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：大连理工大学

队伍名称：Warm

参赛队员：温建月

杜雯菁

沈　潋

带队教师：吴振宇

李　航

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：_____

目　　录

第一章 引 言

随着电子信息技术的迅猛发展，越来越多的自动化设备开始进入到人们的生产生活中，嵌入式系统的快速发展为开展智能研究提供了更广阔的平台。在工业生产、科学探索、救灾抢险、军事等方面，人工智能发挥着越来越重要的作用，在此背景下，智能控制策略变得尤为重要。

“飞思卡尔”杯全国大学生杯智能汽车竞赛是国家教学质量与教学改革工程资助项目，以飞思卡尔半导体公司生产的 16 位或 32 位单片机为核心控制模块，通过利用不同类型的传感器识别道路、设计电机驱动电路以及编写相应程序，制作一个能够自主识别道路的汽车模型。竞赛过程包括理论设计、实际制作、整车调试、现场比赛等环节，要求学生组成团队，协同工作，初步体会一个工程性的研究开发项目从设计到实现的全过程。该竞赛融科学性、趣味性和观赏性为一体，是一个涵盖自动控制、模式识别、传感技术、电子、电气、计算机、机械与汽车等多学科专业的创意性比赛，对学生的知识融合和实践能力的提高，具有良好的推动作用[1]。

本车采用第七届“飞思卡尔”杯全国大学生智能车竞赛组委会指定的汽车模型作为研究平台，以 32 位单片机 Kinetis 60 MK60DN512ZVLQ10 作为主控制单元，运用 IAR 软件作为开发工具进行智能控制策略研究。道路信息检测模块采用 CCD 传感器摄像头，CCD 在工作稳定性、分辨率、价格等方面均优于 CMOS 摄像头。

令人兴奋的是，本届比赛组委会第一次开放 32 位处理器—Kinetis 系列和 Coldfire 系列，这对研究更为复杂完善的算法提供了便利的条件。我们选择的 K60 处理器功能强大，扩展丰富。Kinetis 系列微控制器是飞思卡尔公司于 2010 年推出的基于 ARM Cortex-M4 内核的微处理器，提供了诸如支持数字信号处理，直接内存访问，多用途时钟发生器。1×8 通道 PWM 定时器，2×2 通道正交解码器，低功耗定时器等强大丰富的功能。我们使用的微处理器稳定运行在 100MHz，拥有 512KB 的程序 Flash 空间，非常适合处理数据量大，速度要求快的摄像组智能车[2]。

与第六届的比赛不同，本届车模改为单一电机，且配备的机械式差速器。因此单片机只需要一个脉冲累加器进行速度检测。本车采用采集精度高、运行稳定、结构简单轻便、占用空间小的霍尔传感器组成速度检测模块，实现对小车速度的实时准确监控。

在电源模块设计中，由于 CCD 传感器对电压要求较高，要达到 12V，LM2940 和 LT1085 已经不能够满足要求，因此采用 LT1946 稳压芯片将电池电压稳压到 12V 来对 CCD 传感器进行供电。

CCD 传感器数据处理电路主要部分为 TVP5147M1 和模数转换。由于 CCD 传感器输出的是复合信号，因此需要用 TVP5147M1 进行视频信号的分离，主要为行信号与场信号的分离以及奇偶场的识别等，模数转换的作用是将 CCD 传感器输出的模拟信号转换成数字信号。

控制策略是本文论述的重点，经过大量的试验后，我们决定采用以偏差为主，斜率为辅的策略计算转向角度，经过长时间的调试，这种转向策略转向灵敏，能够达到预期的效果。

本文将主要从智能车机械结构设计，硬件电路设计和控制策略等方面全方位展示我们制作的智能车，各章节主要内容安排如下：

第一章， 引言部分。主要介绍相关背景常识和智能车的主体结构布局。

第二章， 机械结构设计部分。主要介绍智能车前轮角度的调整，CCD 传感器和舵机的安装以及对轮胎的处理。

第三章， 硬件电路部分。主要介绍智能车硬件方案设计与各项电气特性，以模块为单元，逐一介绍。

第四章， 图像采集部分。主要介绍智能车摄像头采集图像的方式与特点。

第五章， 控制策略部分。主要介绍舵机和电机的控制策略，以及为了提高速度而进行的路径优化，速度控制策略。

第六章， 调试相关部分。主要介绍调试过程中所使用的上位机、蓝牙模块等。

第七章， 结论部分。主要介绍智能车制作的心得体会和仍旧存在的不足之处。

第二章　智能车机械结构安装与设计

本文在第七届“飞思卡尔”杯全国大学生智能车竞赛组委会提供的摄像头组车模基础上进行智能车的组装和设计。其中前轮倾角的调节关系到智能车高速行驶的稳定性，经过长时间调节，最终确定出利于智能车行驶与转向的前轮倾角。

2.1 前轮倾角的调节

智能车前轮倾角主要分为三种，即前束角、上开倾角和后倾角。而综合考虑各类角度对智能车行进的影响，我们只选择应用前轮后倾角。

2.1.1　后倾角

后倾角，就是指从侧面看，路面与转向销所形成的夹角。当转向销中心线垂直于路面时，后倾角最小。转向销上侧越往后倾，后倾角的角度越大，如图 1.1 所示。

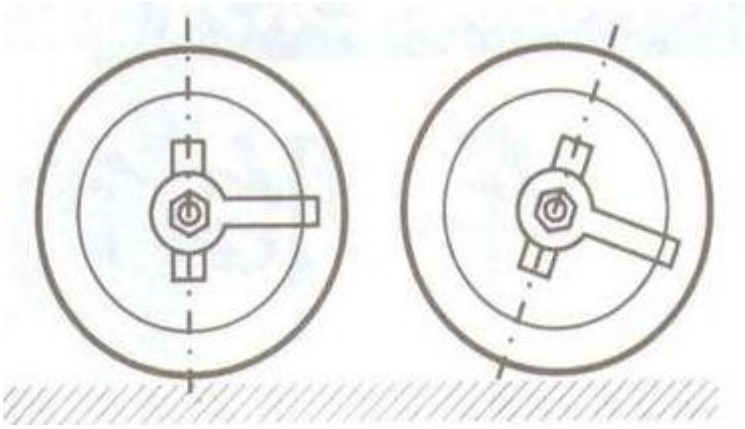

图 1.1　后倾角示意图

当后倾角的角度设定较大时，直线稳定性会变好。但是在弯道转弯时会产生前轮接地面积变小的情况，相反，将后倾角设定小的话，转弯的反应就会很灵敏。因此，转弯过度时，应将后倾角调大，转弯不足时，应将后倾角调小，这样就可以获得稳定的转弯特性。由于本智能车系统对转弯灵敏度要求较高，经反复试验后，最终将后倾角设置为最小。

2.2 CCD 传感器的安装

本智能车系统是依据摄像头采集的图像为依据行驶的，CCD 传感器采集图像的稳定性及图像包含的信息是智能控制策略的基础，而传感器的安装位置及角度则决定了采集的信息量的大小。

2.2.1 摄像头镜头的选择

不同型号的镜头的视角大小是不一样的，长焦镜头的焦距较长，成像较大，畸变较小，易于分析处理，当小车在直道或者小弯道上行驶时，可以很好的获得赛道信息。但是由于本届比赛黑色边线在赛道的两侧，当跑道转弯角度较大时，由于视角小，会出现看不到赛道而丢失黑线的情况，增大了道路信息判断的难度。而广角镜头则与长焦镜头相反，由于视角较大，广角镜头的成像较小，在弯道上可以采集到较多的赛道信息，但是在直道或者小弯道时，远处的赛道信息在图像中所占比例大大减少，分析赛道信息较为困难，即前瞻较短，同时广角镜头有明显的桶形畸变，会对图像的分析处理带来一定的困难。在进行多次尝试后，我们最终确定了一个既能适当采集弯路信息，又能符合前瞻要求的广角镜头。

2.2.2 CCD 传感器的固定

对于在赛道信息非常复杂情况下行驶的智能车，其舵机的打角以及行驶速度每时每刻都在发生变化，车体重量的减小，对速度控制非常有利，车体姿态的调整也越快，整体性能也就越好。因而，我们在固定 CCD 摄像头时选用了质量很轻而刚度又能很好满足要求的圆形碳纤维杆。当摄像头位置处于智能车前面时车体近处视角过窄，转弯时采集赛道信息较为困难，但是由于车模本身的中心偏后，如果，摄像头后置又会加大前后重量差距，因此综合考虑后，本文将 CCD 传感器安装在智能车的中心位置，这样既符合采集要求，又减小了前后重量差距。

2.2.3 CCD 传感器的调节

固定好 CCD 传感器后，还需要对 CCD 传感器的架设高度和俯仰角度进行调节。我们主要考虑到了视野的广度，即前瞻、图像畸变和外界光线对摄像头的影响等三个方面。

摄像头位置越高，采集图像畸变越小，远处道路信息所占比例越大，但是过高会导致智能车行驶不稳定，影响舵机打角的连续性，严重时会导致翻车。角度可以调节摄像头的前瞻距离和智能车近处道路的采集盲区的大小，角度越小，前瞻距离越大，同时近处采集盲区越大。另外，如果摄像头偏于“仰视”，会更容易接受到外界的光线，对图像的采集造成不利的影响（比如广为人知的照相机的闪光灯对摄像头组就会有很大影

响），因此我们适当降低了仰角。经过长时间试验，设计出可以调节 CCD 传感器角度和高度的摄像头固定装置，如图 1.2 所示，其高度距离智能车底盘约为 25 cm，镜头面与碳纤维杆的夹角约为 45°，摄像头支架距车最前端 10 cm。同时，考虑到赛道表面比较光滑，为了更好的滤掉外界光线直射入镜头对图像采集造成不利的影响，我们创新的应用了偏振片，如图 1.3 所示。经过试验，在装了偏振片之后，采集到的图像有很好的对比度，且不再有大片的白色“光斑”，这对区分赛道上的白色和黑色信息非常有帮助。

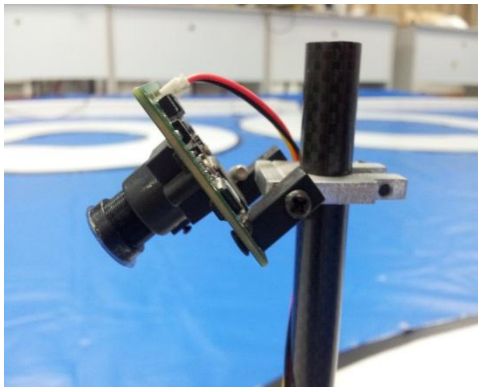

图 1.2　CCD 传感器固定装置

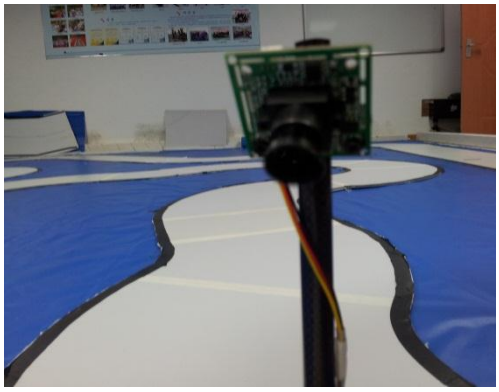

图 1.3　安装偏振片后的 CCD 传感器

2.3　舵机的安装设计

本方案舵机采用直立安装形式，舵机扭矩输出齿轮连接一个圆形舵盘，舵盘通过以块板与两侧车轮连杆连接。舵机的安装方式决定了伺服器控制前轮转弯的连杆的长度，连杆长度越长，前轮转弯越迅速，但同时转矩会越小；反之，连杆长度越短，前轮转弯越缓慢，但同时转矩会越大。根据智能车转弯响应时间及力矩大小要求，本智能车系统对伺服器的安装设计如图 1.4 所示。

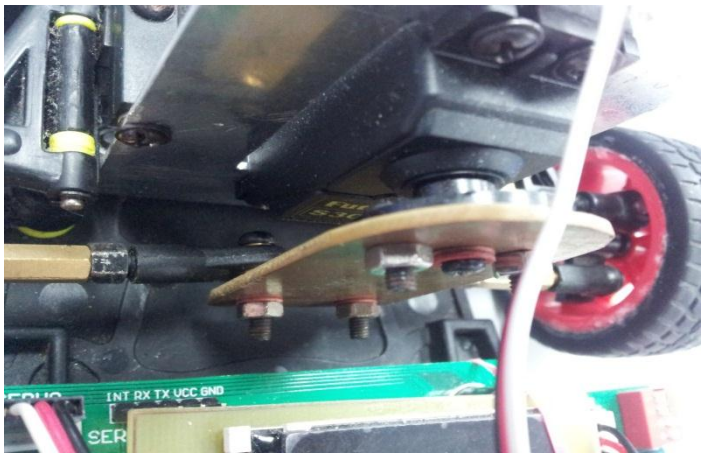

图 1.4　伺服器安装设计

2.4 智能车前后轴重量分配与重心调整

在满足各机械部件正常工作的前提下，应该考虑调整智能车前后轴重量分配，其对智能车车性能影响主要为[3]：

1、动力性能方面

由汽车理论知识可知，汽车行驶时，存在坡度阻力、滚动阻力、空气阻力和加速阻力，汽车要想正常行驶，其驱动力必须大于或等于这四种阻力之和，而汽车的驱动力受限于路面的附着条件。前后轴的重量分配将会影响后轴即驱动轴承担的重量，从而影响附着力的大小。因此，智能车重心位置必须保证驱动轮能够提供足够的附着力，仅从此方面考虑，重心越靠近驱动轴越好。

2、对转向性能的影响

重心前移，易发生后轴侧滑，而且增大舵机的转向力矩；重心后移，前轮摩擦力减小，转向性能有所下降，在转弯时，有可能会滑出赛道。

3、对稳定性能的影响

重心太高，汽车高速急转弯行驶时，会发生侧向倾覆，应尽量降低重心。同时，摄像头位置过低则会影响前瞻和图像畸变程度。另外，调整重心也是为了在通过坡道时尽量降重心，增强稳定性。对此，我们增加前轮垫片，使整体车身降低。

总的来说，智能车的重心调整在几何中心附近较为合适，尽量满足驱动、转向、稳定性、前瞻等的要求，并根据智能车实际行驶要求和情况进行适当调整。

2.5 智能车轮胎的处理

众所周知，汽车轮胎对汽车行驶起着至关重要的作用。一个与地面接触面积较大，转弯过程中胎侧变形较小的轮胎会给汽车转弯提供充足的侧向力。A 车车模的新轮胎由于有中间分模面的缘故，使得轮胎并没有完全接触赛道，因此，在使用前，我们用指甲刀小心的剪去了这一凸起的分模面，提高了轮胎与赛道的接触面积；另外，在调试过程中，当车速超过 2.5 m/s 时，小车在转弯过程中，前轮轮胎特别容易与轮毂松动甚至脱离，导致小车冲出赛道，而后轮的轮胎并没有明显的松动或脱离。因此，我们考虑将前轮轮胎用胶水粘在轮毂上，以改善弯道性能。而在实际调试过程中，还应该特别注意赛道的维护和保养，每当小车冲出赛道后，都应该用抹布小心擦拭轮胎。整个车模改装后如图 1.5 所示和图 1.6 所示。

