第十一届“恩智浦”杯全国大学生

智能汽车竞赛

技 术 报 告

基于图像识别的的智能车 B 车控制算法研究

学　　校：　厦门大学

队伍名称：　南强至臻队

参赛队员：　刘海林

陈默含

王中豪

带队教师：　李继芳　　陈华宾

关于技术报告和研究论文使用授权的说明

本人完全了解第十一届“恩智浦”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，比赛组委会和恩智浦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：刘海林 王檬 陈默含

带队教师签名：李继芳 陈华宾

日　　期：2016.8.14

摘　要

本文是关于基于第十一届全国大学生“恩智浦”杯智能车竞赛设计的智能车，整车的系统以 MK60N512VMD100 微控制器为核心控制单元，通过面阵 CCD 检测识别赛道，并使用光电编码器检测模型车的实时速度，利用 PID 控制算法调节驱动电机的转速和转向舵机的角度，实现了对模型车运动速度和运动方向的闭环控制。为了提高模型车的速度和稳定性，使用上位机、无线蓝牙、键盘液晶模块等调试工具，进行了大量硬件与软件测试。实验结果表明，该系统设计方案确实可行。

关键词：MK60　面性 CCD　PID 控制

目录

第一章 引言

随着信息技术的发展，汽车的电子化模块越来越多，智能汽车领域受到了大量的关注。受教育部高等教育司委托(教高司函[2005]201 号文),高等学校自动化专业教学指导分委员会主办“恩智浦”杯全国大学生智能汽车竞赛，以迅猛发展的汽车电子为背景，涵盖了控制、模式识别、传感技术、电子、电气、计算机、机械等多个学科交叉的科技创意性比赛。

参赛选手须使用竞赛秘书处统一指定并负责采购竞赛车模，自行采用 32 位微控制器作为核心控制单元，自主构思控制方案及系统设计，包括传感器信号采集处理、控制算法及执行、动力电机驱动、转向舵机控制等，完成智能汽车工程制作及调试，于指定日期与地点参加场地比赛。参赛队伍之名次（成绩）由赛车现场成功完成赛道比赛时间为主，技术方案及制作工程质量评分为辅来决定。

本届“飞思卡尔”杯全国大学生智能汽车竞赛比赛分为电轨组，电磁平衡组，光电组，摄像头组，双车组，以及信标组六个组别分别进行比赛，我队参与摄像头组比赛。此次比赛按照官方规定采用 B 型车模，使用面性 CCD 组进行检测。

本篇技术报告主要包括机械系统、硬件系统、软件系统等，详尽地阐述了我们的设计方案，具体表现在硬件电路的创新设计以及控制算法的独特想法，我队参与比赛来的经验之谈，希望能给大家带来帮助

第二章 智能车整体设计

2.1 整车设计思路

智能车主要由三个部分组成：检测系统，控制决策系统和动力系统。其中 检测系统采用鹰眼摄像头，控制决策系统采用 MK60 作为主控芯片动力系统主要控制 SD-5 舵机的转角和直流电机的转速。整体的流程为，通过数字摄像头来检测前方的赛道信息，并将赛道信息发送给单片机。同时，通过光电编码器构成的反馈渠道将车体的行驶速度信息传送给主控单片机。根据所取得的赛道信息和车体当前的速度信息，由主控单片机做出决策，并通过 PWM 信号控制直流电机和舵机进行相应动作，从而实现车体的转向控制和速度控制。

