194,198,202,206,210,214,218,222,226,
   230,234,238,242,246,250,254,258,262,266,
};

/*********************main 函数*******************/
void main(void)
{
/*系统初始化*/
   StartSystem
    (
       Tab_LineConfig,
       ExtInit_Normal,
```

```
        ExtProcess_ReStart
    );

/*程序主循环*/
  for(;;)
  {
    if(Line_Process < Line_Save)                /*行处理*/
    {
      ProcessLine();
    }

    if(LineOverFlag != 0)                       /*场处理*/
    {
      MakeDecision();
      SysDataCopy();
      AuxiliaryFunction();
      LineOverFlag = 0;
    }

    C_WDOG_IntOn();                             /*清看门狗*/
  }
}

/*********************************************************************************************

* MakeDecision.c

*  定义控制策略的相关函数以及变量

*********************************************************************************************/

#include "./MakeDecision.h"

#include "./SpeedControl.h"
#include "../Original_Process/OriginalProcess.h"
#include "../Hardware_Interface/Interrupt.h"
#include "../Hardware_Interface/HardwareOperation.h"
#include "../Math/Math_Car.h"


/*************************全局变量定义*************************/
/******************图像基本信息变量******************/
INT16S PosErr;                  //  坐标误差
INT16S his_PosErr;              //  坐标误差历史值
```

```
INT16S Slope;                    // 图像斜率

/******************设定的参数******************/
INT16S MINspeed;   //一般赛道下的最大速度
INT16S MAXspeed;   //最小速度

/******************其它变量******************/
INT8U FinishFlag;

INT16U CntControl;      /*记录控制策略执行的次数*/

INT16U Now_CntField;   /*当前场数*/
INT16U Now_SysTick;     /*当前系统时钟*/


/**************************内部函数定义**************************/
/*****************************************************************************************
* Turn
*  舵机控制策略(控制车的转角)
*****************************************************************************************/
static void Turn(void)
{
   INT16S temp,i,flag;
   INT16S num = 0;
   INT32S sum = 0;

   //计算坐标误差
   for(i=0;i<NUM_Line;i++)
   {
      flag = 0;
      if(h_ValidFlag[i] == 0x10)
      {
         temp = h_LeftEdge[i] + EptWith[i]/2;
         flag = 1;
      }
      else if(h_ValidFlag[i] == 0x01)
      {
         temp = h_RightEdge[i] - EptWith[i]/2;
         flag = 1;
      }
      else if(h_ValidFlag[i] == 0x11)
      {
         temp = (h_LeftEdge[i] + h_RightEdge[i]) / 2;
         flag = 1;
      }
```

```
        if(flag != 0)
        {
            if(i > NUM_Line/2)
            {
                sum += temp*2;
                num += 2;
            }
            else
            {
                sum += temp;
                num++;
            }
        }
    }
    if(num != 0)
    {
        PosErr = (INT16S)(MID_Image - sum/num);
    }
    else
    {
        PosErr = his_PosErr;
    }

    //舵机 PD 控制
    SetServoPWM((INT16S)(PosErr*75L/10L + (PosErr-his_PosErr)*6L/10L) + Slope*6/10);
    his_PosErr = PosErr;
}
#if EN_ClosedLoop != 0
/*****************************************************************************************
* CalObjectSpeed
*  闭环模式下,计算正常行驶时的车体目标速度
*****************************************************************************************/
static void CalObjectSpeed(void)
{
    INT16S tmp_objectspeed;

    tmp_objectspeed = MAXspeed;
    tmp_objectspeed -= Abs(PosErr)/7;
    tmp_objectspeed -= Abs(Slope)/5;

    if(ObjectSpeed < MINspeed)
    {
        ObjectSpeed = MINspeed;
    }
```

```
    ObjectSpeed=tmp_objectspeed;
}
#else
/*****************************************************************************************
* CalSpeedPWM
*  开环环模式下,计算正常行驶时赋给电机的 PWM
*****************************************************************************************/
static void CalSpeedPWM(void)
{
    ControlSpdPWM(MAX_DCLimit/3);
}
#endif


/***************************接口函数定义**************************/
/*****************************************************************************************
* DataInit_Decision
*  初始化决策的相关变量
*****************************************************************************************/
void DataInit_Decision(void)
{
    CntControl=0;