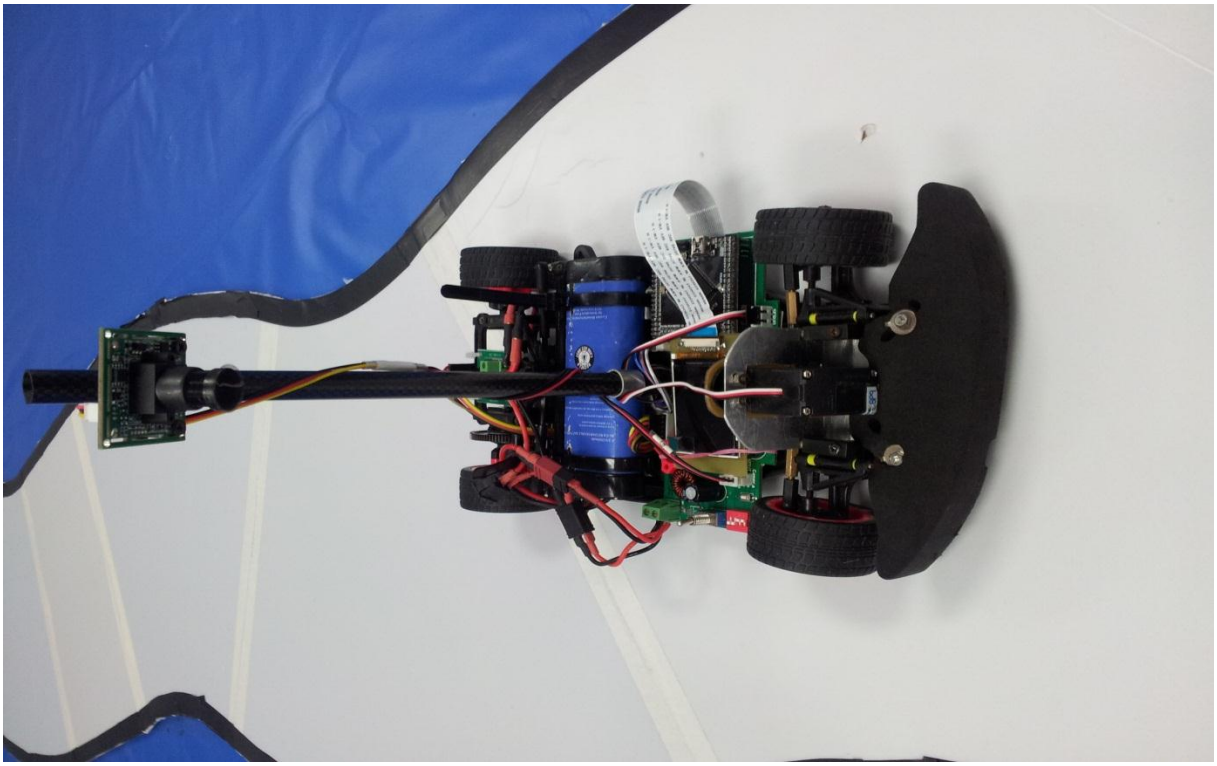

图 1.5　改装后车模图 1

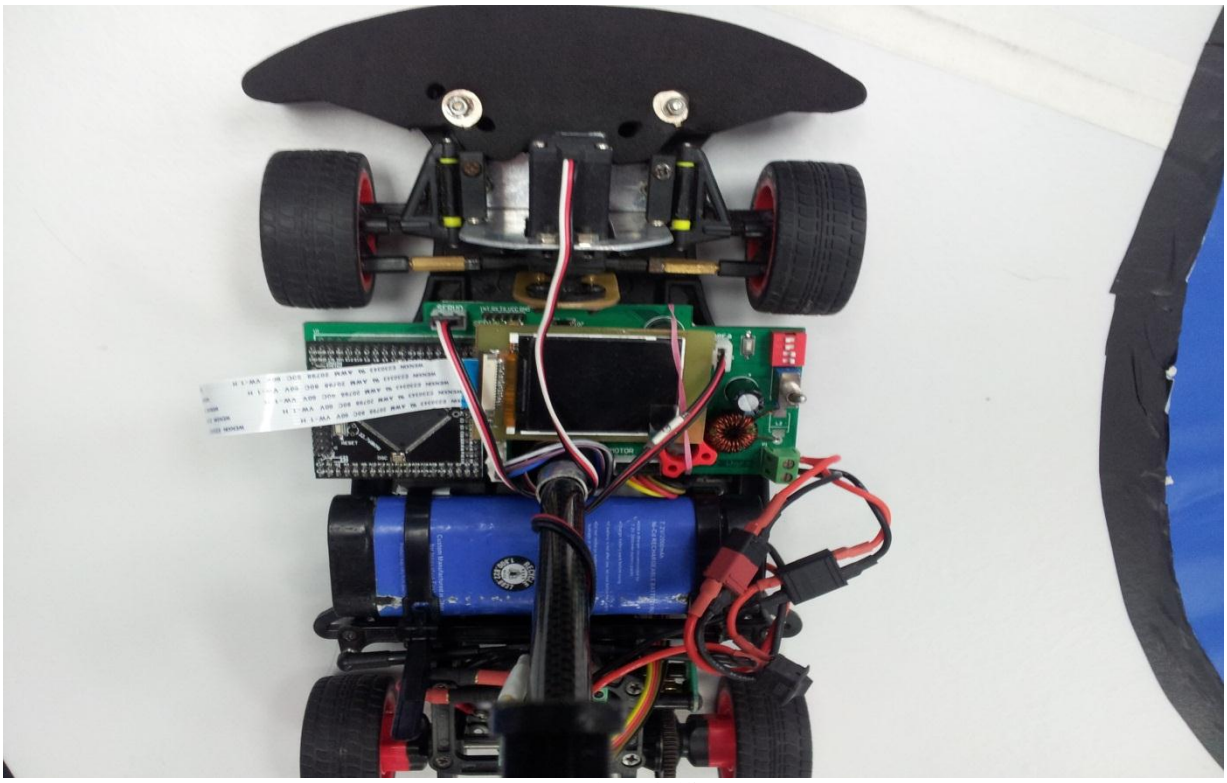

图 1.6　改装后车模图 2

第三章　智能车硬件电路方案设计

本系统的硬件电路采用模块化设计方式。主要包括单片机最小系统模块、电源模块、图像采集及处理模块、测速模块、电机驱动模块、无线通讯模块、显示模块等部分。其中，电源部分为整个系统稳定工作的基础，经过多次改进，最终确定了使用 3.3V 对单片机进行供电。此方案有效解决了系统各模块间电平匹配问题，同时也可以避免因为智能车加减速，造成电池电压瞬时降低，导致的单片机复位的问题。测速模块、电机驱动模块构成驱动电机闭环控制电路，实现电机转速的快速响应。在文献[4]和文献[5]的指导下，智能车的硬件电路模块均达到要求。智能车系统的硬件结构框图如图 3.1 所示。

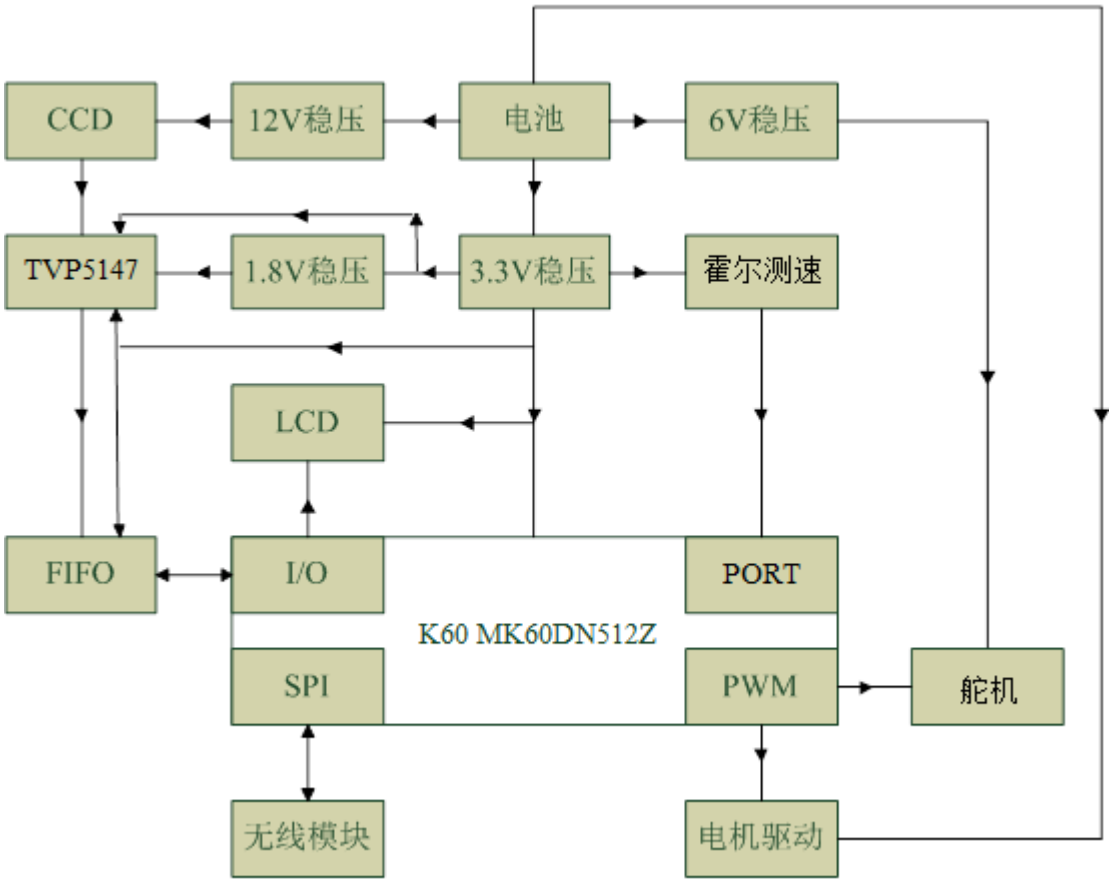

图 3.1　硬件结构框图

3.1　单片机系统板电路

本方案的核心控制器为飞思卡尔公司生产的 32 位单片机 K60 MK60DNZ512VLQ10，所使用的单片机采用 144 引脚封装。

K60 系列单片机具有丰富的系统资源和方便的外部电路接口，其中包括 32 位中央处理单元，它具有 ARM Cortex-M4 内核，采用哈佛结构，拥有独立的指令总线和数据总线，它还拥有多用途时钟发生器，可扩展内存空间，16 位模数转换器，12 位数模转换器，带 6 位 DAC 的高速模拟比较器，1×8 通道电机控制/通用/PWM 定时器，2×2 通道正交解码器/通用/PWM 定时器，丰富的中断模块，多功能定时器模块，DMA 技术，UART 支持 ISO7816 和 IrDA、I2S、I2C、CAN、SPI。GPIO 支持引脚中断 [3]。系统板电路模块如图 3.2 所示。

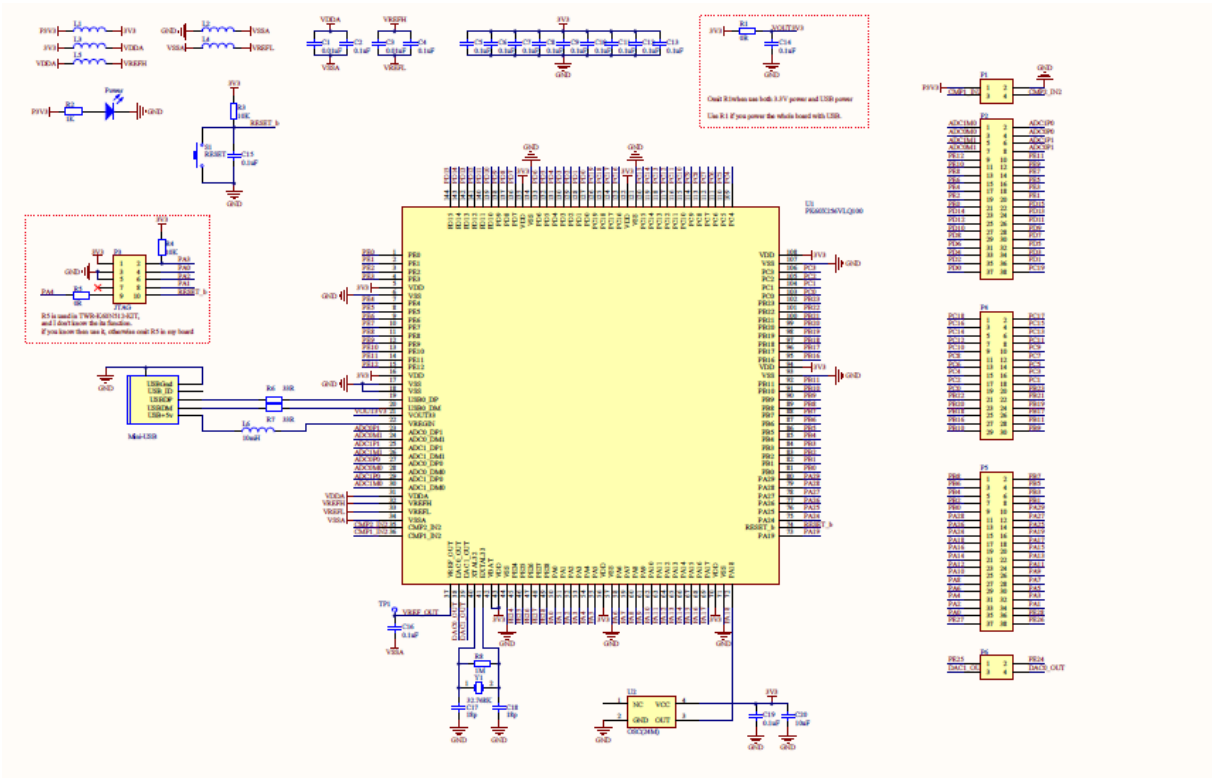

图 3.2　单片机系统板原理图

单片机系统版原理图包括晶振及 PLL 电路，复位电路，MCU 工作模式电路，JTAG 电路以及扩展出的一些普通 I/O 与 ADC 等。其中，晶振电路对于整个系统的稳定性是至关重要的，本设计中使用的是 24MHz 无源晶振，无源晶振外围电路简单方便，而且

价格也较便宜。通过锁相环和锁频环可以将 CPU 时钟倍频到 100MHz，系统可以在此高频下稳定工作，这使得单片机指令执行周期缩短，大大改善了智能车的实时控制性能。

3.2 电源模块

电源为整个系统的运行提供动力，电源模块是系统稳定工作的基础，因此，电源模块输出电压和电流的稳定性在整个智能车系统中起着非常重要的作用。

3.2.1 电池

智能车的电池为 Ni-Cd(镍镉)电池，额定电压 7.2V，但实际使用过程中，一般应将电池电压保持在 7.6V 以上，该电池充满电时电压一般在 8V 左右。该类型电池具有高效的利用率和稳定的性能，一直被作为各种航模、电动车等的供电设备。为了获得最高的性能和最长的寿命，该充电电池必须以正确的方法来使用。

对镍镉电池充电时，通常采用电池容量值的大约两倍来充电，当电池是 1800mAh, 那么用 3 到 4 安培来充电是安全的，而且充得比较饱满。通常电流高，电池的爆发力会强些，但未必如电流低时饱满。同时我们也必须注意充电电流不能过高（>7A），当电流过高时，不仅不能提高电池性能，反而会损坏电池，严重时会导致电池起火、爆炸。根据经验，一般在 4.0 到 6.5 安培之间是效果较好。

当电池压小于 7V 时，应注意及时充电，电池过放会对其造成不可逆转的损害，电压低于 6V 会对电池造成毁灭性伤害。

3.2.2 电源升压电路

由于 CCD 传感器的额定工作电压为 12V，因此需要将电池电压升至 12V。

LT1946 是 LINEAR 公司 生产的高性能稳压芯片，其使用方便，通过改变外围电路电阻的阻值即可以调整输出稳压值。经过测试，当输入达到 2.1v 时，输出便可以稳定在 12V 左右，达到系统要求。因此，本系统采用 LT1946 作为电源升压芯片，电路原理图如图 3.3 所示。

经过升压，成功将系统输入电压稳定在 12V，并给 CCD 传感器供电，对整个系统的稳定起到了至关重要的作用。

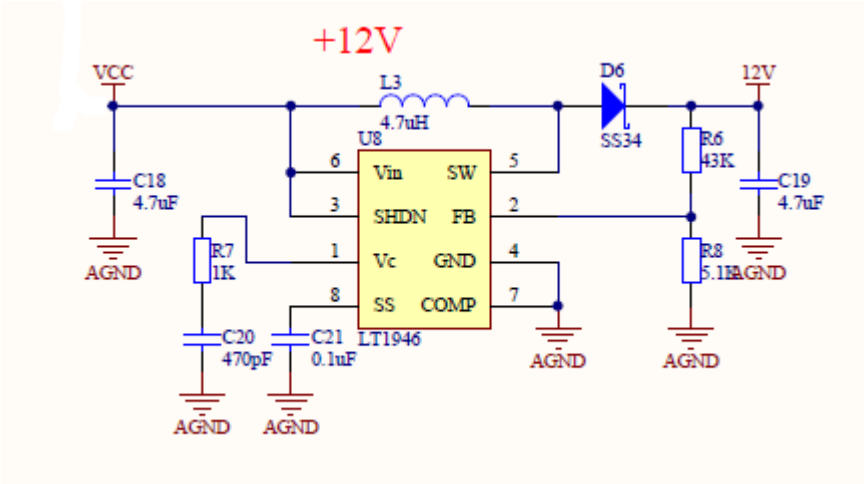

图 3.3　电源升压原理图

3.2.3　稳 3.3V 电路模块

由于单片机的额定电压为 3.3V，同时电机驱动信号端、霍尔测速模块、CCD 信息处理等重要功能模块的电压均为 3.3V，因此，将电压稳定在 3.3V 并给各模块供电是必不可少的。本方案首先将电池电压稳定到 4V，然后再稳定到 3.3V，具体电路的设计原理如图 3.4 和图 3.5 所示。

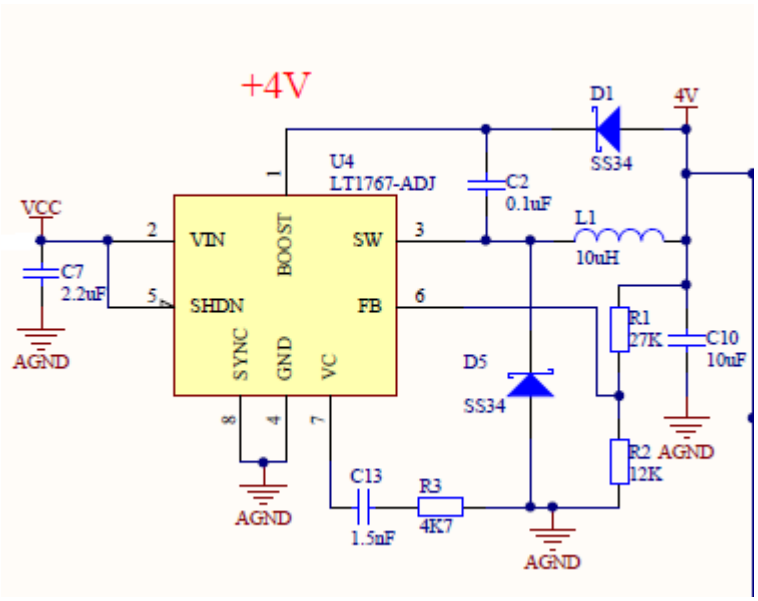

图 3.4　稳 4V 电路原理图

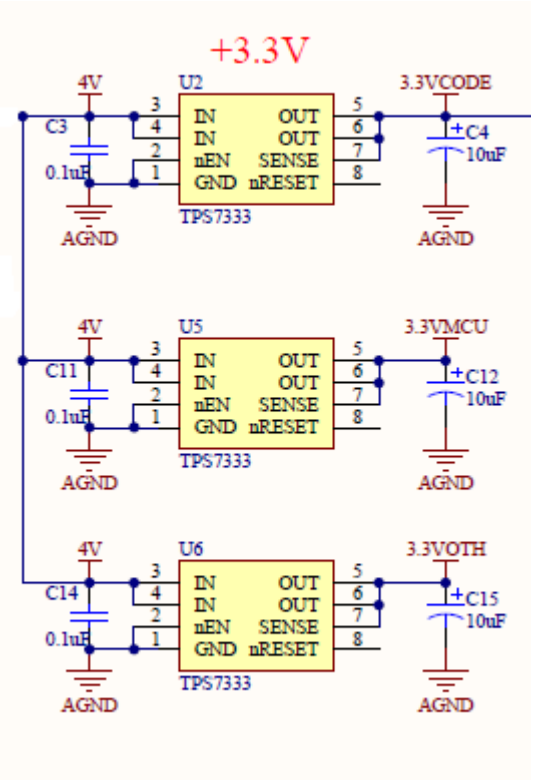

图 3.5　稳 3.3V 电路原理图

3.2.3　稳 6V 电路模块

由于舵机的额定工作电压为 4~6V，而智能车在实际行驶过程中，转弯需要的转矩较大，为了保证舵机的准确快速响应，本系统采用 6V 给舵机供电，电路原理如图 3.6 所示。