系统设计框图：

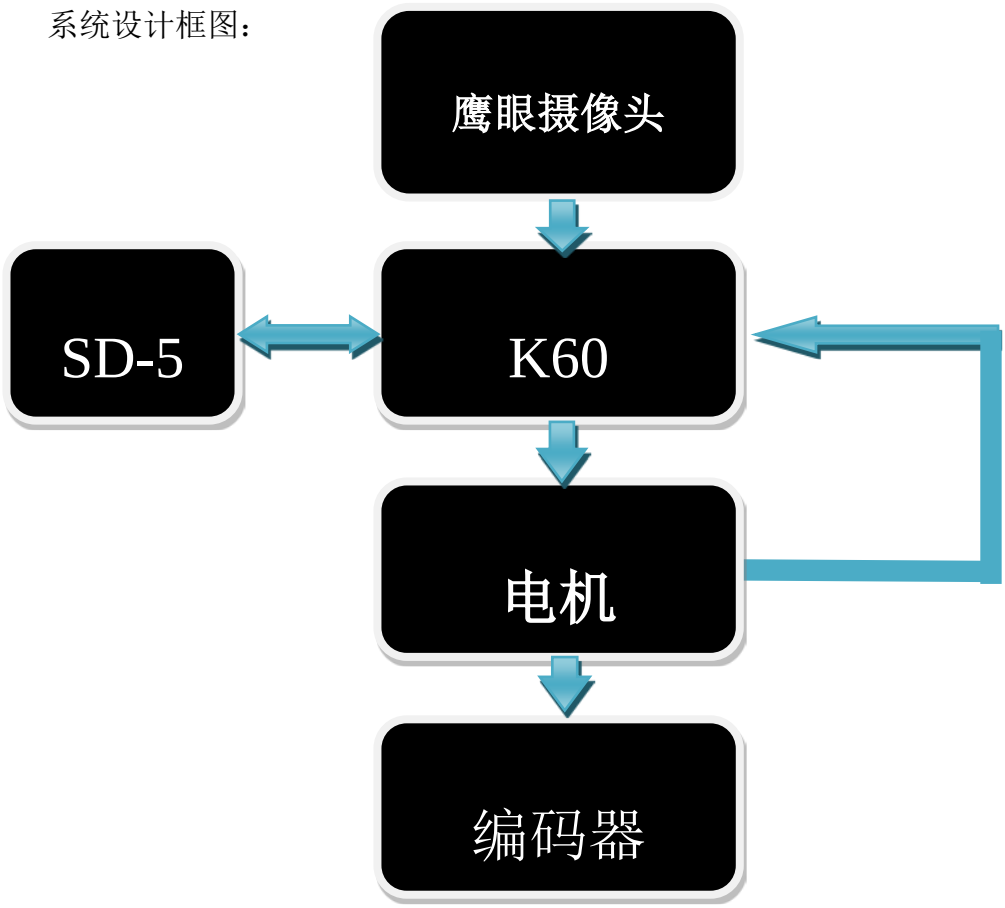


图 2.1 摄像头组 B 车控制流程图

2.2 车模整体布局及参数

本次比赛光电组采用的是 B 型车模，尺寸为 270mmx170x96mm，所配置的驱动电机为 RS-540，工作电压 DC 7.2V，空载转速 23400RPM±10%，最大功率 61.75W。伺服器为 SD5 数字伺服器，工作电压 4.5-5.5v，带堵转保护电路，扭力 5.0kg/cm，动作速度≤0.14±0.02sec/60°。