    MAXspeed=300;
    MINspeed=200;

    his_PosErr=0;
}
/*****************************************************************************************
* MakeDecision
*  决策函数：实现舵机控制、速度设定等功能
*****************************************************************************************/
void MakeDecision(void)
{
    DIS_CarInt();
    Now_CntField=CntField;
    Now_SysTick=SysTick;
    EN_CarInt();

    C_WDOG_IntOn();    //喂狗

//执行舵机控制策略
    Turn();
```

```
//执行速度策略
   if(CntControl >= DELAY_TIM)
   {
   #if EN_ClosedLoop != 0
      if(CntControl == DELAY_TIM)
      {
         ClosedLoop = 1;
      }
      CalObjectSpeed();
   #else
      CalSpeedPWM();
   #endif
   }

   C_WDOG_IntOn();    //喂狗

#if EN_Timing_Stop != 0
   if(Now_CntField >= STOP_TIM+DELAY_TIM)
   {
      FinishFlag = 1;
      BuzzerSet(BZR_TimeStop);
   }
#endif

   if(FinishFlag != 0) StopMotor();

   CntControl++;
}
```

附录 B：图像采集程序源代码

```
/*****************************************************************************

* Interrupt.c

*  定义图像采集(二值化方式)的原始变量
*  定义图像采集(二值化方式)相关中断的设定函数
*  定义图像采集与定时器中断的服务函数
*  为系统提供时间基准(SysTick,单位 ms; CntField,单位一场图像的时间)

*****************************************************************************/

#include "./Interrupt.h"
#include "../System_Init/System_Init.h"
```

```
/*************************全局变量*************************/
/*图像变量*/
INT16S v_point_up[NUM_Line+3][MAX_Edge];      /*存放上升沿与下降沿的原始数据数组*/
INT16S v_point_down[NUM_Line+3][MAX_Edge];
INT8S   PointNum[NUM_Line];                    /*有用边沿数*/

INT16U Line_Process;             /*当前处理行*/
volatile INT16U Line_Save;   /*当前采样行*/

#if EN_Check_TimeOut != 0
   FastestTypeU_t TimeOutFlag;   /*处理超时标志*/
   INT16U           Cnt_TimeOut; /*处理超时次数*/
#endif

/*时间变量*/
volatile INT16U CntField;   /*摄像头输出的场数*/
volatile INT16U SysTick;    /*系统时间基准(其单位在 Platform.h 中配置)*/


/*************************内部变量*************************/
static LineConfig_t   *Ptr_LineCfg;                /*指向采样行分配表的指针*/

static FastestTypeU_t Flag_Down,Flag_Up;        /*当前采集边沿的下标*/
static FastestTypeU_t Flag_Pick;                /*采样标志*/

#if BIT_CPU != 8
   static FastestTypeU_t LineCCD;                 /*摄像头当前输出的行*/
   static FastestTypeU_t LineCCD_Start;           /*起始采样行*/
   static FastestTypeU_t LineCCD_End;             /*停止采样的行*/
#else
   static INT16U LineCCD;                         /*摄像头当前输出的行*/
   static INT16U LineCCD_Start;                   /*起始采样行*/
   static INT16U LineCCD_End;                     /*停止采样的行*/
#endif

#if CAR_DataSegMode != 3
   #if CAR_DataSegMode == 2
      #pragma CAR_DataSeg_Keep
   #endif
   static TimerCallBack_Spd_t TimerFun_Speed;   /*定时中断对应的速度处理回调函数*/
   static TimerCallBack_Ext_t TimerFun_Extra;   /*定时中断对应的额外处理回调函数*/
   static FastestTypeU_t Flag_SpdCallBack;      /*速度处理回调函数非空标志*/
   static FastestTypeU_t Flag_ExtCallBack;      /*额外处理回调函数非空标志*/
   #if CAR_DataSegMode == 2
```

```
      #pragma CAR_DataSeg_Normal
   #endif
#else
   CAR_Keyword_Keep static TimerCallBack_Spd_t TimerFun_Speed;
   CAR_Keyword_Keep static TimerCallBack_Ext_t TimerFun_Extra;
   CAR_Keyword_Keep static FastestTypeU_t Flag_SpdCallBack;
   CAR_Keyword_Keep static FastestTypeU_t Flag_ExtCallBack;
#endif