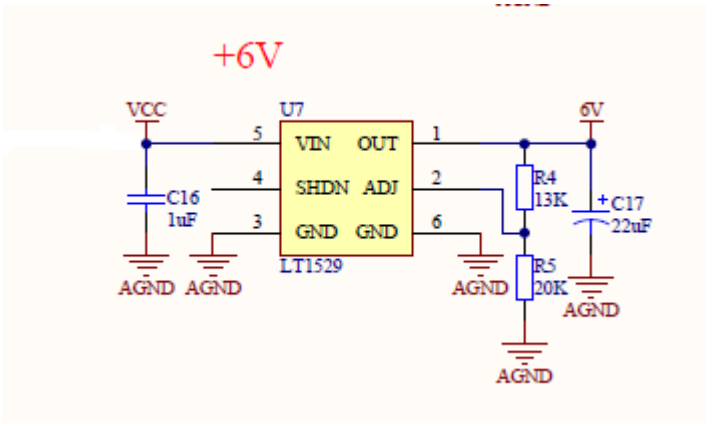

图 3.6　稳 6V 电路原理图

3.3　CCD 数据处理模块

目前 CCD 电荷耦合器件是主要的实用化固态图像传感器件，它具有读取噪声低，动态范围大，响应灵敏度高等优点[4]。

CCD 传感器输出的信号为视频混合信号即全电视信号，本系统采用 TVP5147M1 芯片对混合信号进行解码，分离出行同步信号、场同步信号等，并通过外置 FIFO 缓存数据。

3.3.1　CCD 传感器输出信号介绍

CCD 传感器直接输出的信号为彩色全电视信号，除与黑白全电视信号相同含有亮度、 复合同步、 复合消隐、 均衡等脉冲信号外，还含有彩色信息的色度信号和保证彩色稳定的色同步信号。因此，CCD 传感器输出信号具有如下特点：

1、参于混合的各种信号均保持着独立性，也就是说，可用各种方法将它们一一分离。 例如色度与亮度信号在时域重叠而在频域交错，色度与色同步在频域重叠而在时域交错，扫描用的同步与消隐信号在频域、时域均重叠，但在电平高低上有区别，它们与图像信号在时域交错，互不干扰。

2、它是视频单极性信号，既有直流成分，又含有交流成分，且是上下不对称的信号，占有 0~6MHz 的频带宽度。

3、对静止的图像而言，其电视信号以帧为周期重复，其场间、行间相关性也较大。对活动图像而言，则可说是帧间、 行间相关性较大的非周期信号，但其同步与消隐信号仍是周期性的。

传统的彩色全电视信号分离框图如图 3.7 所示。

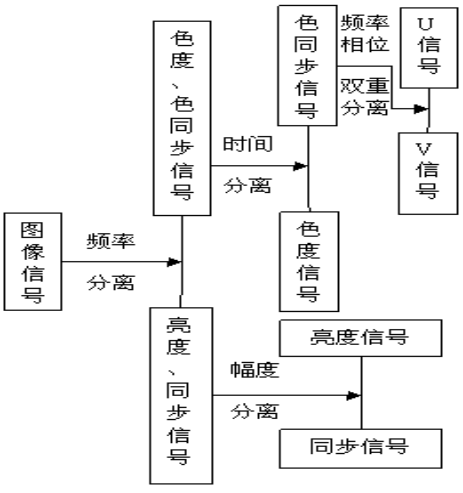

图 3.7　彩色全电视信号分离框图

3.3.2 CCD 信号分离电路

由于 CCD 传感器输出信号为混合信号，因此需要对其信号进行分离，将图像进行还原。本智能车系统只需要采集道路上的黑色引导线信息，因此，将信号分离后，只需利用灰度值即可确定跑道信息。

TVP5147M1 可以从 0.5～2V 的标准负极性 NTSC 制、PAL 制、SECAM 制视频信号中提取复合同步、场同步、奇偶场识别等信号，也能对非标准的视频信号进行同步分离，通过固定的时间延迟产生默认的输出作为场同步输出。电路原理图如图 3.8 所示。

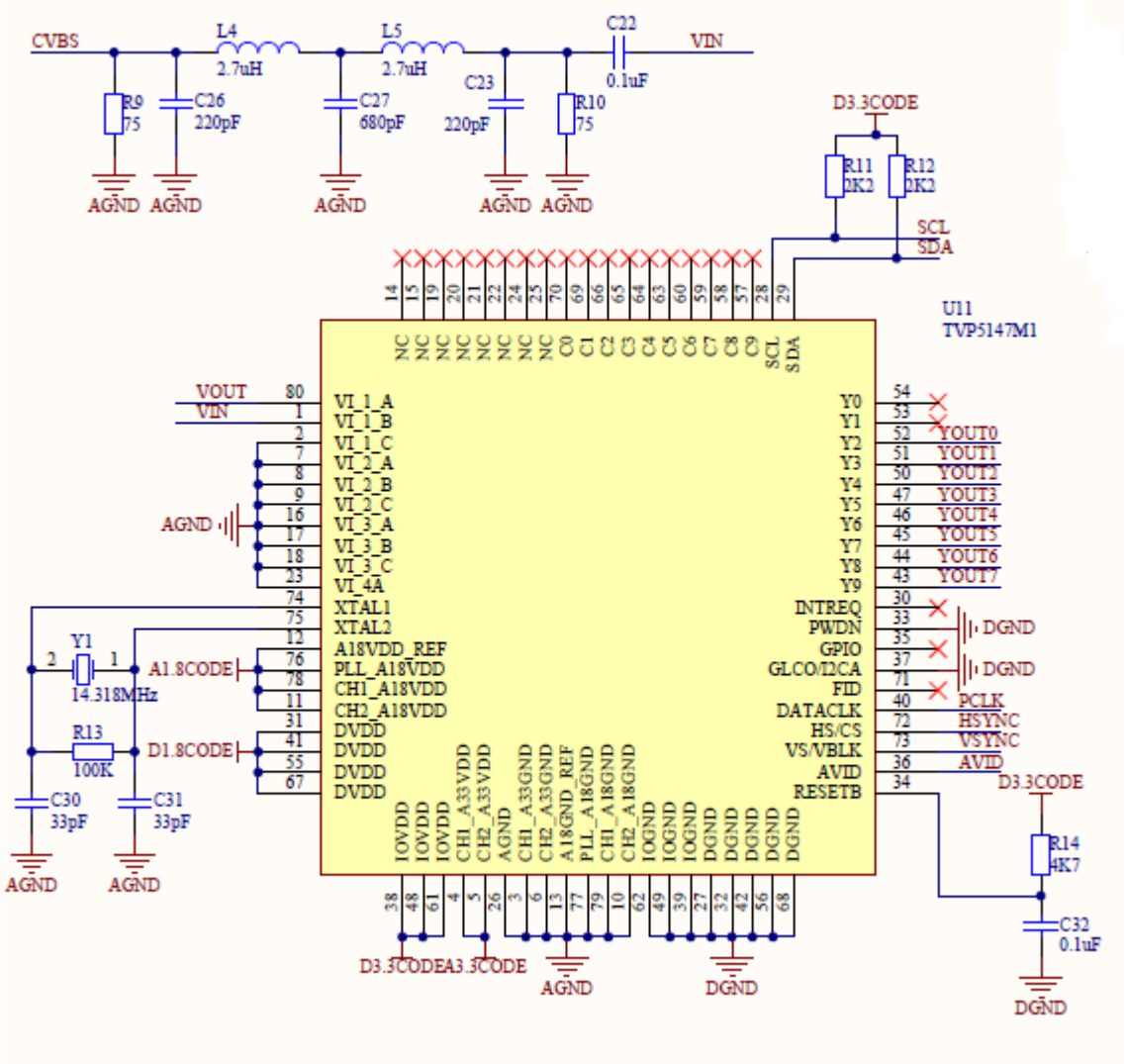

图 3.8　CCD 信号分离电路原理图

3.3.3　FIFO 电路

本设计采用 AL422B 对解码后的数字信号进行缓存，电路图如图 3.9 所示。

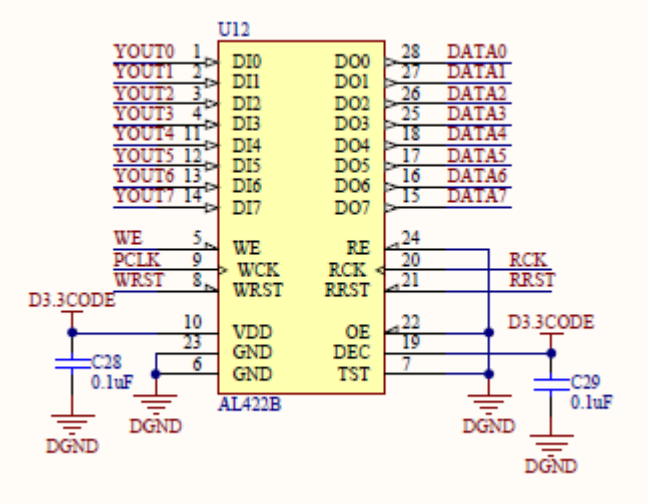

图 3.9　AL422B 电路原理图

3.4　电机驱动模块

3.4.1　H 桥电路模块

本智能车系统车模的电机型号较小，对电机驱动的输出电流的要求并不苛刻，因此本设计的驱动电路由 2 片 BTS7960 构成 H 桥。通过控制 4 个 MOS 管的导通和关断来实现正反转，并通过控制输入的 PWM 波的占空比来调节电机两端的平均电压，达到控制电机的转速的目的，具体电路图如图 3.10 所示。

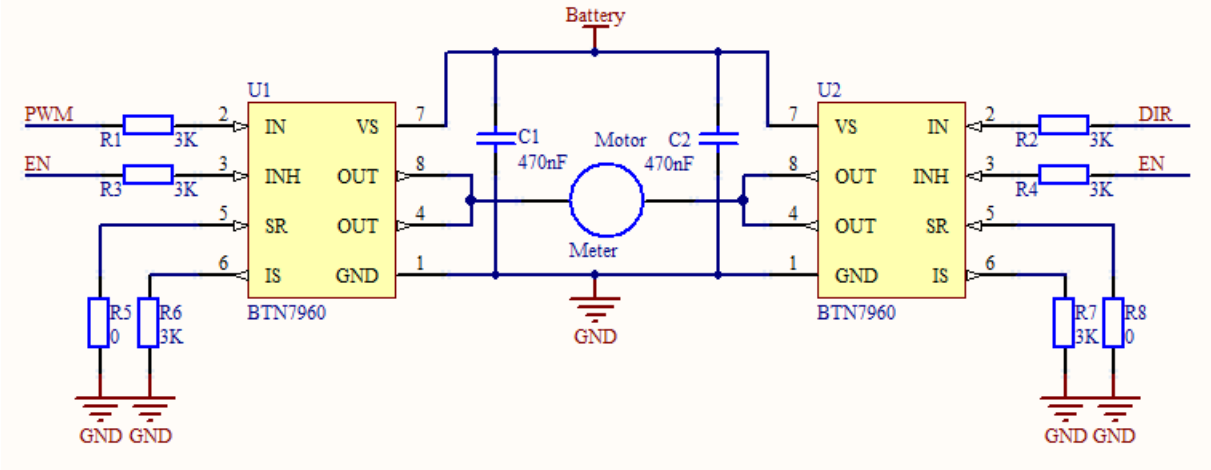

图 3.10　电机驱动原理图

3.5 串行通信模块

智能车调试的一个重要手段就是与电脑主机的通讯。为此，我们使用了串口通讯，实际调试中，我们将其与蓝牙模块连接，实现无线通讯，非常方便，极大的提高了调试效率。串行通信模块原理图如图 3.11 所示。

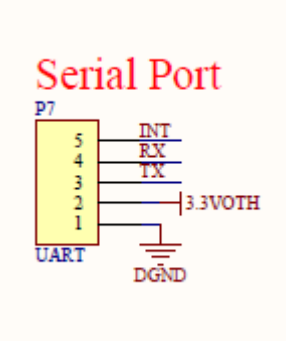

图 3.11 串行通讯模块

3.6 LCD 模块

对于摄像头组智能车来说，很重要的就是看图像，为了能够将摄像头的信息实时的显示出来，我们创新的应用了彩色 LCD 液晶显示屏。它不仅可以将图像实时显示出来，还可以将我们通过运算提取出的赛道信息实时显示，极大的提高了调试的效率，很多关于图像的问题一目了然。同时，它还可以显示非常丰富的信息，例如计算出的打角值，偏差值等等。而且它支持彩色显示，使得图像更为清晰直观。LCD 模块的原理图如图 3.12 所示，使用效果图如图 3.13 和图 3.14 所示。

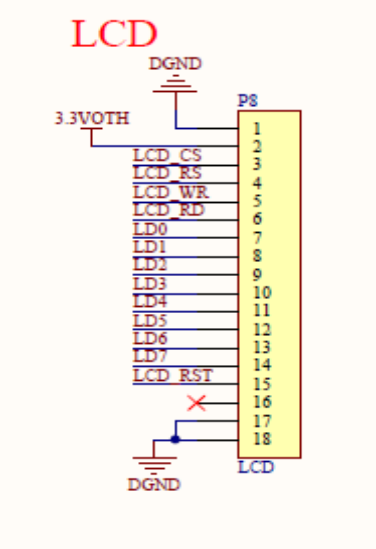

图 3.12 LCD 模块

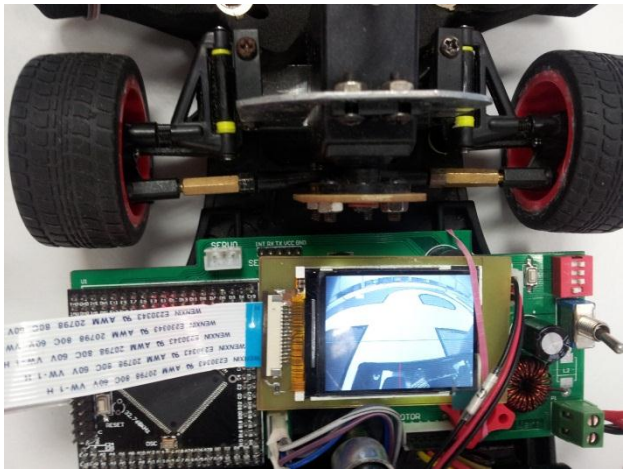

图 3.13 实时显示赛道信息

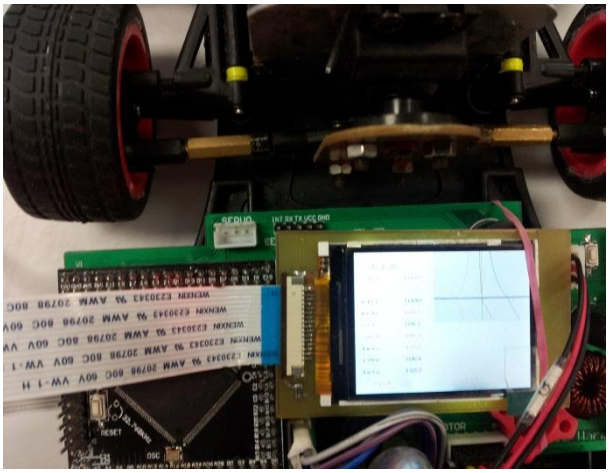

图 3.14 实时显示图像信息处理结果

第四章　智能车图像信息的采集与处理

对于摄像头组智能车，图像的采集与处理是至关重要的一步，直接关系到后面控制策略的效果，只有准确的识别出赛道，提取出路径信息，智能车才能以理想的路径与速度快速行驶。我们主要应用 C 语言编写程序，并适当使用了一些汇编语句。

4.1　图像采集

4.1.1　摄像头工作原理简介

摄像头的主要工作原理是[1]：摄像头以隔行扫描的方式采集图像，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对应的电压值，然后将此电压值通过视频信号端输出。具体而言（参见图 4.1），摄像头连续地扫描图像上一行，就输出一段连续的视频信号，该电压信号的高低起伏反应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个低于最低视频信号电压的电平（如 0.3V），并保持一段时间。这样相当于紧接着每行图像对应的电压信号之后会有一个电压“凹槽”，此“凹槽”叫做行同步脉冲，它是扫描换行的标志。然后扫描新的一行，如此下去，直到扫描完该场的信号，接着会出现一段场消隐信号。其中有若干个复合消隐脉冲（简称消隐脉冲），在这些消隐脉冲中，有一个消隐脉冲远宽于其他的消隐脉冲，该消隐脉冲又称为场同步脉冲，标志着新的一场的到来。摄像头每秒扫描 25 帧图像，每帧又分奇、偶两场，故每秒扫描 50 场图像。

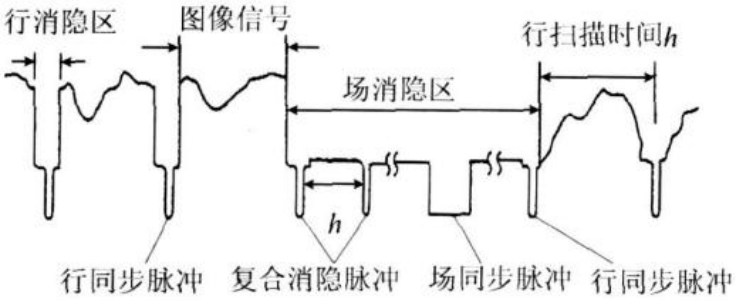

图 4.1　摄像头视频信号

CCD 采集的彩色全电视信号经过 TVP5147M1 芯片信号分离后，可以得到场同步信号、行同步信号和图像灰度信息，其中图像灰度信息通过模数转换后可以直接输入到单片机 I/O 接口，这样单片机就可以读取图像信息进行处理了。

4.1.2　单片机读取图像信息方法

本智能车系统采用的 CCD 传感器输出为 PAL 制信号，分奇偶场输出，由于奇场和偶场所呈现的灰度值图像的效果是相同的，因此本设计只读取其中一场信息即可得到路径信息。由上文知，摄像头每秒扫描 50 场图像，由此可知扫描周期为 20 ms。而舵机响应周期也为 20 ms，由此，我们希望将程序周期控制为 20 ms，这样，就需要对摄像头采集的行数进行处理。

由于采用了广角镜头，视野很广，无关信息也比较多，综合考虑，我们最终决定只选择采集每场中的 61 行数据。而令人兴奋的是，K60 系列芯片拥有 DMA（Direct Memory Access，直接内存存取）技术，经过一段时间的学习与调试，我们成功运用实现了 K60+DMA 的图像采集策略。

DMA 原理简介[6][7]：DMA（Direct Memory Access，直接内存存取） 是所有现代电脑的重要特色，它允许不同速度的硬件装置来沟通，而不需要依于 CPU 的大量中断负载。否则，CPU 需要从来源把每一片段的资料复制到暂存器，然后把它们再次写回到新的地方。在这个时间中，CPU 对于其他的工作来说就无法使用。DMA 传输将数据从一个地址空间复制到另外一个地址空间。CPU 初始化这个传输动作后，传输动作本身是由 DMA 控制器来实行和完成。典型的例子就是移动一个外部内存的区块到芯片内部更快的内存区（对应于本系统，是将数据从 FIFO 寄存器移到芯片内的寄存器）。像是这样的操作并没有让处理器工作拖延，反而可以同时去处理其他的工作。DMA 传输对于高效能嵌入式系统算法和网络是很重要的。在实现 DMA 传输时，是由 DMA 控制器直接掌管总线，因此，存在着一个总线控制权转移问题，即 DMA 传输前，CPU 要把总线控制权交给 DMA 控制器，而在结束 DMA 传输后，DMA 控制器应立即把总线控制权再交回给 CPU。

经过测试，单片机在 100MHz 的工作频率下，每场采集 61 行，每行仍可以稳定采集到 704 个有效点。同时，我们将算法不断改善，可以保证在 19 ms 内将采集到的图像进行处理识别，确定舵机和电机的控制策略，并最终将舵机的转角值输出，大幅提高了智能车的响应速度，我们也将程序周期控制在 20 ms.