车模整体布局如下图所示：

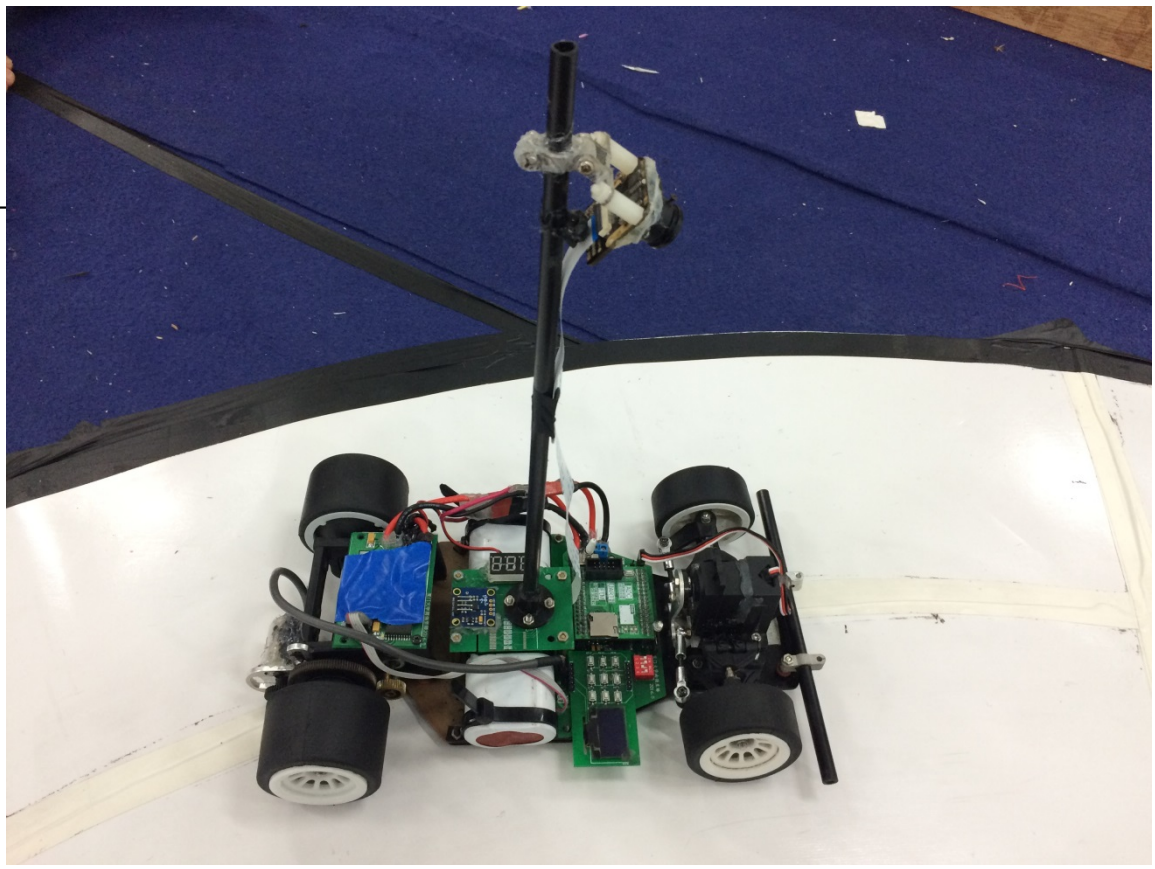


图 2.2 C 车整体结构

整体布局有以下特点：

（1）车模底盘降低，同时主板以及各部件贴近底盘，以尽可能降低重心

（2）舵机采用立式安装方式

（3）用轻便坚固玻纤杆作为摄像头支撑杆的材料。

（4）主板驱动分离

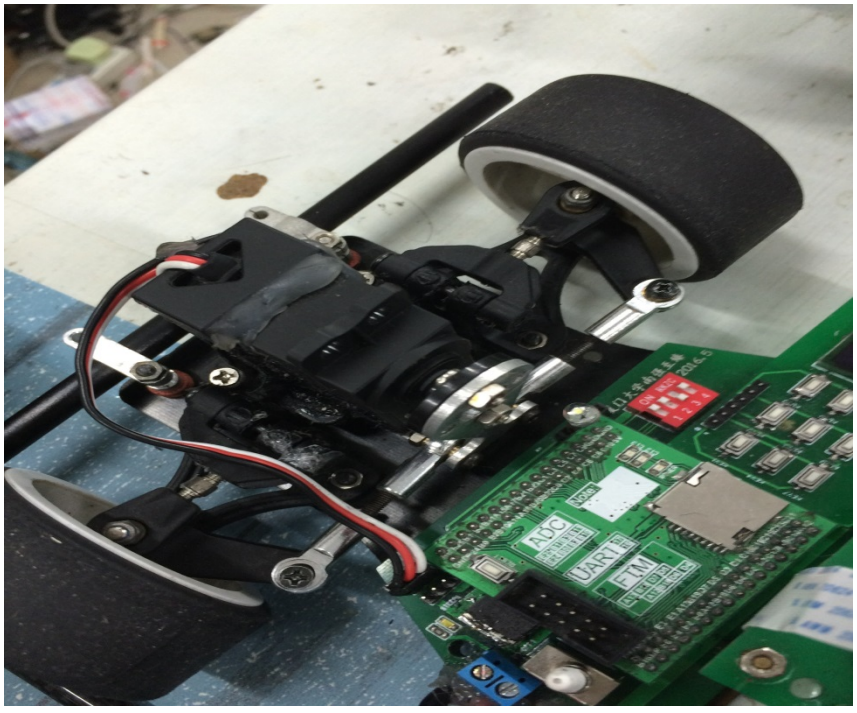
第三章　机械结构设计

根据比赛规则要求，第十一届摄像头组采用 B 车模。固以下通过前轮定位、舵机安装、后轮驱动与差速调节、传感器安装、车模加固及重心降低五个方面就对 B 车模的改造进行介绍。