/***************************接口函数***************************/
/*****************************************************************************************
* Timer_Init
*  初始化定时计数器的相关硬件与变量
*****************************************************************************************/
void Timer_Init(void)
{
   SysTick = 0;
   TimerFun_Speed = NULL;
   TimerFun_Extra = NULL;
   Flag_SpdCallBack = 0;
   Flag_ExtCallBack = 0;

   IntTick_Init();          /*初始化系统时钟中断*/
   CameraTimer_Init();   /*初始化摄像头采样定时器*/
   SpeedCounter_Init(); /*测速计数器初始化*/
}
/*****************************************************************************************
* Timer_Init_Quick
*  仅初始化定时计数器的相关硬件,不对变量进行初始化
*****************************************************************************************/
void Timer_Init_Quick(void)
{
   IntTick_Init();          /*初始化系统时钟中断*/
   CameraTimer_Init();   /*初始化摄像头采样定时器*/
   SpeedCounter_Init(); /*测速计数器初始化*/
}
/*****************************************************************************************
* Camera_init
*  初始化摄像头的相关中断与变量,保证启动后的第一场图像是正确的
*  入口参数    pcfg:指定的采样行分配表
*****************************************************************************************/
void Camera_Init(LineConfig_t *pcfg)
{
/*相关变量初始化*/
```

```
    Ptr_LineCfg = pcfg;
    LineCCD_Start = Ptr_LineCfg[0];
    LineCCD_End = Ptr_LineCfg[NUM_Line-1]+1;

    LineCCD = 0;
    Line_Save = 0;
    Line_Process = NUM_Line;

    if(Flag_POR != 0)
    {
      CntField = 0;
    }

/*关闭摄像头相关中断*/
    DIS_FieldInt();
    DIS_LineInt();
    DIS_PointInt();

/*硬件初始化*/
    IntPoint_Init();              /*初始化上升/下降沿中断*/
    IntLine_Init();               /*初始化行中断*/
    IntField_Init();              /*初始化场中断*/

    CLR_LineInt();        /*清行中断与场中断标志*/
    CLR_FieldInt();
    EN_FieldInt();        /*使能场中断*/
}
/*****************************************************************************
* SetTimerCallBack_Spd
*  设定系统时钟中断的速度处理回调函数
*  入口参数    pspeed:速度处理函数的地址
*****************************************************************************/
void SetTimerCallBack_Spd(TimerCallBack_Spd_t pspeed)
{
    DIS_CarInt();

    TimerFun_Speed = pspeed;
    if(TimerFun_Speed != NULL)
    {
      Flag_SpdCallBack = 1;
    }
    else
    {
      Flag_SpdCallBack = 0;
    }
```

```
    if(Flag_Run != 0)
    {
        EN_CarInt();
    }
}
/*********************************************************************************************
* SetTimerCallBack_Ext
*  设定系统时钟中断的额外处理回调函数
*  入口参数    pextra:其它处理函数的地址
*********************************************************************************************/
void SetTimerCallBack_Ext(TimerCallBack_Ext_t pextra)
{
    DIS_CarInt();

    TimerFun_Extra = pextra;
    if(TimerFun_Extra != NULL)
    {
        Flag_ExtCallBack = 1;
    }
    else
    {
        Flag_ExtCallBack = 0;
    }

    if(Flag_Run != 0)
    {
        EN_CarInt();
    }
}


/***************************中断服务程序***************************/
#if (CAR_CodeSegMode == 2) || (CAR_CodeSegMode == 3)
    #pragma CAR_CodeSeg_IntISR
#endif

/*********************************************************************************************
* ISR_System_Tick
*  系统时钟中断的中断服务程序
*  负责读取测速计数器的值；提供系统时间基准；
*  并运行用户指定的函数
*********************************************************************************************/
ISR_System_Tick()
{
```

```
    INT16U cnt_spd;

    CLR_TickInt();

#if CAR_IntMode == 1
    EN_CarInt();
#endif

    cnt_spd=READ_Speed_Counter();        /*读取测速计数器*/
    CLR_Speed_Counter();                 /*测速计数器清零*/

#if CAR_SpdCounterMode == 1
    RST_Speed_Counter();                 /*重新开始计数*/
#endif

    SysTick++;                           /*系统时间基准加 1*/

    if(Flag_SpdCallBack != 0)            /*执行相关处理*/
    {
      TimerFun_Speed(cnt_spd);
    }
    if(Flag_ExtCallBack != 0)
    {
      TimerFun_Extra();
    }