接下来将重点介绍摄像头采集行的确定方法。

4.1.3　摄像头采集行确定

行同步信号的频率是低于单片机运行的频率的，因此理论上我们可以收到每一行的同步信号，进而将视野内摄像头看到的每一行图像数据都传回单片机，但是由于单片机的存储空间有限，无法也能没有必要存储整幅图像，因此采集行的确定应采用间隔读取方法。

经过测试发现，由于摄像头畸变，会发生如下现象：在离摄像头近处的区域，读取的数据所占比例很大，而远处区域所占比例很小。而实际在智能车行驶过程中，车近处的道路信息是比较稳定的，而变化大的是远处区域，如果仅仅按照原方法采集图像，远处区域只能用较少的点表示较大的变化，使得远处每个的代表跑道信息的点间隔都很大，造成远处信息无法利用的情况，这无形中使智能车的前瞻大打折扣，对于高速行驶的智能车来说是致命的缺点。这时我们想到，可以减少近处采集行的数量，增加远处采集行的数量，采用非等间隔采集的策略。而兼顾摄像头的纵向畸变的问题，我们想到，可以将摄像头采集的行数与真实世界中的距离对应起来，这样一来，摄像头的纵向畸变便被无形的缩小了很多。

综上所述，本设计系统创新性采用等距离采行的方法，即每隔 2 cm 采集一行数据，这样既可以保证远处区域信息的数量，又消除了摄像头纵向压缩畸变的影响。而之所以选择每隔 2 cm，是考虑到起跑线的宽度为 2.5 cm，2 cm 的采集宽度刚好可以保证不遗漏起跑线，便于起跑线的检测与处理。

等距离间隔采行法的具体实现方法如下：我们创新的用 A4 纸打印了尺寸为 2cm×2cm 的黑白相间的正方形色块，然后将它们拼接在一起，组成一个与长直道类似的图纸。将智能车的前轴与图纸边缘对齐，然后将采集到的图像用蓝牙模块发回电脑，用看图像上位机读取图像，并输出 CSV 格式的数据文件，然后再 Excel 里读出实际距离与采集行的对应关系，并记录下来。这种方法相比于利用卷尺或其他标记物效率和精度有很大的提高，起到了事半功倍的效果。试验用的 A4 图纸如图 4.2 所示，Excel 数据处理示意图如图 4.3 所示。等距离间隔采行法的程序流程图如图 4.4：

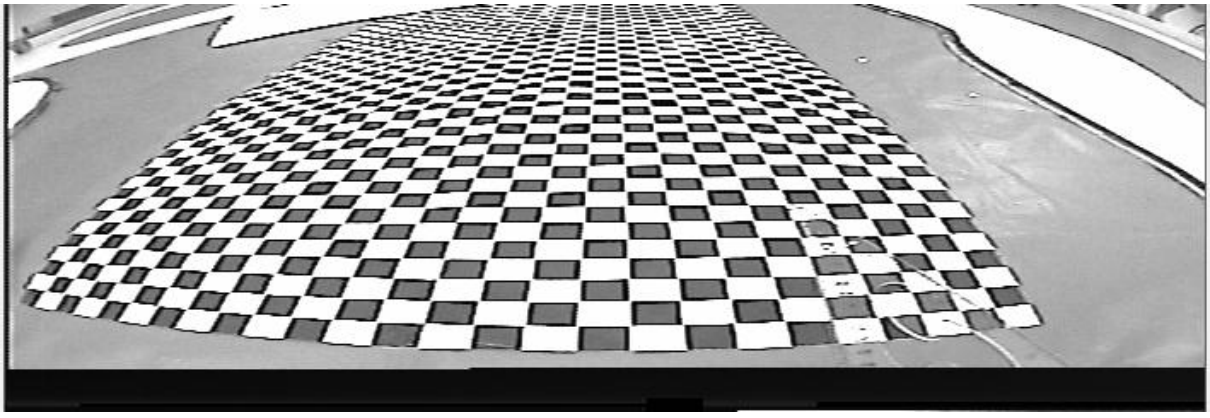

图 4.2　试验用图纸

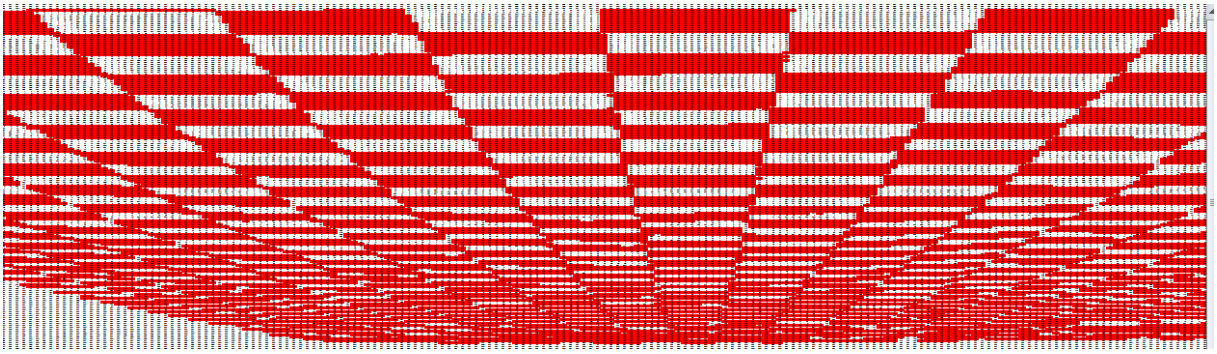

图 4.3 Excel 处理数据结果

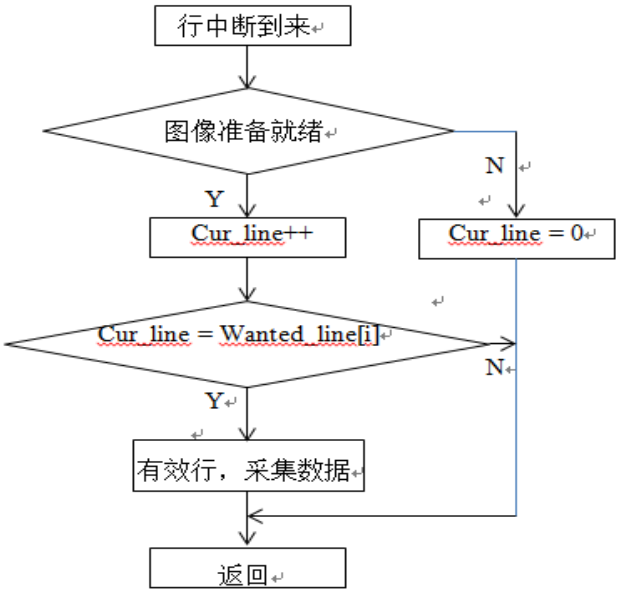

图 4.4　等间隔采行法流程图

图中 Cur_line 为行同步信号计数器，记录当前采集行数。Wanted_line[i]数组存放需要读取的行数。

最后，考虑到前方图像准确性很高，而且单片机的运行速度和存储空间相比 16 位的产片改善的很多，同时为了便于更加准确的确定起跑线和十字交叉线，我们决定在距离智能车前轴 10 cm 到 30 cm 之间采取每隔 1 cm 采集一行的策略，最终确定的采集行—距离表如表 4.1 所示。

表 4.1 采集行—距离表

采集行	距前轴距离/cm	采集行	距前轴距离/cm
2	10	149	52
9	11	152	54

16	12	155	56
22	13	157	58
28	14	159	60
34	15	162	62
40	16	164	64
45	17	166	66
51	18	168	68
55	19	170	70
60	20	171	72
65	21	173	74
70	22	174	76
74	23	176	78
78	24	177	80
82	25	178	82
86	26	180	84
90	27	181	86
95	28	182	88
97	29	183	90
101	30	184	92
107	32	185	94
112	34	186	96
117	36	187	98
123	38	188	100
128	40	189	102
132	42	190	104
135	44	191	106
139	46	192	108
143	48	193	110
146	50		

4.2 图像处理

图像处理是路径识别的基础，也将影响到控制策略能否很好的实现，因此，图像处理是至关重要的一部分[1]。

4.2.1 二值化算法

在数字化后的图像数组中，较暗的部分数值较小，亮的部分数值较大，而智能车跑道上的黑色引导线和白色的跑道对比是非常明显的，数值差也较大。根据这一点，可以将图像数据的每一点与适当的数即阈值比较，即设定一个阈值 threshold，对于视频信号

矩阵中的每一行，从左至右比较各像素值和阈值的大小，若像素值大于或等于阈值，则判定该像素对应的是白色赛道；反之，则判定对应的是黑色的目标引导线。

阈值分为静态阈值和动态阈值两种。静态阈值是指将阈值设定为固定值，在车行驶过程中，阈值不发生变化。这种方法的优点是计算简单，只需作比较即可，节省运算时间。缺点是环境适应性差，一旦换一种环境或者车行驶在光线变化较大的场地中，固定的阈值就可能不再合适，需要手动调整。动态阈值是指阈值可以随车的行驶环境进行动态调整，以适应环境的变化。这种方法的优点很明显，环境适应性很强。缺点是需要实时计算阈值，增加了计算量。由于本智能车摄像头加装了偏振片，同时，摄像头的对比度也已经调至最高，经过实际测试，发现采集到的图像对比度很高，黑白区别的很明显，受外界干扰并不大，因此，我们采取静态阈值，这也为控制算法节约了宝贵的时间。

本设计对图像数据每一行每一个像素点按照相应的阈值进行二值化，小于阈值时置1，表示其为黑色，否则置 0，表示为白色。

4.2.2　跟踪边缘检测法

识别出图像中的黑白信息后，就需要将赛道提取出来，即需要找到图像中黑白突变的边缘。由于黑色的目标引导线是连续曲线，所以相邻两行的黑白跳变点比较接近。跟踪边缘检测正是利用这一特性，对直接边缘检测进行了简化。其思路是：若已寻找到某行的边缘，则下一行就在上一行边缘附近进行搜寻。这种方法的特点是始终跟踪每行边缘的附近（左或者右），去寻找下一行的边缘。

在具体边缘呢检测过程中，为了尽可能消除噪点的影响，我们规定，在当前点为白，且其接下来两个点同为黑时，才认为找到了边沿，并将其位置坐标记录在数组 PathL[]与 PathR[]中，作为提取到的左右端的赛道。

4.3　路径提取

路径信息是整个智能车工作的软件基础，是转角控制和速度控制的主要依据；而如果能准确提取出路径特征，则可以采取特定的方案优化跑法，甚至实现“抄近路”。

4.3.1　路径提取总体设计思想

通过前面的图像处理方法可以知道，最终得到是左右两端的黑色引导线坐标的数组。由于引导线在赛道边缘，如果只跟随其中一条引导线，势必会增大冲出赛道的危险，降低稳定性，综合考虑，我们使用左右黑线的中线作为指引赛车前进的引导线，即使用 PathC[i] = (PathL[i] + PathR[i]) / 2，作为真正的“赛道”。

由于我们采用了“开窗口”的跟踪边缘检测法，要想准确提取赛道，必须要有一个可靠的起始行。在这个过程中，第一行从摄像头视野中心向两侧寻线，只有当左端黑线坐标与右端相差超过 200 个像素点时，才认为起始行可靠，否则，则继续从中心搜寻，直到找到可靠的起始行。一旦找到起始行，就开始跟踪边缘，基于上一行的边沿依次寻找下一行的边沿。

4.3.3 路径提取实现

有了准确可靠的基准行的位置，后面识别算法就简单可行了，从基准行不断地向后寻找，直至达到最后一个想要的采集行，或者达到一个不可信的行（例如，曲率特别大的弯道，会导致在实际世界中的一条黑线上找到两个边沿）。

为了更好的识别弯道，我们利用寻找首次跨越车模前方的黑线的办法，具体来讲，是寻找第一次跨越车模前方的黑线，如果该黑线是从左测来的，那么，车模就应该向右侧打角；反之，如果是从右侧过来的黑线跨越车模前方中心，车模应该向左打角。这种方法对弯道的识别非常准确，而最重要的是，可以通过跨越点距离车模的距离（这也体现了等间距采集的优点）来估算出弯道的曲率。如果这个距离很小，说明黑线就在眼前，必须以很大的角度转弯，如果距离很大，这应该以较小的角度转弯。实践证明，该方法对大弯，连续弯特别有效，而且，会自动将小 S 默认为直线。这种方法一个显著的缺点就是打角量跳跃现象很严重，转角并不连续，为了弥补这一缺点，我们还融入了斜率的打角方案，两种综合起来，对打角的控制有了较大的改善。

直线的判定可以根据摄像头能找到的最远的行数，如果达到了设定的最远行且中途没有发现弯道（因为广角镜视野很广，几乎不会发生丢线的现象，所以，出弯的时候，视野一下子会很广，容易误识别为直线），则可以认为来到了直线。

针对不同路径的过渡阶段，下文会提到我们使用的几个状态，在这里不再赘述。

总的来说，我们并没有很针对的根据曲率还判断是急弯还是缓弯，亦或是大 S 弯或小 S 弯，所以，我们的路径识别策略理论性不强，还是很粗糙的，我们更多的把注意力放在调试弯道打角时机与大小上。

第五章　智能车的控制策略研究

控制策略是智能车行驶的灵魂，它控制着智能车的转弯与加减速，还要识别出起跑线，同时应该可以应对一些突发状况，例如冲出赛道后的保护。本文将从路径优化策略，舵机控制策略，电机控制策略，起跑线识别策略和保护策略等几个方面介绍本智能车的控制策略。

5.1　路径优化策略

智能车比赛的赛道信息复杂多变，但主要仍是几种基本类型的赛道，大 S，小 S，直到和十字交叉线等等。通过观察成绩较好的车辆所走的路径，不难发现它们对弯道都进行了很好的处理，基本保证智能车在转弯时能尽量沿着内弯行驶。而本文的路径优化的目的就是使智能车能够在不冲出跑道的前提下以最快的速度按优化后的路线通过相应的区域。

5.1.1　路径判断策略

虽然弯道曲率各有不同，我们并没有采取严格的路径识别策略，即找到 180 度 U 型弯，大 S 等等，我们只识别三种形式的赛道：十字交叉线，长直道，弯道。这样做有两个好处，一是降低了程序的难度，抛弃了复杂的特征判断语句，将问题简单化；二是增加了智能车的适应性，判断语句越多，误判的概率便越大，将不同类型的弯道统一为弯道，只要将转角策略设置得当，是完全可以满足要求的。总的来说，我们将智能车行驶时每一刻的状态分为：

On_Straight：处于长直道上，应该降低转角幅度，并大幅度加速。

In_Straight：　处于连续弯过度中的直道以及出弯时刻，小幅增大转角幅度，小幅加速。

On_Corner：　处于弯道中，应该大幅加大转角幅度，按弯道速度行驶。

In_Corner：　处于入弯状态，适当减速，并适当增加转角幅度。

On_Cross：　处于十字交叉线，可以按直线加速，控制转角，不要走错路径。

5.1.2　路径优化的实现—弯道

由于本届赛事采用单电机驱动，不太可能实现主动差速，因此，我们的策略集中在转角时机与转角大小上。我们认为，制约智能车行驶速度的重要因素之一就是弯道的速度，如果智能车能以很高的速度，较为理想的路径过弯，将大大提高比赛成绩。一个理想的过弯过程应该是稍稍靠近外侧入弯，然后再沿内侧切弯而过。具体实现过程为，首

先利用视野前瞻判断出弯道，给舵机少量提前打角，进入弯道后，再大幅增大转角，以便切内线行驶。这种策略有一个很大的缺点，就是参数调试非常困难，时机和程度的掌握需要长时间的测试。