3.1 舵机的安装方式

参考往年的舵机安装方式我们发现舵机有立式安装和卧式安装两种,比较两种安装方式，我们可以发现力臂较短的连接方式优点是能够输出更大力矩，调节精度更高但是不足的是反应速度不够快,而对于长的连接方式优点是反应速度快，调节精度低，但是输出力矩不足，所以综合考虑舵机的反应特性和输出、极限转角以及实际调车过程的现象，我们决定将舵机安装方式定位正立，摆臂的原点略远。

舵机安装方式如下图所示：



3.2 重心调整

通过重心的调整，可使模型车转弯时更加稳定、高速。其调整主要分为重 心高度的调整以及重心在整车上局部分布的调整。考虑到车子的稳定性，在保证车模顺利通过坡道以及障碍的前提下，我们尽可能的降低车子的重心。同时均匀车身重量，使重心在整车的中轴线上。由于靠前的重心会造成舵机负担，过后的重心又会导致侧滑，经过多次试验，我们找到了一个合理的位置安排重心。

3.3 前轮定位

前轮定位的主要目的在于提高车模的转向控制的灵敏性和提高车模的抓地力，主要包括：主销后倾、主销内倾和前轮前束。

3.3.1 主销后倾

主销后倾（见图 3.2）是指前轮主销在车模的纵向平面内（小车的侧面） 有一个向后的倾角γ,即主销轴线与地面垂直线在车模在纵向平面的夹角,称 为“主销后倾角”。采用主销后倾，车模在车轮偏转后会产生一回正力矩，纠 正车轮偏转。车模的主销后倾不可直接调教,在此我们采用预设的主销后倾角，大概为 5 度。

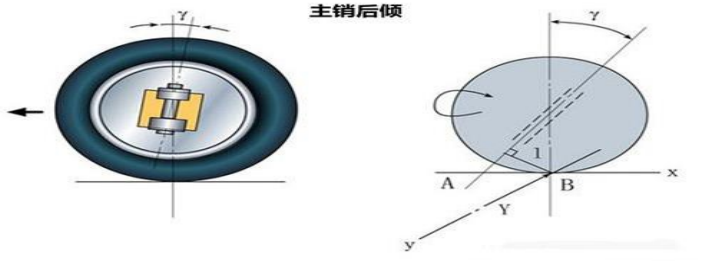


图 3.2 主销后倾

3.3.2 主销内倾

前轮主销在车模的横向平面内向内倾斜一个β角,即主销轴线与地面垂直线 在车模的横向断面内的夹角，称为“主销内倾角”（见图 3.3）。

主销内倾角 β 也有使车轮自动回正的作用。当转向轮在外力作用下发生偏转 时，由于主销内倾的原因，车轮连同整个车模的前部将会被抬高；外力消失后， 车轮在重力作用下恢复到中间位置。另外主销内倾还会使主销线延长线与赛道的交点到车轮中心平面的距离减小,同时转向时赛道作用在转向轮上的阻力矩也会减小，从而减小转向阻力，使转向轻便，灵敏。

轮胎调整为倾斜以后直线行走的时候是轮胎内侧着地,而当过弯的时候,由 于惯性车体会要向弯道外侧倾斜,而这时候的外侧轮胎如果倾斜角度事先调整得 当则正好可以胎面着地,从而使车辆在弯道获得最佳抓地力。而如果事先把轮胎 装成是完全垂直于地面的话,尽管直线行驶的时候轮胎是胎面着地,而到弯道就 会变成轮胎外侧着地了,那样将大大减小轮胎与地面的有效接触面积,从而导致 弯道时抓地力大幅降低。主销内倾角不可过大,否则在直道和弯道时均使用轮胎 内侧，导致轮胎严重磨损，抓地力极低。