#if CAR_SpdCounterMode == 2
    cnt_spd = 0;
    while(READ_Speed_Counter() != 0)    /*等待计数器清零成功*/
    {
      if((++cnt_spd) > CAR_SpdClearLimit) break;
    }
    RST_Speed_Counter();                 /*重新开始计数*/
#endif
}

/*****************************************************************************************
* ISR_Sample_Up
*  上升沿(即白到黑的跳变)采集中断的中断服务程序
*****************************************************************************************/
ISR_Sample_Up()
{
    INT16U cnt_up_tmp ;

#if CAR_IntMode == 2
```

```
    DIS_CarInt();
#endif

    cnt_up_tmp = READ_Camera_Timer();
    CLR_UpInt();

       if(Flag_Down - Flag_Up == 1)
     {
          v_point_up[Line_Save][Flag_Up] = cnt_up_tmp;
             Flag_Up ++;
       }

#if CAR_IntMode == 2
    EN_CarInt();
#endif
}
/*****************************************************************************************
* ISR_Sample_Down
*  下降沿(即黑到白的跳变)采集中断的中断服务程序
*****************************************************************************************/
ISR_Sample_Down()
{
    INT16U cnt_down_tmp;

#if CAR_IntMode == 2
    DIS_CarInt();
#endif

    cnt_down_tmp = READ_Camera_Timer();
    CLR_DownInt();

    if(Flag_Down - Flag_Up == 0)
    {
            v_point_down[Line_Save][Flag_Down] = cnt_down_tmp;
            Flag_Down ++;
    }

#if CAR_IntMode == 2
    EN_CarInt();
#endif
}
#endif

/*****************************************************************************************
* _INT_Field
```

```
* 场中断的中断服务程序
*********************************************************************************/
ISR_Field()
{
#if CAR_IntMode == 2
   DIS_CarInt();
#endif

   CLR_FieldInt();
   EN_LineInt();
   DIS_FieldInt();
   NEG_FieldInt();

   CntField++;

#if CAR_IntMode == 2
   EN_CarInt();
#endif
}
/*********************************************************************************************
* ISR_Line
* 行中断的中断服务程序
*********************************************************************************************/
ISR_Line()
{
#if CAR_IntMode == 2
   DIS_CarInt();
#endif

   CLR_LineInt();

   START_Camera_Timer();                    /*开启定时器*/
   DIS_PointInt();                         /*关上升/下降沿中断*/

   LineCCD++;
   if(LineCCD == LineCCD_End)               /***本场采样结束***/
   {
      LineCCD = 0;
      PointNum[Line_Save++] = (INT8U)Flag_Down;     /*记录最后一行的边沿数*/
      DIS_LineInt();                                /*关行中断*/

      CLR_FieldInt();                               /*清场中断标志*/
      EN_FieldInt();                                /*使能场中断*/
   }
   else if(LineCCD > LineCCD_Start)        /***正常采样行***/
```

```
    {
        if(Flag_Pick != 0)
        {
            PointNum[Line_Save++] = (INT8U)Flag_Down; /*记录上一行的边沿数*/
            Flag_Pick = 0;
        }
        if(Ptr_LineCfg[Line_Save] == LineCCD)           /*判定是否需要采样*/
        {
            Flag_Down = 0;
            Flag_Up = 0;
            Flag_Pick = 1;

            CLR_PointInt();                             /*清上升(下降)沿中断标志*/
            EN_PointInt();                              /*开上升/下降沿中断*/
        }
    }
    else if(LineCCD == LineCCD_Start)       /***一场的开始***/
    {
    #if EN_Check_TimeOut != 0
        if(Line_Process != NUM_Line)
        {                                               /*处理超时判定*/
            TimeOutFlag = 1;
            Cnt_TimeOut++;
        }
    #endif

        Line_Process = 0;
        Line_Save = 0;
        Flag_Down = 0;
        Flag_Up = 0;
        Flag_Pick = 1;

    #if CAR_PointIntMode == 1
        Set_PointInt_Down();
    #endif

        CLR_PointInt();
        EN_PointInt();
    }

#if CAR_IntMode == 2
    EN_CarInt();
#endif
}
```

```
#if CAR_CodeSegMode == 3
   #pragma CAR_CodeSeg_EndIntISR
#endif
```