5.1.3 路径优化的实现—十字交叉线

由于引导线移到了赛道的两边，十字交叉线附近会出现全白的状况，这对图像的采集和处理是非常不利的，这使得十字交叉线成为本届赛事一个非常大的挑战。尤其是当十字交叉线入口和出口出以弯道衔接时，难度更大。对此，我们采取这样的策略：我们使用跟踪边缘检测法，即“开窗口”法，下一行会基于上一行找到的线从其两侧寻线，当只有左侧的黑线在“窗口”最左端（窗口的距离要设置得当，否则在弯道里容易发生错误，右侧同理）仍找不到满足条件的黑线时，则认为已找到了一个十字交叉线的起点，这是将此行的左侧黑线位置像左移动对多三个像素点；而如果两侧同时发生这种情况，则将此行左右两侧的黑线位置同时向内或向外移动三个像素点。经过反复调试，这样的算法可以将十字交叉线近似的补成连续地黑线，然后，再依据此打角。这样的策略效果很好，基本不会发生误识别的现象。同时，即便入或出十字交叉线时车身不正，仍可以很好的调整回来，而不会走错，达到了预期的效果。另外，由于十字线的判断还是会消耗一定的时间的，我们最终选择只在车模前方 60 cm 以内的行进行十字交叉线的判断。

5.2 舵机控制策略

5.2.1 舵机控制方法选择

本系统伺服器控制目标是舵机可以快速而连续的转动合适的角度。舵机控制常见的算法有三种：一是模糊自适应控制[11]；二是改进 PD 算法或模糊 PD 控制算法[12]；三是直接 PWM 控制法[13]。

模糊控制的优势是其调节响应速度快，但是却会导致舵机出现打角不连续且参数不直观，调试困难。PD 算法打角比较连续，但是响应速度较慢，当车速度较快时打角迟缓现象会很明显。直接 PWM 控制是指根据检测的不同路径，判断出小车所在位置，按不同的区间给出不同的舵机 PWM 控制信号。这种方法的优点很明显，由于舵机内部电机为闭环控制，因此可以准确快速的响应 PWM 占空比的变化，因此这种控制是最迅速、最准确的。但同时由于完全依赖 PWM 占空比，如果输入占空比不连续，就会导致打角不连续的情况，因此这种方法对于输入占空比的连续性要求较高。虽然闭环控制 PD 算法在实现所需的角度值方面优于开环控制算法，但是 PD 参数选择不当会很容易导致角度过冲，这使得智能车剧烈摇晃并且转向齿轮转动不顺畅[14]。

一方面为了打角连续，提高智能车运行的稳定性；一方面，还要保证舵机能以最快的速度按照要求打角，我们创新的将 PD 控制和直接 PWM 控制两者结合起来。

5.2.2　转向角度的计算

进行路径提取后，理想的路径应该是一条连续的直线，一个最简单的想法就是让智能车按照路径行驶，根据产生的偏差进行打角。角度的计算有如下几种方法。

1、偏差法。根据实际值与目标值计算出一个偏差，然后乘以一个系数，再将转角方向考虑进去，最终形成打角输出给舵机。

2、斜率法。即计算根据优化路径计算路径的斜率和截距，根据斜率和截距确定车转向角度。

3、模式识别法。赛道曲线大致分为直线、90 度曲线、大 S 曲线、小 S 曲线和环路。根据识别出的赛道曲线类型来确定转向角度方案。

综合考虑打角的实际效果，我们最终采用了以偏差法为主，斜率法为辅的计算转角策略。

同时，为了考虑不同类型赛道的特点，我们将 PD 参数分段进行控制，这样做事因为小车处于弯道和直道的转向模型不同，若采用统一的比例系数设置，那么该系数过大会导致小车振荡，过小会导致最大控制量偏小，小车转向不足，过弯时容易冲出赛道。使用分段比例控制便可以方便的解决上述问题。

5.3　驱动电机控制策略

本系统驱动电机控制采用闭环 PID 控制，为了避免误差累加，采取增量 PID 算法。

5.3.1　PID 算法简介

PID 控制即比例、积分、微分控制，该方法在工程实际中应用相当广泛。PID 控制算法具有结构简单、工作可靠、便于调整、性能稳定等优点。当被控对象的性能、结构等一系列参数无法得知，而且建模等控制方法亦无从下手时，我们便可以应用 PID 控制技术，但是，系统控制器的结构和参数必须依靠经验和现场调试来确定。当我们不完全了解一个系统和被控对象，或不能通过有效的测量手段来获得系统参数时，最适合用 PID 控制技术。PID 控制，实际中也有 PI 和 PD 控制。PID 控制器就是根据系统的误差，利用比例、积分、微分计算出控制量进行控制的。图 5.1 为 PID 控制系统原理图。

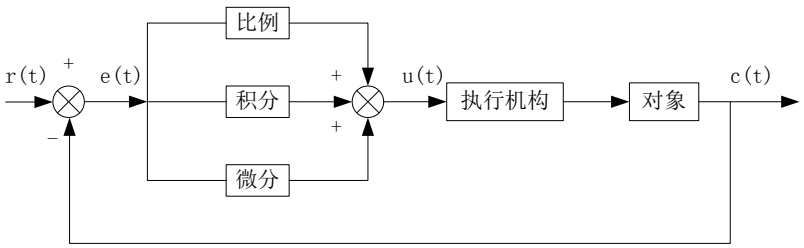

图 5.1　PID 控制系统原理图

PID 函数的一般形式为：U(s)=kp(1+1/(TI*s)+TD*s)；其中 kp 为比例系数，TI 为积分时间，TD 为微分时间常数。

比例控制是一种简单实用的控制方式。该方法的输出与输入误差成比例关系。比例系数越大，调节作用亦越大，减少误差的过程则越迅速，但是比例系数如果过大，则系统的稳定性会大大下降。

对于积分控制，控制器的输出与系统的输入误差信号的积分成正比关系。积分项的引入是为了消除稳态误差，积分项对误差取决于时间的积分，随着时间的增加，积分项会增大。这样，即便误差很小，积分项也会随着时间的增加而加大，它推动控制器的输出增大使稳态误差进一步减小，直到等于零

微分控制中，控制方法的输出与输入误差的变化率成正比关系。智能车控制系统由于存在有较大惯性，其变化总是落后于误差的变化自动控制系统，在克服误差的调节过程中可能会出现振荡，严重时会导致不稳定。为了充分解决此问题，我们需要提前抑制误差的变化作用，即在误差接近零时，抑制误差的作用就应该是零。这就是说，在控制器中仅引入“比例”项往往是不够的，比例项的作用仅是放大误差的幅值，而目前需要增加的是“微分项”，它能预测误差变化的趋势，这样，具有比例+微分的控制器，就能够提前使抑制误差的控制作用等于零，甚至为负值，从而避免了被控量的严重超调。

5.3.2　PID 参数整定

PID 控制算法中 PID 参数的整定是控制系统设计的核心内容。该过程是根据被控过程的特性确定 PID 控制器的比例系数、积分时间和微分时间的大小。

PID 控制器参数整定的方法很多，概括起来有两大类：一是理论计算整定。该方法需要依靠一定的数学模型，即通过一定的模型进行理论计算，最后确定控制器参数。但是，理论计算的方法所得到的数据常常是不能够直接运用到实际中的，其必须通过实际工程进行验证，并进行调整和进一步修改。二是工程经验整定。该方法主要依赖工程经验，直接在控制系统的试验中进行，且方法简单、易于掌握，在工程实际中被广泛采用。

工程整定方法中的 PID 参数整定主要有临界比例法、衰减法、反应曲线法。三种方法各有其特点，每一种方法所得到的控制器参数，都需要在实际运行中进行最后调整与完善。此外，其共同点都是通过试验，然后按照工程经验公式对控制器参数进行整定。

进行 PID 控制器参数的整定步骤一般如下：

1、预选择一个足够短的采样周期让系统工作；

2、仅加入比例控制环节，直到系统对输入的阶跃响应出现临界振荡，记下这时的比例放大系数和临界振荡周期；

3、在一定的控制度下通过公式计算得到 PID 控制器的参数；

4、根据实际运行情况对计算出的 PID 控制器的参数进行调整。

本系统的 PID 参数为通过上位机调试得来，具体调试方法为：先将 PID 参数设置为经典参数，然后通过上位机观察速度曲线，不断改变 PID 参数，直至观察速度曲线发现其加减速时间很短，超调量很少，则说明此时的 PID 参数已经基本比较合适，这样就确定出适合本系统的一组 PID 参数。

5.3.3　增量型 PID

数字 PID 控制算法包括位置式和增量式 PID，为了避免历史数据中误差的累加，本设计采用增量式 PID。

增量型算法具有很多优点：增量型算法每次计算的是输出控制量的增量，而控制量是需要加上上次的输出量，这样产生误动作的几率大大降低；增量型算法不需要做累加，增量的确定仅与最近几次偏差采样值有关，计算精度对控制量的计算影响较小，而位置型算法要用到过去偏差的累加值，容易产生大的累加误差；采用增量型算法，易于实现手动到自动的无冲击切换。

控制编程依据：

$$\begin{aligned} u(n) &= u(n-1) + Kp*[e(n) - e(n-1)] + Ki \\ &\quad + Kd*[e(n-2) - 2e(n-1) + e(n)] \end{aligned} \tag{4.24}$$

其中：$u(n)$为第 n 次输出控制量；$u(n-1)$为第 n-1 次输出控制量；$e(n)$为第 n 次偏差；$e(n-1)$为第 n-1 次偏差；$e(n-2)$为第 n-2 次偏差；Kp 为比例增益系数；Ki 为积分增益系数；Kd 为微分增益系数。

电机 PID 控制子程序流程如图 5.2 所示。

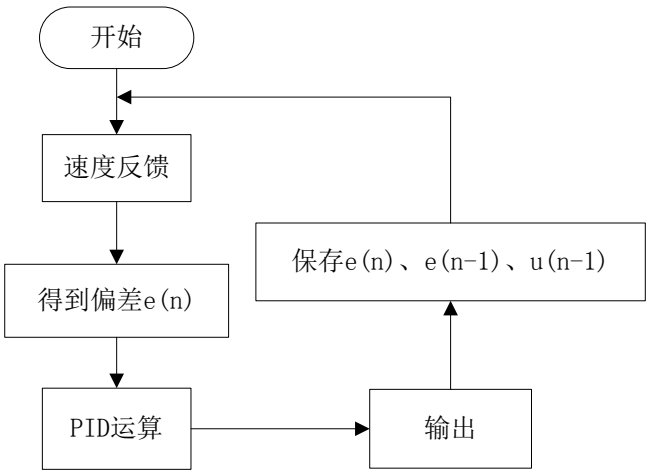

图 5.2　PID 控制流程图

为了实现电机的连续闭环控制，单片机需要实时对测速传感器输出的信号进行采样，通过对电机最大转速、霍尔传感器分辨率、单片机指令执行速度等参数进行分析，结合以往电机控制中的经验，每次采样的周期定为 5 毫秒。

本系统增量型 PID 参数确定是根据 Z-N 条件先确定出一组 PID 参数，然后根据智能车实际行驶状况进行调整，最终找到一个较为理想的 PID 参数。

5.3.4　速度控制策略

速度控制策略主要控制的是加速和减速。主要思想为在入弯时迅速减速，然后过弯，在弯内和出弯时加速，以便安全快速通过弯道。

同时，为了考虑稳定性，我们设置一个用来控制直道加速的标志，straight_nr。在 On_straight 状态下，straight_nr 会自增 1，当达到 4 时，即连续 4 场图像都为直道时，速度直接增至 straight_max，否则则速度每次增加一点（speed_set += speed_step），直至加至 straight_max。这样做的目的是防止在连续弯衔接处出现直线而以满速行驶发生危险。而在其他类型的路径中，我们对速度的控制都增加了优化，不仅仅是单纯的急加速和急减速，实践表明，我们的速度控制策略较好的达到了我们预期的目的，效果理想。

5.4　起跑线识别策略

本届赛事的起跑线如图 5.3 所示。

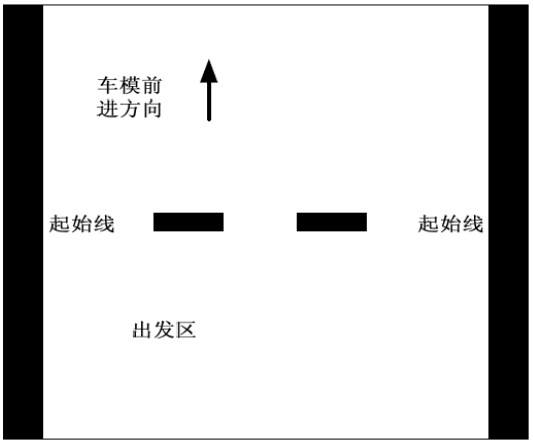

图 5.3 起跑线

一个较为理想的方案就是按照黑白的分布进行判断，即出现了黑、白、黑、白、黑、白、黑后，就判断为起跑线。而在实际测试中，这样的条件是在有些苛刻，当小车车身不正时，经常发生不能识别的现象。因此，我们将条件放宽，只要出现白、黑、白、黑、白即可。考虑到执行这样的判断也会花费时间，我们只对车身前方 30 cm 以内的行进行起跑线检测，事实表明，是可以准确检测起跑线并在规定的距离内及时停车的。

5.5　停车保护策略

在调试过程中，尤其是速度显著挺高之后，赛车冲出的赛道的可能性大大增加，为了避免出现危险，我们对赛车进行了两方面的保护策略。一方面是，一旦摄像头采集到的图像全部为黑，则认为已冲出赛道，立即停车；另一方面是，如果测速模块返回的速度连续 5 个周期都为 0，则认为赛车已撞到障碍物停车，此时也立即停车。当然，最后一个方面会严重干扰正常的停电机调试，因此，我们在主控板上增加了一个按钮，当检测到按钮被按下时，便取消第二方面的保护。

实际调试过程中，保护策略的效果非常明显，极大避免了赛车撞坏的可能。

第六章　开发环境与上位机系统设计

6.1　开发环境

由于采用了基于 ARM 架构的 32 位微处理器，开发环境与之前相比有很大变化。应用广泛的开发软件主要有 Code Warrior 10. 2 和 IAR Embedded Workbench，现就二者的特点简要介绍如下。

新版本的 Code Warrior 功能变得异常强大，界面也更加完善友好，更为重要的是保留了 Processor Expert 功能，这将大大节约编写底层驱动的时间。但是其缺点也很明显，首先是 PE 功能配置十分繁琐，相比于 XS128 时代下的 PE，配置一个 K60 的模块需要很多选项，有时还达不到预期的效果；其次是软件运行速度很慢，在更改设置生效过程中，需要等待较长时间才会有响应；还有就是编译和下载速度也很慢，影响调试的效率。

IAR Embedded Workbench 由 IAR System 推出，IAR System 是全球领先的嵌入式系统开发工具和服务供应商，其推出的编译器 IAR Embedded Workbench 是其最著名的产品，它支持众多知名半导体公司的微处理器。IAR 编译器最突出的优点是运行速度非常快，从系统配置到编译，再到下载，整个过程没有丝毫拖拉。而缺点同样也是显而易见的，首先是底层驱动需要自己动手写，这个过程比较耗时，我们参照了大量的资料才得以完成；其次是编程环境文本编辑功能十分薄弱，没有语句或关键词自动补全功能，用起来比较费事。不过，IAR 显然意识到了这个缺点，令人高兴的是，它支持外部编辑器，也就是说，人们可以用其他的编辑器编写代码，而这些代码会即时同步到 IAR 的工程文件中。综合考虑，我们最终选择了 IAR Embedded Workbench 作为我们的调试工具。调试过程中，我们使用非常著名的 Vim 编辑器编辑代码，用 IAR 生成目标文件并下载到单片机中，两者的组合大大提高了代码编写和调试赛车的效率。

IAR Embedded Workbench 开发环境如图 6.1 所示，代码编辑环境如图 6.2 所示。

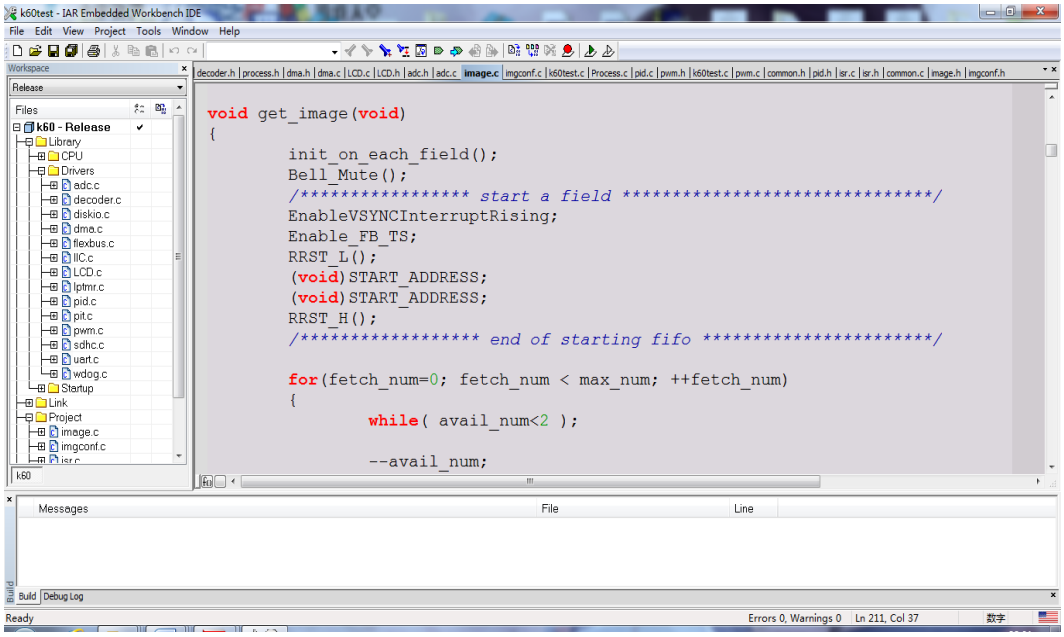

图 6.1　IAR 开发环境

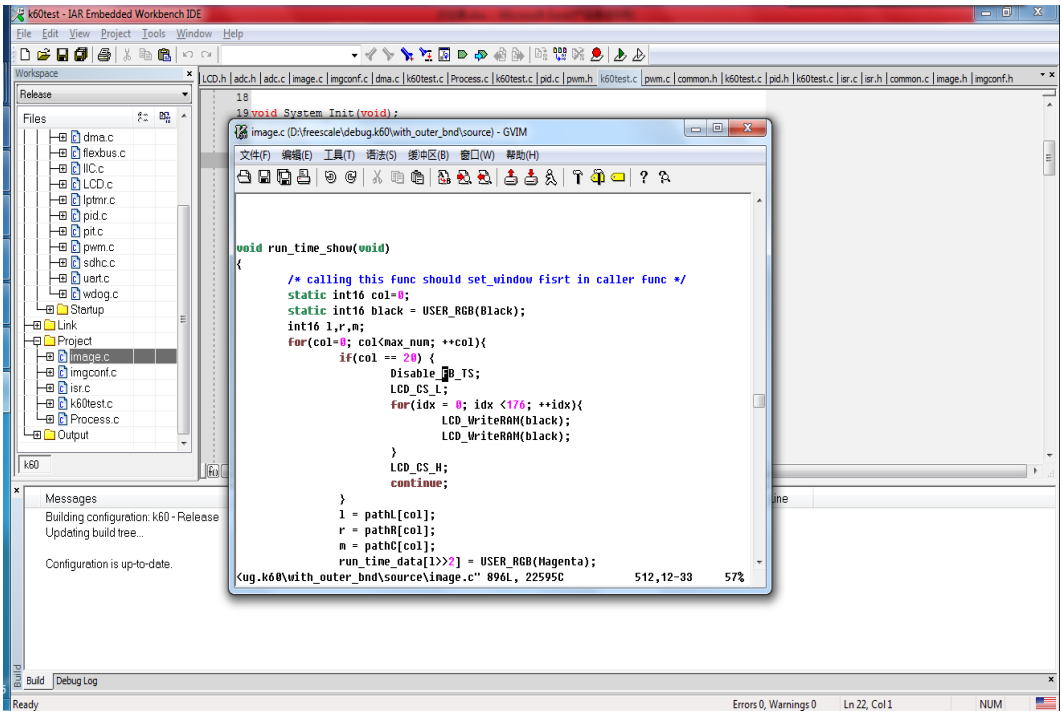

图 6.2　IAR+VIM 编辑代码

6.2 上位机系统设计

摄像头之于智能车如同眼睛之于人类，能否直观看到摄像头采集到的图像直接影响着控制策略的编写，因此，高效，强大的上位机就成为必须。一般的上位机系统应该主要包括以下几个基本功能：（1）串口通讯功能，该功能是整个上位机系统的基础，此环节是单片机与 PC 机通讯的保证；（2）存储数据并进行回放显示，即把采集的数据保存起来，以便于分析；（3）数据分析，把采集到的数据进行适当的处理分析，此处的数据分析需要进行适当的变成测试，并加入一些算法进行验证。上位机系统流程图如图 6.3 所示。