由于赛道一般会设有连续弯道,车模需在高速下连续转弯,在此我们将车模 主销内倾调为-8 度左右。

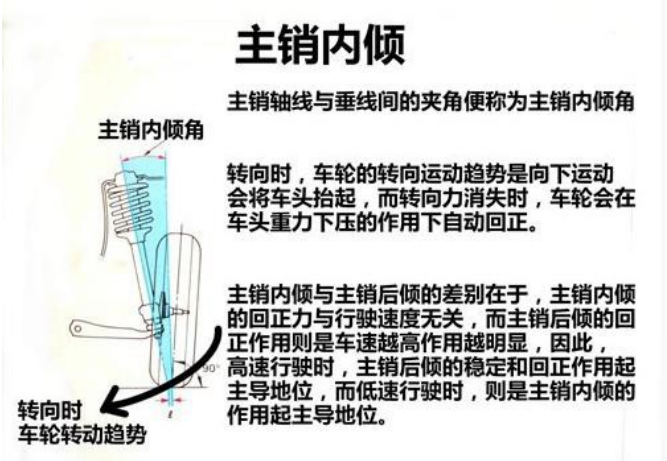


图 3.3 主销内倾

3.3.3 前轮前束

前束是转向灵敏度与稳定性的权衡。前束不可以无限度增大，太大了的话，直道行驶时车轮与地面发生的就不是滚动摩擦，而是滑动摩擦，轮胎磨损将急剧增大，且会导致阻力加大，降低直道速度。过度减小，会导致稳定性降低，车辆抖动，难以操控。为了增加车子的转向性能以及满足阿克曼转弯原理，我们将车子的前束调教为大概 5 度左右。具体见图 3.4：