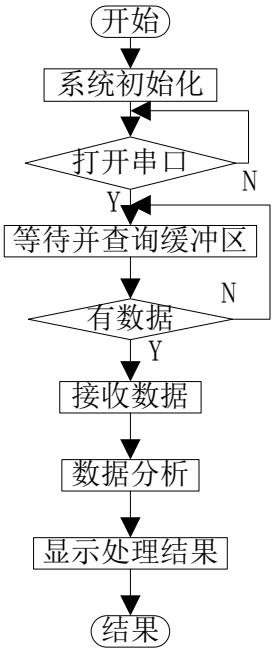

图 6.3 上位机系统流程图

我校智能车团队使用的上位机主要用以 C#语言编写，C#语法表现力强，简单易学，其语法与 C\C++非常相似，因此便于只有普通 C 语言基础的用户学习。与 C++相比，C#的语法简化了许多的复杂特性，如指针访问；而且还提供了很多比较强大的功能，如枚举、委托等。C#拥有像 VB 一样的快速开发能力；C#的效率接近 VC++，高于 VB；他完全面向对象，符合时代的需要；C#为托管型语言，开发效率更高，出错率更低，并且有很好的异常处理能力；现在 C#的资料越来越多，很方便学习。

我们使用的开发环境为 Visual Studio 2010。

6.2.1 使用到的上位机介绍

调试过程中，我们主要用到了两个上位机，现在它们分别介绍如下。

首先是看图像用的上位机。它可以以串口通讯的方式接收到智能车发回的图像，并能根据实际需要改变接受的行数和每行的像素点数，方便对不同采集模式的下的测试工作。它还能将采集到的图像绘图，并生成 CSV 格式的 Excel 文件，大大便利了数据的后期处理。软件的运行结果如图 6.4 所示。

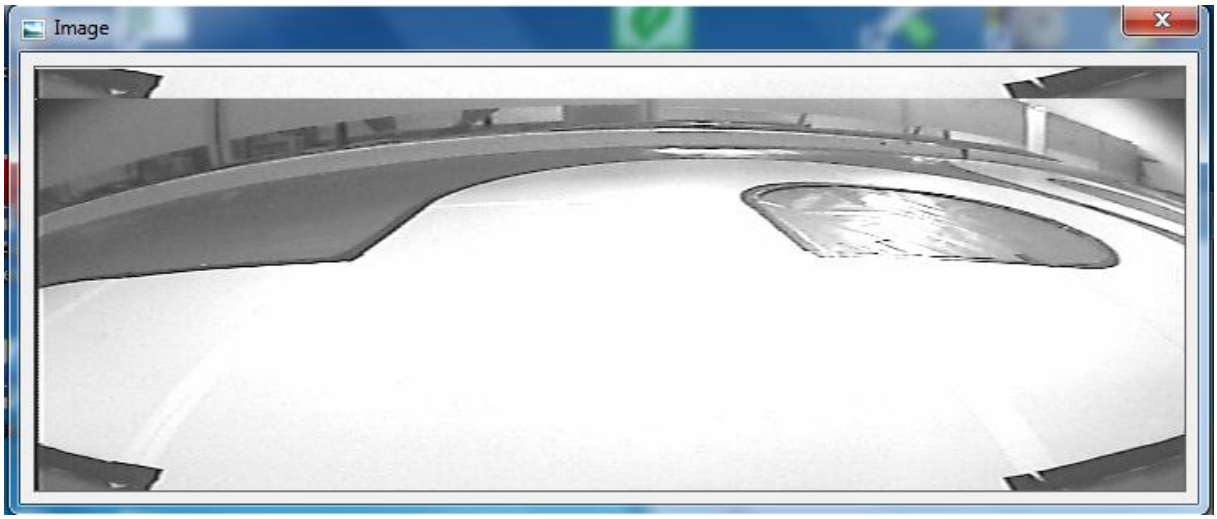

图 6.4 用上位机看摄像头采集的图像

其次是 PID 调节上位机，PID 调节使用非常广泛，是给支参赛队伍必备的速度控制策略，而其三个参数的整定则是用好 PID 至关重要的一步。PID 调节上位机可以绘制目标数据的曲线，通过与智能车无限连接，可以实时接收智能车传回的数据，并进行动态绘图，这大大提高了 PID 参数整定的效率。同时，它还具备无限发送功能，可以实现动态更改智能车运行参数，例如加减速等。该上位机运行效果如图 6.5 所示。

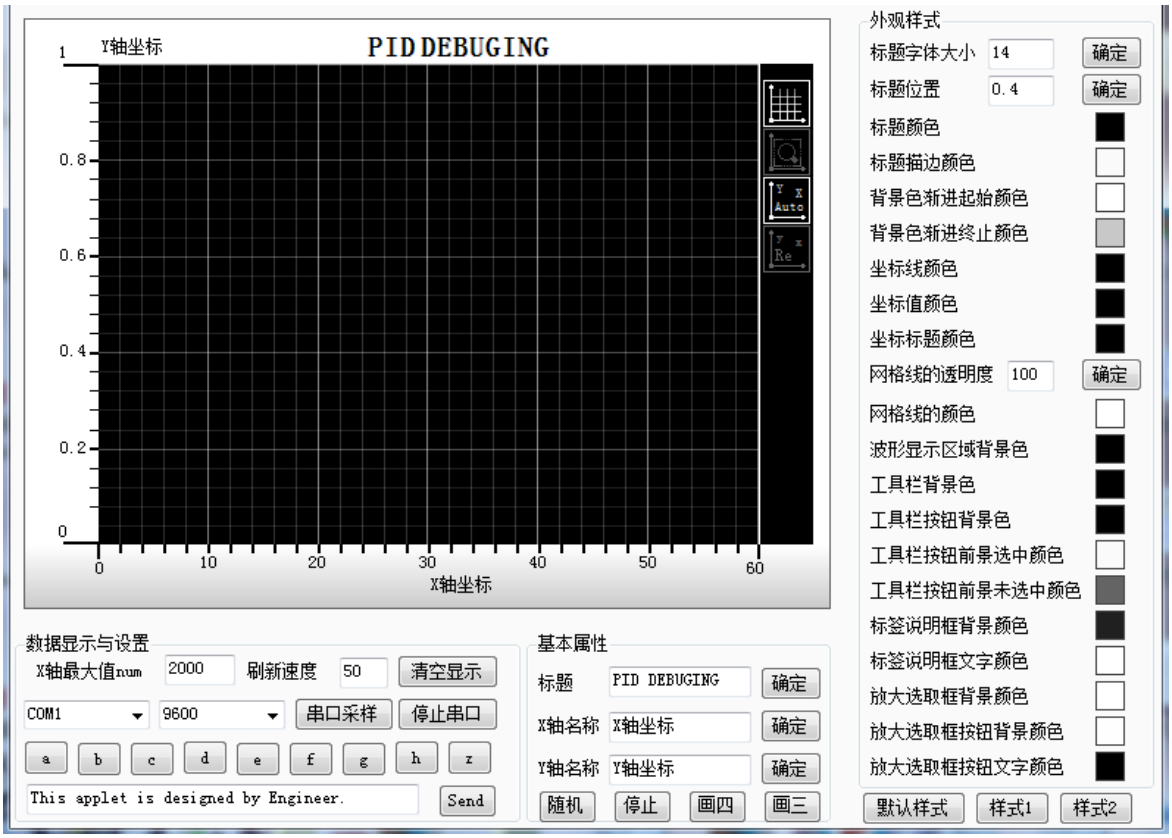

图 6.5 上位机运行效果图

当然，以上两个上位机还有其他一些比较人性化的功能，限于篇幅，不在赘述。我们团队也编写其他一些小插件，例如速度计算，PID 参数转换等等。可以说，在智能车调试过程中，上位机系统提供了准确的智能车运行参数，为调试及分析数据提供了重要依据，极大地加快了调试进程。

第七章 结 论

7.1 改装后模型车的机械电气参数

详见表 7.1

表 7.1

参数项目	参数
车体尺寸（长宽高）(mm)	300/170/280
车重(kg)	1.8
功耗(W)	<7（空载） 15（带载）
总电容(uF)	<1200
传感器(个)	CCD 传感器 1 个
	霍尔传感器 1 个
新增伺服电机(个)	0
赛道检测精度(mm)	10
赛道检测频率(Hz)	50

7.2 关于智能车制作

这是我们团队第一年参加这项赛事，从最初的一无所知，到后来一路过关斩将闯入全国总决赛，这期间我们倾注了大量的心血，投入了巨大的时间和精力。与往届的单线不同，这一届的双线对各参赛队提出了新的要求和挑战。综合前辈的经验和技巧，我们在控制策略上下了很大功夫，做了很大的改变，提出了很多新的思路与想法。经过数月的调试，我们的智能车运行稳定，基本达到了我们的预期目标。

但是，人无完人，限于时间紧迫等原因，我们还有很多可以改进的地方：

1、坡道识别。在调试过程中，我们发现我们的摄像头智能车在坡道处并不会丢失黑线，仍然会处理成直线行驶，因此，我们并没有添加关于坡道的策略。可是在实际比赛过程中，发现如果入坡道车身不处于赛道的中心位置，会造成上坡时车身抖动，虽然不至于冲出赛道，但这会损失速度。因此，可以考虑单独加设坡道识别装置，例如使用陀螺仪。当然也可以从软件角度，寻求坡道图像特征并加以识别。

2、起跑线的识别

起跑线识别方面我们没有增加任何辅助设备，单纯凭借图像特征进行识别，效果比较好，误识率很低。不过，我们也考虑增设激光传感器，更直观的识别起跑线，还能减少代码复杂度和运行时间。

3、上位机系统改进

我们的上位机系统比较离散化，调试时需要打开很多不同功能的软件，有时会造成串口的冲突，降低调试的效率和连贯性。如果时间允许的话，我们希望将各个上位机整合起来，形成一个调试系统，这样会使调试过程变得更加便利。

7.3 心得体会

“是机遇更是挑战，坚持就是胜利”，这是我们三个队员由衷的心声。我们认为“飞思卡尔”杯全国大学生智能汽车竞赛难度很高，挑战很大，含金量也非常足。这种类似体育赛事的竞速比赛也非常刺激，这也是它吸引众多厂商和大学生的原因。近一年的制作调试，我们遇到过很多问题，烧过芯片，撞坏过机械，和考试冲突……，面对这些，我们都坚持了下来，并乐此不疲。回想起来，近一年的实践确实大大提高了我们理论分析问题，动手解决问题的能力，这一年的辛勤付出定会让我们终身受益。最后，我们还是要感谢一直以来陪伴我们的带队教师，你们为我们提供了太多技术上和经验上的帮助；也还要感谢学校智能车团队的队员们，能结识你们这些志同道合的伙伴也是我们宝贵的财富；最后，还要感谢组委会老师们的辛勤工作，是你们的无私奉献为我们展示才华，挑战自我提供了一个广阔的平台。

参 考 文 献

[1] 吴怀宇,程磊,章政. 大学生智能汽车设计.北京:电子工业出版社,2008.

[2] 王宜怀,吴瑾,蒋银珍. 嵌入式系统原理与实践.北京:电子工业出版社,2012.

[3] 余志生. 汽车理论(第 5 版).北京:机械工业出版社,2009.

[4] 康华光，陈大钦，张林.电子技术基础.北京:高等教育出版社,2006.

[5] 阎石.数字电子技术基础.北京:高等教育出版社,1998.

[6] Freescale. K60 Sub-Family Reference Manual Rev.6, 2011

[7] 百度百科.http://baike.baidu.com/view/32471.htm

[8] YUAN Quan,ZHANG YunZhou,WU Hao,et al. Fuzzy Control Research In The Courses Of Smart Car[C]. Machine Vision and Human-Machine Interface (MVHI), Kaifeng,China, 2010: 764-767.

[9] 侯虹.采用模糊 PID 控制律的舵机系统设计.航空兵器，2006,2(1):7-9.

[10] 孙浩,程磊,黄卫华,等.基于 HCS12 的小车智能控制系统设计.单片机与嵌入式系统应用,2007,03(16):51-54.

[11] 谭浩强.C 程序设计.北京:清华大学出版社,2005.

[12] Freescale.K60 Sub-Family Data Sheet Rev.6,2011.

[13] Freescale.Kinetis Peripheral Module Quick Reference Rev.0,2010.

附　录

附录 A：程序源代码

*##**##*K60test.c:

```
#include "common.h"
#include "IIC.h"
#include "PWM.h"
#include "flexbus.h"
#include "dma.h"
#include "LCD.h"
#include "uart.h"
#include "lptmr.h"
#include "pit.h"
#include "pid.h"
#include "IMAQ.h"
#include "Process.h"
#include "diskio.h"
#include "decoder.h"
#include "adc.h"
#include "process.h"


void System_Init(void);


void main(void)
{
   speed_set = 100;
   DisableInterrupts;
   delay_ms(500);
   System_Init();
   delay_ms(500);
   EnableInterrupts;
   delay_ms(1000);
   //while(!global_flag);
   process();
}

void System_Init(void)
{
```

```
/*********************Init MCU*********************/
   /* PORTA_PCR4: Set as GPIO,Do not use NMI */
   PORTA_PCR4 = PORT_PCR_MUX(1);
   /* SIM_SOPT6: RSTFLTEN=0,RSTFLTSEL=0 */
   SIM_SOPT6 = (uint32_t)0x00UL;              /* Set reset pin filter */
   /* SIM_SCGC7: MPU=1 */
   SIM_SCGC7 |= (uint32_t)0x04UL;
          /* Initialization of the MPU module */
   /* MPU_CESR: SPERR=0,VLD=0 */
   MPU_CESR &= (uint32_t)~0xF8000001UL;
          /* Initialization of the PMC module */
   /* PMC_LVDSC1: LVDACK=1,LVDIE=0,LVDRE=1,LVDV=0 */
   PMC_LVDSC1 = (uint8_t)((PMC_LVDSC1 & (uint8_t)~(uint8_t)0x23U) | (uint8_t)0x50U);
   /* PMC_LVDSC2: LVWACK=1,LVWIE=0,LVWV=0 */
   PMC_LVDSC2 = (uint8_t)((PMC_LVDSC2 & (uint8_t)~(uint8_t)0x23U) | (uint8_t)0x40U);
   /* PMC_REGSC: TRAMPO=0,??=0,BGBE=0 */
   PMC_REGSC &= (uint8_t)~(uint8_t)0x13U;
   /* MC_PMPROT: ??=0,??=0,AVLP=0,ALLS=0,??=0,AVLLS3=0,AVLLS2=0,AVLLS1=0 */
   MC_PMPROT = (uint8_t)0x00U;                /* Setup Power mode protection register */
/*****************************************************/

/*****************Init GPIO for Bell**************/
   Bell_Port |= PORT_PCR_MUX(1)|PORT_PCR_DSE_MASK;
   GPIOE_PDDR |= Bell_Mask;         //output
/*****************************************************/

/******************Ring the bell*********************/
   Bell_Ring();          //set bit
/*****************************************************/

/******************Init Test Pin*******************/
   PORTE_PCR24 |= PORT_PCR_MUX(1)|PORT_PCR_DSE_MASK;
   GPIOE_PDDR |= 0x01000000;//Output
/*****************************************************/

/******************Init TVP5150**********************/
   TVP5147_Init();
/*****************************************************/

/*********************Init ADC*********************/
   Alarm_pit_init();
```

```
    ADC_init();
/*****************************************************/

/******************Init FlexBus**********************/
    flexbus_init();
/*****************************************************/

/******************Init DMA**************************/
    FIFO_DMA_Init();
    LCD_DMA_Init();
/*****************************************************/

/*************Init GPIO for FIFO ******************/
    RRST_Port |= PORT_PCR_MUX(1)|PORT_PCR_DSE_MASK;//RRST
    WRST_Port |= PORT_PCR_MUX(1)|PORT_PCR_DSE_MASK;//WRST
    WE_Port |= PORT_PCR_MUX(1)|PORT_PCR_DSE_MASK;//WE
    RRST_H();
    WRST_H();
    WE_L();
    //Output
    GPIOB_PDDR |= RRST_MASK;
    GPIOC_PDDR |= WRST_MASK;
    GPIOB_PDDR |= WE_MASK;
/*****************************************************/

/*******************Init FIFO***********************/
    RRST_L();
    WRST_L();
    //delay_systick(100*200);
    delay_ms(10);
    RRST_H();
    WRST_H();
/*****************************************************/

/******************Init LCD signal******************/
    PORTC_PCR0 |= PORT_PCR_MUX(1)|PORT_PCR_DSE_MASK;//RESET
    LCD_RESET_H;
    GPIOC_PDDR |= LCD_RESET_MASK;//Output
    PORTD_PCR15 |= PORT_PCR_MUX(1)|PORT_PCR_DSE_MASK;//CS
    LCD_CS_H;
    GPIOD_PDDR |= LCD_CS_MASK;//Output
```

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

/**********************Init LCD**********************/
   Disable_FB_TS; //Mask FIFO RCK when write to LCD
   LCD_RESET_L;
   delay_ms(10);
   LCD_RESET_H;
   LCD_Initializtion();
/*************************************************/

/**********************Init PWM**********************/
   init_Motor_PWM();
   init_Servo_PWM();
/*************************************************/

/**********************Init Motor**********************/
   I1_Port |= PORT_PCR_MUX(1)|PORT_PCR_DSE_MASK;
   I1_L();// disable motor
   GPIOB_PDDR |= I1_Mask;
   I2_Port |= PORT_PCR_MUX(1)|PORT_PCR_DSE_MASK;
   I2_L();
   GPIOB_PDDR |= I2_Mask;
/*************************************************/

/***************Interruput Init**********************/
   enable_irq(91);                    //Enable Port E Interrupt for VSYNC
   set_irq_priority(91,1);            //Set Port E Interrupt priority to 1
   PORTE_ISFR = 0x00000800;           //Clear PE11 interrupt flag
   PORTE_PCR11 = PORT_PCR_MUX(1);    //Set PE11 to input GPIO

   enable_irq(90);                    //Enable Port D Interrupt for HSYNC
   set_irq_priority(90,2);            //Set Port D Interrupt priority to 1
   PORTD_ISFR = 0x00000100;           //Clear PD8 interrupt flag
   PORTD_PCR8 = PORT_PCR_MUX(1);      //Set PD8 to input GPIO
/*************************************************/

/*************Init GPIO for switch*****************/
   PORTE_PCR10 |= PORT_PCR_MUX(1)|PORT_PCR_DSE_MASK;
   PORTE_PCR7   |= PORT_PCR_MUX(1)|PORT_PCR_DSE_MASK;
   PORTE_PCR8   |= PORT_PCR_MUX(1)|PORT_PCR_DSE_MASK;
   PORTE_PCR9   |= PORT_PCR_MUX(1)|PORT_PCR_DSE_MASK;
```

```
   PORTE_PCR6    |= PORT_PCR_MUX(1)|PORT_PCR_DSE_MASK|PORT_PCR_PS_MASK
                  |PORT_PCR_PE_MASK|PORT_PCR_IRQC(0xA);
/*****************************************************/

/***********Init GPIO for blueteeth INT************/
   PORTC_PCR5                                                                    |=
PORT_PCR_MUX(1)|PORT_PCR_DSE_MASK|PORT_PCR_PS_MASK|PORT_PCR_PE_MASK;
/*****************************************************/

/*********************Init UART*********************/
   uart_init (115200);
   //debug_info("Init UART 115200 baud : OK\n");
/*****************************************************/

/*****************Init SD Card*********************/
   sd_state = disk_initialize(0);
//    debug_info("Init SD card : ");
//    if(RES_OK == sd_state)
//        debug_info("OK\n");
//    else
//        debug_info("Failed\n");
/*****************************************************/

/*************Init Quadrature Decoder*************/
   Decoder_init();
/*****************************************************/

/*****************Init Cycle Timer******************/
   Cycle_timer_init();
/*****************************************************/

/*********************Init PID**********************/
   pid_init();//5
/*****************************************************/

/******************Init SpeedPIT********************/
   speed_set = 100;
   Speed_pit_init();
/*****************************************************/

/******************Init TVP5150*********************/
```

```
   TVP5147_Init();
/*****************************************************/

/******************Mute the bell*********************/
   Bell_Mute();
/*****************************************************/
}
```

*##**##*isr.c:

```
#include "isr.h"
#include "pid.h"
#include "pit.h"
#include "lptmr.h"
#include "imgconf.h"
#include "uart.h"
// use pit 0 as speed pit

void speed_isr(void)
{
      // clear pit0 flag
      PIT_TFLG(0) |= PIT_TFLG_TIF_MASK;
      pid();

}

void VSYNC_isr(void)
{
      // clear interrupt flag
      uint32_t flag;
      flag = PORTE_ISFR;
      PORTE_ISFR = flag;

      if(flag & 0x40){ // switch5 pushed.
            global_flag = TRUE;
            //bluetooth_on = TRUE;
            protection_on = FALSE;
      }
      if(flag & 0x800 == 0) // VSYNC not coming.
            return ;

      // FIFO write reset
      WRST_L();
      asm("nop");
      asm("nop");
      asm("nop");
      WRST_H();

      // enable HSYNC Interrupt
      EnableHSYNCInterrupt;
```

```
        // reset related parameters
        coming_num = 0;
        valid_num = 0;
        avail_num = 0;
        fifo_recv_done = FALSE;
        field_start = TRUE;

        // Reset cycle timer
        ResetCycleTime();
        ToggleTestPin;

}

void HSYNC_isr(void)
{
        // clear HSYNC flag
        uint32_t flag;
        flag = PORTD_ISFR;
        PORTD_ISFR = flag;

        // wanted lines are all recvd in fifo
        if(valid_num == max_num)
        {

                WE_L(); // write disable
                ++avail_num;
                ++avail_num;
                fifo_recv_done = TRUE;
                DisableHSYNCInterrupt;
                DisableVSYNCInterrupt;
                return ;
        }

        if(wanted_num[valid_num] == coming_num)
        {
                WE_H();
                ++valid_num;
                ++avail_num;
        }
        else
                WE_L();

        ++coming_num;
}


void ADC_TOF(void)
{
   (void)ADC1_RA;
   if((ADC1_SC2&ADC_SC2_ACFGT_MASK)==ADC_SC2_ACFGT_MASK)
   {
      ADC1_SC2 &= ~ADC_SC2_ACFGT_MASK;
```

```
        DisableAlarmPITInterrupt();
        Bell_Mute();
    }
    else
    {
        ADC1_SC2 |= ADC_SC2_ACFGT_MASK;
        EnableAlarmPITInterrupt();
    }
}
// ·äÃ ùÆ ÷± ¾ ¯
void Alarm(void)
{
    PIT_TFLG(1) |= PIT_TFLG_TIF_MASK;
    Bell_Toggle();
}
```

*##**##*process.c:

```
#include "process.h"
#include "image.h"
#include "common.h"
#include "adc.h"

static int16 turn =1160;



void process(void)
{
        I1_H();//enable motor


        while(1){
                run_time_show_aux();
                real_time_show_aux();
        }
}
```

*##**##*imgconf.c:

```
#include "imgconf.h"
const uint16 wanted_num_debug[] = {
   0,   1,   2,   3,   4,   5,   6,   7,   8,   9,
  10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
  20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
  30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
  40, 41, 42, 43, 44, 45, 46, 47, 48, 49,
  50, 51, 52, 53, 54, 55, 56, 57, 58, 59,
  60, 61, 62, 63, 64, 65, 66, 67, 68, 69,
  70, 71, 72, 73, 74, 75, 76, 77, 78, 79,
  80, 81, 82, 83, 84, 85, 86, 87, 88, 89,
  90, 91, 92, 93, 94, 95, 96, 97, 98, 99,
100,101,102,103,104,105,106,107,108,109,
110,111,112,113,114,115,116,117,118,119,
120,121,122,123,124,125,126,127,128,129,
```

```
130,131,132,133,134,135,136,137,138,139,
140,141,142,143,144,145,146,147,148,149,
150,151,152,153,154,155,156,157,158,159,
160,161,162,163,164,165,166,167,168,169,
170,171,172,173,174,175,176,177,178,179,
180,181,182,183,184,185,186,187,188,189,
190,191,192,193,194,195,196,197,198,199,
200,201,202,203,204,205,206,207,208,209,
210,211,212,213,214,215,216,217,218,219,
220,221,222,223,224,225,226,227,228,229,
230,231,232,233,234,235,236,237,238,239,
};

const uint16 wanted_num_run[] = {
   0,    8, 14, 21, 26, 33, 39, 45, 50, 55,
  60, 65, 70, 75, 79, 84, 88, 92, 95, 99,
102,106,109,112,115,118,121,123,126,128,
130,133,135,137,139,141,143,145,147,150,
153,156,159,162,164,166,168,171,173,175,
177,178,179,180,181,182,183,184,185,186,
187,188,189,190,191,192,193,194,195,196,
197,198,199,200,201,202,203,204,205,206,
207,208,209,210,211,212,213,214,215,216,
217,218,219,220,221,222,223,224,225,226,
227,228
};

const uint16 wanted_num_simple[] = {
   4, 12, 33, 45, 56, 67, 76, 86, 94,102, //10..2..28
108,123,138,146,154,164,172,181,188,192, //30..5..75
195,200,205,208,213,222,229,                    //80..5..100..10..120
};    // total 27

const uint16 wanted_num_spaced[] = {
   2,    9, 16, 22, 28, 34, 40, 45, 51, 55,  //10..1..19
  60, 65, 70, 74, 78, 82, 86, 90, 94, 97,   //20..1..29
101,107,112,117,123,128,132,135,139,143,  //30..2..48
146,149,152,155,157,159,162,164,166,168,  //50..2..68
170,171,173,174,176,177,178,180,181,182,  //70..2..88
183,184,185,186,187,188,189,190,191,192,  //90..2..108
193                                                    //110
};


const uint16 *wanted_num = wanted_num_run;
uint16 max_num = max_num_run;

// num of HSYNC interrupts, ie. lines sensor sends out
vuint16 coming_num = 0;
// indexed lines wanted
vuint16 valid_num = 0;
// avail num in fifo
```

```
vuint16 avail_num = 0;
// lines that is gona be fetched from fifo
uint16 fetch_num = 0;
// flag that all wanted lines are in fifo already
vuint8 fifo_recv_done = FALSE;
// field start
vuint8 field_start = TRUE;
```

*##**##*image.c:

```
#include "image.h"
#include "imgconf.h"
#include "dma.h"
#include "lcd.h"
#include "common.h"
#include "lptmr.h"
#include "pwm.h"
#include "pid.h"
#include "uart.h"

/********** road datas **********/
// used outside
static uint8 image[100][704];
const uint8 threshold = 200;
static int16 pathL[100]={0}, pathR[100]={0}, pathC[100]={352};
static uint16 run_time_data[176];  // needed init to White

/**********track data******************/
int16 track_cycle = 0;
int16 track_nr = 0;
int16 time_to_stop = 0;
int16 check_stop_cnt = 0;
int16 cycle_to_run = 2;
/***************************************/

static int16 line_spaced[] = {
  0, 0, 0, 0, 0, 0, 2, 4, 6, 8,       // 10..18
10,12,14,16,18,20,21,22,23,24,
25,26,27,28,29,30,31,32,33,34,
35,36,37,38,39,40,41,42,43,44,
45,46,47,48,49,50,51,52,53,54,
55,56,57,58,59,60,61,62,63,64
};

static int16 cross_left = 0;
static int16 cross_right = 0;
static int16 cross_road = 0;
static int16 cross_start = 0;
static int16 cross_end = 0;
static int16 lstart = 0;
static int16 rstart = 0;
```

```
const int16 car_center = 352;
static int16 pre_center = 352;


/********** path related variables *************/
static int16 left_nr, right_nr;
static int16 valid_end;    // ÓÐÐ§ÐÐÊý
static int16 valid_start = FALSE;
static int32 target_nr = 0;
// the black line crossed center line
// if right side=1, else side=-1
static int16 black_idx, black_side;
static int16 white_lidx, white_ridx;
static int16 white_idx, white_side;
static int16 pre_black_idx, pre_black_side;
static int16 off_center;
/*******************servo_param***********************/
static int32 servo_Kp = 0;
static int32 servo_Kd = 0;

// track of the current and the previous 7 path_types
static int16 path_type[8] = {0};
static int16 path_nr;// accumulating VSYNCs, used as path_type's idx
static int16 pre_path_nr;
/************************************************/

/*********** commonly used local variables******/
static int16 idx;
static int16 fields = 0;
/************************************************/

/***************** temp variables ****************/

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


/* locally used functions */
static void calc_fps(void);
static void find_path_side(int16 );
static void find_path_center(int16 );
static void check_start(int16 );
static void check_cross(int16 );
static void check_bw_idx(int16 );
static void check_reliability(int16 row);
static void send_image(void);
static void run_time_show(void);
static void init_on_each_field(void);

static void road_calc(void);
static int8 is_on_straight(void);
static int8 is_on_corner(void);
```

```
static int8 is_in_straight(void);
static int8 is_in_corner(void);
static int8 is_to_stop(void);

void get_image(void)
{
    init_on_each_field();
    Bell_Mute();
    /****************** start a field *******************************/
    EnableVSYNCInterruptRising;
    Enable_FB_TS;
    RRST_L();
    (void)START_ADDRESS;
    (void)START_ADDRESS;
    RRST_H();
    /******************* end of starting fifo ************************/

    for(fetch_num=0; fetch_num < max_num; ++fetch_num)
    {
        while( avail_num<2 );

        --avail_num;
        DMA_TCD0_DADDR = (uint32_t)(&image[fetch_num][0]);
        Enable_FB_TS;
        FIFO_DMA_START();
        Wait_FIFO_transfer_complete();

        // valid_start is used to deal with starting/ending line
        if(!valid_start || cross_road)
            find_path_center(fetch_num);
        else
            find_path_side(fetch_num);
    }
    if(Switch3) {
      run_time_show();
    }
    fields++;

}

void find_path_side(int16 row)
{
    static int16 col = 0;
    static int16 search_dir = 0;
    static int16 left_bnd=0, right_bnd=0;
    static int16 inner_width = 40, outer_width = 30;

    cross_left = FALSE;
    cross_right = FALSE;
    inner_width = 40 - row/4;

    if(row > valid_end) {
```

```
            pathL[row] = pathC[row] = pathR[row] = pre_center;
            return ;
        }

        /* search right path */
        /************************* right side line*****************************/
        col = pathR[row-1] + inner_width;
        if(col > 700) col = 700;
        right_bnd = (pathR[row-1] - outer_width < 3) ? 3 : (pathR[row-1] - outer_width);

        for(; col>=right_bnd; col--)
            if(image[row][col] < threshold && image[row][col-1] < threshold)
                break;
        pathR[row] = col;
        if(pathR[row] < right_bnd)
            cross_right = TRUE;
        /*********************************************************************/


        /* search left path */
        /************************* right side line*****************************/
        col = pathL[row-1] - inner_width;
        if(col < 3) col = 3;
        left_bnd = (pathL[row-1] + outer_width > 700) ? 700 : (pathL[row-1] + outer_width);

        for(; col<=left_bnd; col++)
            if(image[row][col] < threshold && image[row][col+1] < threshold)
                break;
        pathL[row] = col;
        if(pathL[row] > left_bnd)
            cross_left = TRUE;
        /*********************************************************************/


        if(row < line_spaced[15]) check_start(row);
        if(row < line_spaced[40]) check_cross(row);
        check_bw_idx(row);

        pathC[row] = (pathL[row] + pathR[row])/2;
        pre_center = pathC[row];

        /* if left cross right, judge as end of valid lines*/
        if((pathL[row] - pathR[row] < 80) && valid_end == max_num)
            valid_end = row - 1;
}


void find_path_center(int16 row)
{
        static int16 col = 0;
        /********************* find path ***************************************/
        /* find right path */
```

```
    for(col=pre_center-1; col>=3; col--){
        if( image[row][col-2]    < threshold &&
            image[row][col-1] < threshold &&
            image[row][col] < threshold )
            break;
    }
    pathR[row] = col;
    if(pathR[0] < 3) pathR[row] = 30;
    /* find left path */
    for(col=pre_center+1; col<=700; col++){
        if( image[row][col+2] < threshold &&
            image[row][col+1] < threshold &&
            image[row][col] < threshold )
            break;
    }
    pathL[row] = col;
    if(pathL[row] > 700) pathL[row] = 670;
    /*************************************************************************/

    // if cur line is starting/ending line, it is not valid start line,
    // the next line should be searched from center instead of side.
    if(pathL[row] - pathR[row] > 200)
        valid_start = TRUE;

    // if needed, tolerance check added here
    pathC[row] = (pathL[row] + pathR[row])/2;
    pre_center = pathC[row];

}


void putint16(int32 tx)
{
    putchar((tx>>8) & 0xff);
    putchar((tx) & 0xff);
}


void servo_calc(void)
{
    static int32 servo_out_ratio = 0;
    static int32 servo_err0 = 0;
    static int32 servo_err1 = 0;
    static int32 near, mid, far,lfar;
    static int32 center_diff;
    static int32 slope_diff;
    static int32 black_diff;
    static int32 index;

    target_nr = (black_idx == max_num) ? max_num-1 : black_idx;
    /********************* calc all diffs ************************************/
    black_diff = (max_num - black_idx);
```