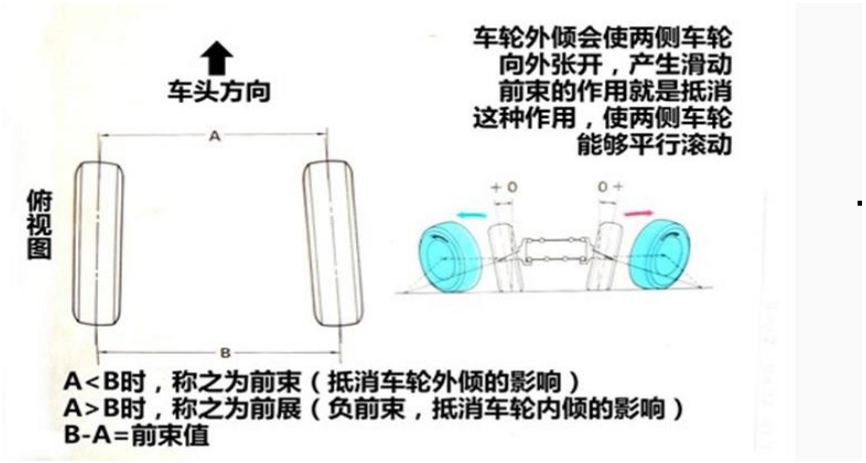


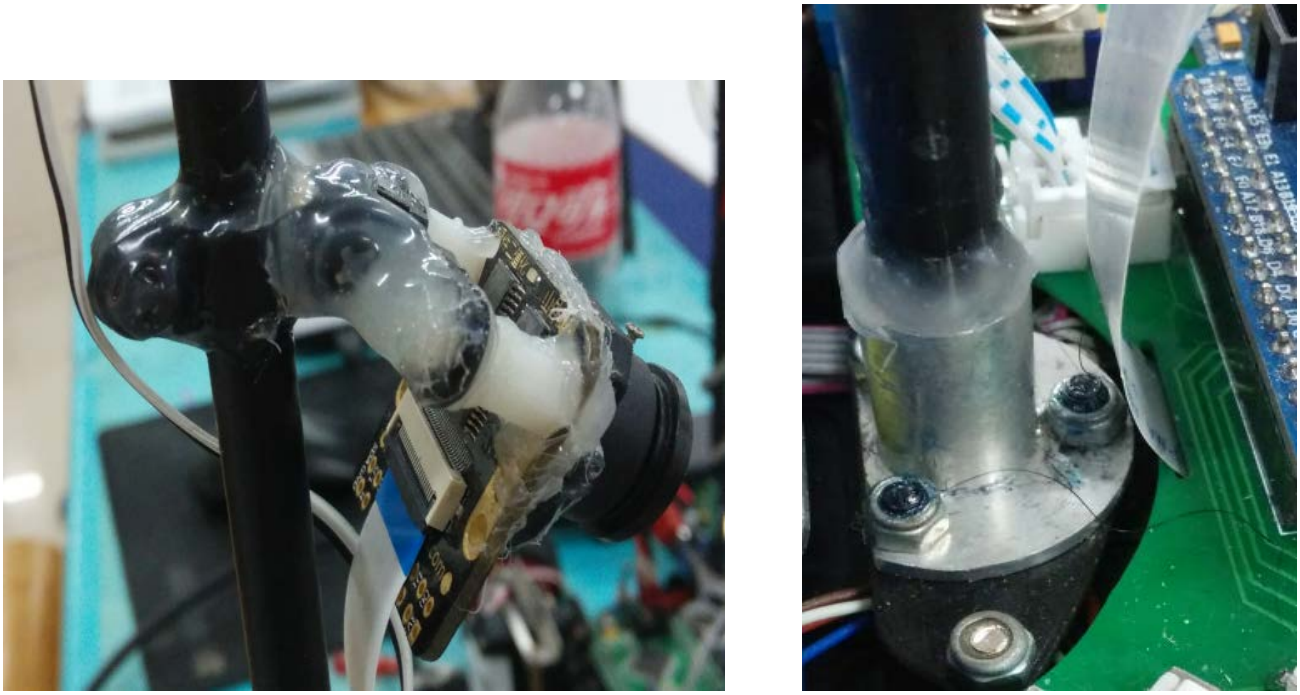
图 3.4 前轮前束

3.4 摄像头安装

摄像头是整辆车的眼睛，摄像头的安装是最重要的。摄像头的安装要求使摄像头位于整个车模的中心位置，而且高度要适合于图像的采集和处理。通过多组对比试验，我们最后决定使用单杆结构来固定摄像头，单杆使得整个支架十分简约，但也会带来摄像头易抖动的问题。这时，单杆与底盘之间的连接可靠性就显得尤为重要，如图3.3 所示。为了严格的控制整辆车重量和降低车重心，我们采用树脂碳杆作为摄像头的支架。单纯的碳杆刚性太强，遇到碰撞可能会发生折断，树脂碳杆可以在保证低质量的同时，也保证了高强度。我们自己设

计并制作了摄像头固定装置，该结构重量轻，可以方便的对摄像头进行调 整和校正，如图3.7 所示。

另外，摄像头的高度也会影响其抖动，架的越高则越容易抖动。为了图像的稳定可靠，我们最终把摄像头架在了与地面的距离为35cm左右的位置。



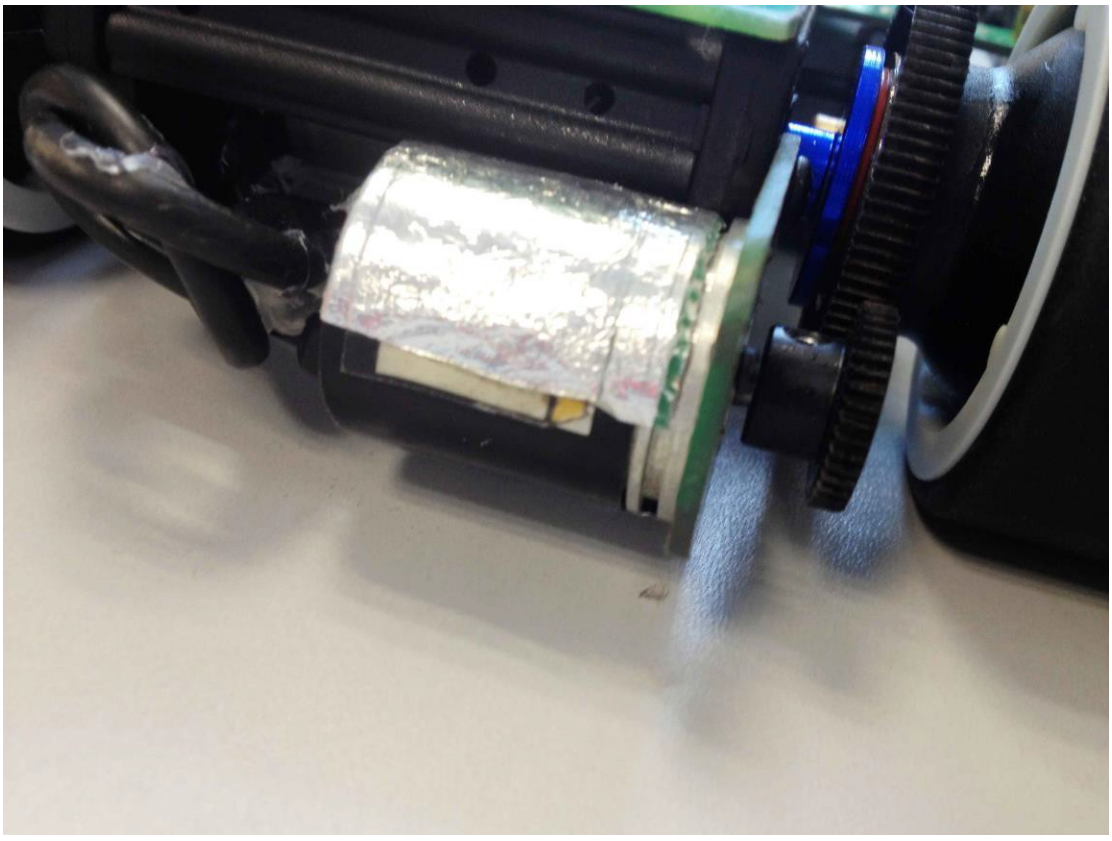
(a)　　(b)