```
slope_diff = 0;
switch(path_type[path_nr]){
      case ON_STRAIGHT:
         far = (pathC[0] + pathC[20]+ pathC[35]) / 3;
         near = car_center;
         center_diff = (far - near) / 3;
         black_diff = black_diff * black_side * 1 / 3;
         slope_diff = (pathC[30] + pathC[35] - pathC[10] - pathC[0]) / 4;
         servo_Kp = 100;
         servo_Kd = -20;
         break;
      case IN_STRAIGHT:
         far = (pathC[0] + pathC[20] + pathC[30]) / 3;
         near = car_center;
         center_diff = (far - near) / 2;
         slope_diff = (pathC[35] + pathC[30] - pathC[10] - pathC[0]) / 4;
         black_diff = black_diff * black_side * 1 / 3;
         servo_Kp = 110;
         servo_Kd = 0;
         break;
      case ON_CORNER:
         far = (pathC[0] + pathC[10] + pathC[20]) / 3;
         near = car_center - black_idx * black_side ;
         center_diff = (far - near) / 2;
         slope_diff = (pathC[black_idx] + pathC[black_idx+1] - pathC[0] - pathC[1]) / 4;
         black_diff = black_diff * black_side * 2;
         servo_Kp = (100 + (30 - white_idx)*2);
         servo_Kd = 20;
         break;
      case IN_CORNER:
         far = (pathC[0] + pathC[15] + pathC[25]) / 3;
         near = car_center - black_diff * black_side * 5 / 2;
         slope_diff = (pathC[30] + pathC[31] - pathC[0] - pathC[1]) / 6;
         center_diff = (far - near) / 2;
         black_diff = black_diff * black_side * 8 / 3;
         servo_Kp = 120;
         servo_Kd = 30;
               break;
      default:
         far = (pathC[line_spaced[5]] + pathC[line_spaced[15]]) / 2;
         near = car_center;
         center_diff = (far - near) / 3;
         black_diff = black_diff * black_side * 2;
         servo_Kp = 100;
         servo_Kd = 0;
            break;
};
if(path_type[path_nr] & (ON_CORNER | IN_CORNER))
      servo_Kp *= (100 + cur_speed) / 4;
else  // if(path_type[path_nr] & (ON_STRAIGHT | IN_STRAIGHT))
      servo_Kp *= 100;
servo_err0 = 0;
```

```
if(Switch1)
      servo_err0 += (center_diff + slope_diff)*servo_Kp / 10000;//00;
if(Switch2)
      servo_err0 += (black_diff)*servo_Kp / 10000;//00;

//if(path_type[path_nr] == ON_CORNER) servo_err0 = (servo_err0 + servo_err1) / 2;
servo_err0 += (servo_err0 - servo_err1)*servo_Kd / 100;//00;

servo_out_ratio = servo_mid + servo_err0;

servo_err1 = servo_err0;
// set servo output ratio
servo_set_ratio(servo_out_ratio);
/*
putchar('I');
PutInt(path_type[path_nr]);
putchar('|');
PutInt(0);
putchar('|');
putchar('0');
putchar('\r');
putchar('\n');
*/
index = (black_side > 0) ? pathL[black_idx] :
      (black_side < 0) ? pathR[black_idx] :
      0;
if(Switch3){
      Disable_FB_TS;
      LCD_CS_L;
      if(servo_err0>0){
         LCD_PutChar(20,280,'L',Red,White);
         LCD_PutInt(20,240,servo_err0,Red,White);
      }
      else{
         LCD_PutChar(20,280,'R',Blue,White);
         LCD_PutInt(20,240,servo_err0,Blue,White);
      }
      LCD_PutInt(40,200,1000,Red,White);
      LCD_PutInt(40,200,slope_diff,Red,White);
      LCD_PutInt(60,200,1000,Magenta,White);
      LCD_PutInt(60,200,center_diff,Magenta,White);
      LCD_PutInt(80,200,1000,Black,White);
      LCD_PutInt(80,200,black_diff,Black,White);
      LCD_PutInt(100,200,1000,Red,White);
      LCD_PutInt(100,200,black_idx,Red,White);
      LCD_PutInt(120,200,1000,Black,White);
      LCD_PutInt(120,200,white_idx,Black,White);
      LCD_PutInt(140,200,1000,Red,White);
      LCD_PutInt(140,200,valid_end,Red,White);
      LCD_PutInt(160,200,1000,Red,White);
      LCD_PutInt(160,200,index,Red,White);
      LCD_PutInt(180,200,1000,Black,White);
```

```
            LCD_PutInt(180,200,path_type[path_nr],Black,White);
            LCD_PutInt(200,200,1000,Black,White);
            LCD_PutInt(200,200,servo_Kp,Black,White);
            LCD_CS_H;
        }

}

void speed_calc(void)
{
        static int16 speed_level = 18;
        static int16 straight_max;
        static int16 straight_min;
        static int16 corner_max;
        static int16 corner_min;

        static int16 straight_step = 15;
        static int16 corner_down_step = 20;
        static int16 corner_up_step = 10;
        static int16 center_diff = 0;

        straight_max = 21 * speed_level;
        straight_min = 21 * speed_level;    //speed_set = 200;
        corner_max = 21 * speed_level;
        corner_min = 20 * speed_level;
        center_diff = (car_center + 50*black_side) - (pathC[0]+pathC[1])/2;
        //center_diff = center_diff * black_side > 0 ? corner_up_step : -corner_down_step;

        // road analysis
        road_calc();

        switch(path_type[path_nr]){
            case ON_STRAIGHT:
                speed_set = straight_max;
                break;
            case IN_STRAIGHT:
                speed_set = (speed_set + straight_step > straight_max) ?
                                straight_max : (speed_set + straight_step);
                break;
            case ON_CORNER:
                if(center_diff > 50)
                    speed_set = (speed_set + corner_up_step) > corner_max ?
                                corner_max : (speed_set + corner_up_step);
                else if(center_diff < -50)
                    speed_set = (speed_set - corner_down_step) < corner_min ?
                                corner_min : (speed_set - corner_down_step);
                                break;
            case IN_CORNER:
                speed_set = corner_min;
                break;
            case STOP_CAR:
                speed_set = 0;
```

```
            break;
        default:
            speed_set = 200;
            break;
    }
}

void road_calc(void)
{
    static int16 straight_nr=0;
    static int16 corner_nr=0;
    static int8 black_meet_bnd = FALSE;

    pre_path_nr = path_nr;
    ++path_nr;
    path_nr &= 0x7;

    if(black_idx <= line_spaced[40])
        black_meet_bnd = (black_idx>=white_idx && black_side*white_side>0);
    else
        black_meet_bnd = FALSE;

    switch(path_type[pre_path_nr]){
    case ON_STRAIGHT:
        if(valid_end == max_num)
            path_type[path_nr] = ON_STRAIGHT;
        else if(valid_end != max_num){
            path_type[path_nr] = IN_CORNER;
            corner_nr = 1;
        }
        break;
    case IN_STRAIGHT:
        if(valid_end == max_num && black_idx >= line_spaced[40])
            path_type[path_nr] = ON_STRAIGHT;
        else if(valid_end >= line_spaced[45]){
            path_type[path_nr] = IN_STRAIGHT;
            //++straight_nr;
            if(straight_nr >= 5) path_type[path_nr] = ON_STRAIGHT;
        }
        else
            path_type[path_nr] = ON_CORNER;
        break;
    case ON_CORNER:
        if(black_side * pre_black_side < 0){
            path_type[path_nr] = IN_CORNER;
            corner_nr = 1;
        }
        else if(valid_end != max_num || black_meet_bnd)
            path_type[path_nr] = ON_CORNER;
        else{
            path_type[path_nr] = IN_STRAIGHT;
            straight_nr = 1;
```

```
            }
            break;
        case IN_CORNER:
            if(valid_end == max_num && !black_meet_bnd)
                path_type[path_nr] = ON_STRAIGHT;
            else if(black_idx >= line_spaced[35]){
                path_type[path_nr] = IN_CORNER;
                //++corner_nr;
                //if(corner_nr >= 6) path_type[path_nr] = ON_CORNER;
            }
            else
                path_type[path_nr] = ON_CORNER;
            break;
        case STOP_CAR:
                path_type[path_nr] = STOP_CAR;
            break;
        default:
                path_type[path_nr] = ON_STRAIGHT;
            break;
        }

        if(is_to_stop())
            path_type[path_nr] = STOP_CAR;

}



void run_time_show(void)
{
        /* calling this func should set_window fisrt in caller func */
        static int16 col=0;
        int16 l,r,m;
        for(col=0; col<max_num; ++col){
            l = pathL[col];
            r = pathR[col];
            m = pathC[col];
            run_time_data[l>>2] = USER_RGB(Magenta);
            run_time_data[r>>2] = USER_RGB(Blue);
            run_time_data[m>>2] = USER_RGB(Red);
            run_time_data[car_center>>2] = USER_RGB(Green);

            DMA_TCD1_SADDR = (uint32_t )(&run_time_data[0]);
            DMA_TCD1_NBYTES_MLNO = 352;
            DMA_TCD1_SLAST = (unsigned int)(-352);

            /* start lcd dma transfer*/
            Disable_FB_TS;
            LCD_CS_L;
            LCD_DMA_START();
            Wait_LCD_transfer_complete();
            LCD_DMA_START();
```

```
            Wait_LCD_transfer_complete();
            LCD_CS_H;

            /* reset data to White */
            run_time_data[l>>2] = run_time_data[r>>2]
                                = run_time_data[m>>2]
                                = run_time_data[car_center>>2]
                                = USER_RGB(Gray1);
        }

        if(bluetooth_on){
            send_image();
            bluetooth_on = FALSE;
        }

}

void real_time_show(void)
{
        /* init the the vars, according to diff conditions*/


        /* real tmp vars*/
        int i,j,k;
        uint8 tmp;
        uint8 *img_ptr;
        uint16 *rgb;
        int8 send_img = FALSE;

        img_ptr = &(image[0][0]);
        rgb = (uint16*)(&(image[1][0]));

        DMA_TCD0_DADDR = (uint32_t)img_ptr;
        DMA_TCD1_SADDR = (uint32_t)rgb;
        DMA_TCD1_NBYTES_MLNO = 640;
        DMA_TCD1_SLAST = (unsigned int)(-640);


        while(1)
        {
            if(Switch2){
                wanted_num = wanted_num_spaced;
                max_num = max_num_spaced;
            }
            else{
                wanted_num = wanted_num_debug;
                max_num = max_num_debug;
            }
            Disable_FB_TS;
            LCD_CS_L;
            if(Switch2)
                set_window(0,0,3*max_num-1,VMaxPosition);
```

```
else
      set_window(0,0,max_num-1,VMaxPosition);
LCD_CS_H;
EnableVSYNCInterruptRising;
// while(!fifo_recv_done && avail_num < 2) ;
Enable_FB_TS;
RRST_L();
(void)START_ADDRESS;
(void)START_ADDRESS;
RRST_H();

send_img = FALSE;
if(global_flag) {
   send_img = TRUE;
   //delay_ms(500);
   global_flag = FALSE;
}

for(i=0; i<max_num; ++i)
{
      while( avail_num < 2 );
      --avail_num;
      Enable_FB_TS;
      FIFO_DMA_START();
      Wait_FIFO_transfer_complete();

      for(k=0; k<64; ++k)
      {
            for(j=0; j<5; ++j)
            {
                  if(Switch3)
                     tmp = img_ptr[2*j+k*11] > threshold ? 255 : 0;
                  else
                     tmp = img_ptr[2*j+k*11];
                  rgb[j+k*5] = (((uint16_t)((tmp>>3)|((tmp<<3)&0xE0)))<<8)
                              |(tmp&0xF8)|(tmp>>5);
                  if(j+k*5 == 160) rgb[j+k*5] = USER_RGB(Red);
                  if(i == wanted_num_spaced[line_spaced[15]]) rgb[j+k*5] = USER_RGB(Blue);
            }
      }

      if(send_img)
            for(idx=0; idx<704; ++idx)
                  putchar(img_ptr[idx]);

      Disable_FB_TS;
      LCD_CS_L;
      LCD_DMA_START();
      Wait_LCD_transfer_complete();
      if(Switch2){
                  LCD_DMA_START();
                  Wait_LCD_transfer_complete();
```

```
                    LCD_DMA_START();
                    Wait_LCD_transfer_complete();
                }
                LCD_CS_H;
            } // end for
            calc_fps();
            if(!Switch4) return ;
        } // end while

}

void check_start(int16 row)
{
        static int16 start_tolerance = 200;
        static int16 col = 0, l_col = 0, r_col = 0;
        static int16 cnt = 0;
        static int16 center_state;

        if(track_nr==0 && row < line_spaced[15]){
           l_col = pre_center + 200;
           r_col = pre_center - 200;
           if(l_col > 700) l_col = 700;
           if(r_col < 3) r_col = 3;

           // search left start line, meeting three black point,stop
           col = pre_center;
           for(; col <= l_col; col+=3)
                   if(image[row][col] < threshold &&
                      image[row][col+2] < threshold)
                     break;
           l_col = col;

           col = pre_center;
           for(; col >= r_col; col-=3)
                   if(image[row][col] < threshold &&
                      image[row][col-2] < threshold)
                     break;
           r_col = col;

           col = l_col - r_col;
           if(l_col < 600 && r_col > 100 && col > 80 && col < 200){
                   track_cycle++;
                   track_nr = 100;
                   Bell_Ring();
           }
        }

}

void check_bw_idx(int16 row)
{
```

```
    if(pathL[row] <= car_center && black_idx == max_num){
        black_idx = row;
        black_side = -1;
    }
    if(pathR[row] >= car_center && black_idx == max_num){
        black_idx = row;
        black_side = 1;
    }

    if(cross_left    && white_lidx == max_num){
        white_lidx = row;
    }
    if(cross_right && white_ridx == max_num){
        white_ridx = row;
    }

    if(black_side > 0 && white_lidx != max_num) {
      white_idx = white_lidx;
      white_side = 1;
    }
    else if(black_side < 0 && white_ridx != max_num){
      white_idx = white_ridx;
      white_side = -1;
    }

}


void check_cross(int16 row)
{
    static int16 idx, lstep, rstep;
    if(!cross_left && !cross_right)
        return;

    if(cross_left && cross_right){
        pathL[row] = pathL[row-1] - 3;
        pathR[row] = pathR[row-1] + 3;
    }
    else if(cross_left){
      pathL[row] = (pathL[row-1] + 3 > 703) ? 703 : (pathL[row-1] + 3);
        if(pathR[row] < pathR[row-1])
            pathR[row] = pathR[row-1] + 3;
    }
    else{
      pathR[row] = (pathR[row-1] - 3 < 0) ? 0 : (pathR[row-1] - 3);
        if(pathL[row] > pathL[row-1])
            pathL[row] = pathL[row-1] - 3;
    }
}
```

```
void real_time_show_aux(void)
{
    if(Switch4)
    {
        disable_speed_pit();
        motor_set_ratio(0);
        real_time_show();
        enable_speed_pit();
    }
}


void run_time_show_aux(void)
{
        // following data need to be inited on entering run time show
        int32 i=0;
        wanted_num = wanted_num_spaced;
        max_num = max_num_spaced;
        for(idx = 0; idx<176; ++idx)
            run_time_data[idx] = USER_RGB(Gray1);

        /// clear screen
        i = (HMaxPosition+1) * (VMaxPosition+1);
        Disable_FB_TS;
        LCD_CS_L;
        set_window(0,0,HMaxPosition,VMaxPosition); // 176x100
        while(i--){
            LCD_WriteRAM(White);
        }
        LCD_CS_H;

        Disable_FB_TS;
        LCD_CS_L;
        LCD_PutString(40,300,"sdif",Red,White);
        LCD_PutString(60,300,"cdif:",Magenta,White);
        LCD_PutString(80,300,"bdif:",Black,White);
        LCD_PutString(100,300,"bidx:",Black,White);
        LCD_PutString(120,300,"widx:",Red,White);
        LCD_PutString(140,300,"vend:",Red,White);
        LCD_PutString(160,300,"both:",Black,White);
        LCD_PutString(180,300,"type",Black,White);
        LCD_PutString(200,300,"Kp",Black,White);
        LCD_CS_H;

        while(!Switch4){
            get_image();
            speed_calc();
            while(GetCycleTime()<timer_out) ;
            ResetCycleTime();
            servo_calc();
        }
```

```
}

void init_on_each_field(void){

	/* init the two vars, according to diff conditions*/
	/* these two should be inited in a higher level func*/
	pre_black_side = black_side;
	pre_black_idx = black_idx;
	black_idx = max_num;   // each field set black_idx = 0
	black_side = 0;
	white_idx = max_num;
	white_side = 0;
	white_lidx = white_ridx = max_num;

	left_nr = max_num;
	right_nr = max_num;
	valid_end = max_num;
	valid_start = FALSE;

	pre_center = pathC[0];
	// assume road is not cross_road on each field
	cross_road = FALSE;
	cross_left = FALSE;
	cross_right = FALSE;
	cross_start = max_num;
	cross_end = max_num;



		/*****************************************************/
		Disable_FB_TS;
		LCD_CS_L;
		set_window(0,0,max_num*2-1,175); // 176x100
		LCD_CS_H;
		/*****************************************************/
}


void calc_fps(void)
{
	/* fps related*/
	static uint32 field;
	static uint32 time_passed;
	static uint32 fps;

	++field;
	time_passed += (GetCycleTime()+7499)/7500;
	if( time_passed >= 50){
			fps = field;
			field = 0;
			time_passed = 0;
	}
```

```
    Disable_FB_TS;
    LCD_CS_L;
    LCD_PutInt(40,240,fps,Red,White);
    LCD_CS_H;
}


void send_image()
{
  int16 i,j;
  for(j=0; j<max_num; ++j)
  for( i=0; i<704; ++i)
    putchar((image[j][i]));
}


void show_stop(void)
{
        Disable_FB_TS;
        LCD_CS_L;
        LCD_PutString(220,280,"stop",Red,White);
        LCD_CS_H;
}

int8 is_to_stop(void)
{
    if(stop_flag){
        show_stop();
        return TRUE;
    }
    if(track_nr > 0) track_nr--;
    if(track_cycle == cycle_to_run    && !stop_flag) {
        time_to_stop = 6;
        track_cycle = 0;
    }

    if(time_to_stop){
        if(time_to_stop == 1) stop_flag = TRUE;
        else        time_to_stop--;
    }

    return FALSE;
}
```