图 3.7　摄像头及支架安装 （a）摄像头的固定（b）杆与底座的连

3.5　编码器安装方式

速度传感器一般可以选择对射式光栅或光电编码器。对射式光栅的重量轻，阻力小精度也高，然而光栅暴露在外界容易受到外界光线或粉尘等的影响，导致计数不准确；而光电编码器就不存在此类问题。所以最后我们选择了编码器，该编码器线数 500 线，可以达到很高的精度，符合我们得要求。

在安装编码器的时候要保证有合适的齿轮咬合。咬合完美的原则是：两个传动齿轮轴保持平行，齿轮间的配合间隙要合适，过松容易打坏齿轮，过紧又会增加传动阻力；传动部分要轻松、顺畅，容易转动。判断齿轮传动是否调整好的一个依据是，听一下电机带动后轮空转时的声音。声音刺耳响亮，说明齿轮间的配合间隙过大，传动中有撞齿现象；声音闷而且有迟滞，则说明齿轮间的配合间隙过小，咬合过紧，或者两齿轮轴不平行，电机负载加大。调整好的齿轮传动噪音小，并且不会有碰撞类的杂音。根据编码器的机械参数，我们自行设计和加工了轻巧的零件来使编码器与电机合适地咬合在一起如图 3.6 所示



3.6 驱动轮调整

车模前轮采用的电机由竞赛主办方提供。齿轮传动机构对车模的驱动能力有很大的影响。齿轮传动部分安装不恰当，会增大电机驱动前轮的负载；齿轮配合

间隙过松则容易打坏齿轮过紧则会增加传动阻力,所以我们在电机安装过程中尽量使得传动齿轮轴保持平行，传动部分轻松、流畅，不存在卡壳或迟滞现象。

差速机构的作用是在车模转弯的时候，降低前轮与地面之间的滑动；并且还可以保证在轮胎抱死的情况下不会损害到电机。差速器的调整中要注意滚珠轮盘间的间隙，过松过紧都会使差速器性能降低，转弯时阻力小的车轮会打滑，从而影响车模的过弯性能。好的差速机构，在电机不转的情况下，右轮向前转过的角度与左轮向后转过的角度之间误差很小，不会有迟滞或者过转动情况发生。当车辆在正常的过弯行进中(假设：无转向不足亦无转向过度)，此时 4 个轮子的转速 (轮速)皆不相同，依序为：外侧前轮＞外侧后轮＞内侧前轮＞内侧后轮。差速器的特性是阻力越大的一侧，驱动齿轮的转速越低；而阻力越小的一侧，驱动齿轮的转速越高。前轮差速的调整主要是调整差速器中差速齿轮的咬合程度，差速的松紧与自己所要求的速度相匹配，已达到自己想要的状态。

第四章 电路设计

电源模块为小车系统的其他各模块提供所需要的电源,在整个智能车中充当着单片机和外部硬件间信息传递的桥梁,既能把单 片机中的程序命令传达给硬件，又能把从外部硬件读到的信号反馈给单片机，是 整个智能车的关键。。设计中，除了需要 考虑电压的范围和电流容量等基本参数外，还要在电源转换效率，降低噪声， 防止干扰和电路简洁方面进行优化。可靠的电源方案是整个硬件电路稳定可靠 运行的基础。

由于带动整个小车系统工作的能量均来自一块由6 颗1.2V 镍镉充电电池串联而成的 7.2V 电池，容量为 1800mAh。而系统中各模块所需的工作电压和工作电流各不相同，因此电源模块应该包括多个稳压电路，将电池电压转换为各个模块所需的电压，具体的电路结构如图 4.1 所示：

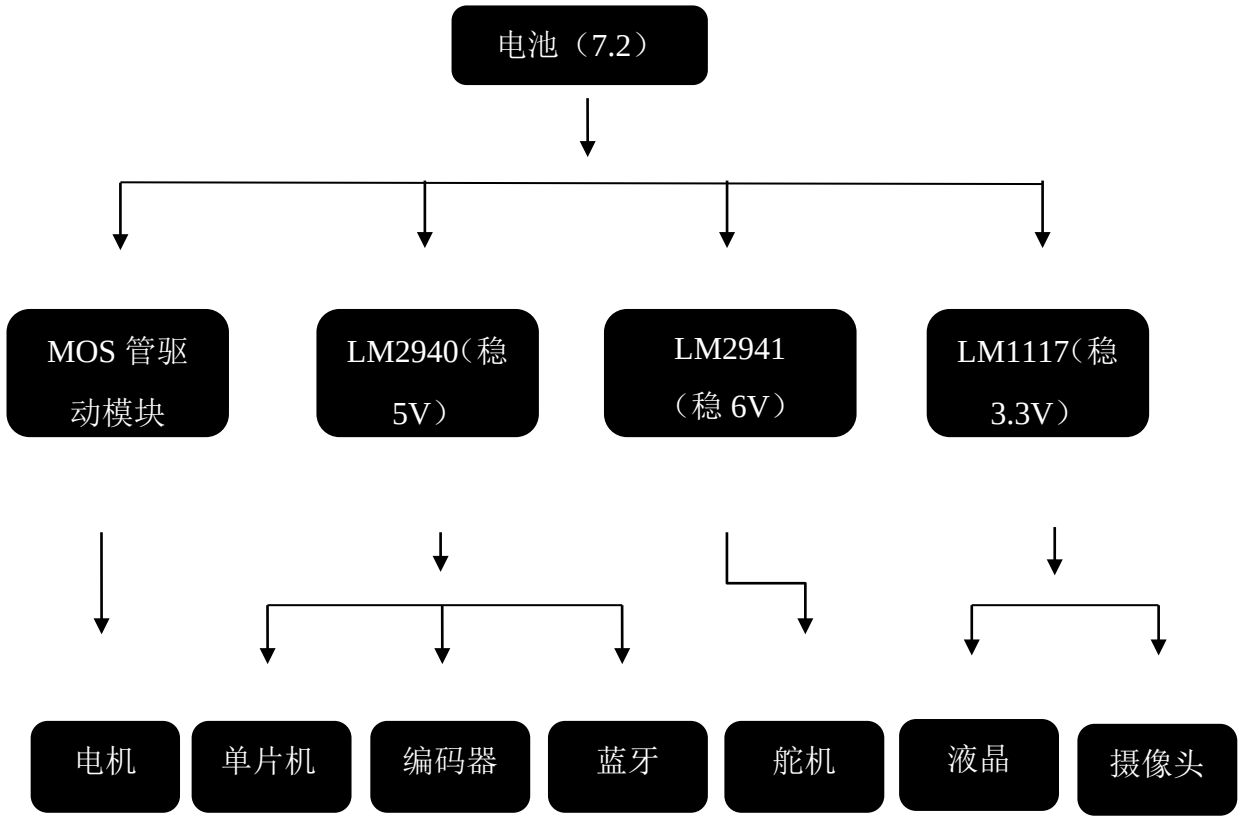
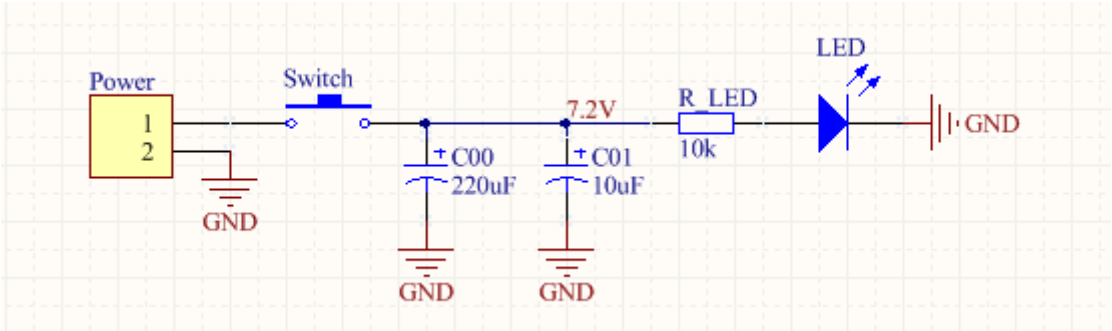


图 4. 1 电源供电模块结构图

4.1 电源模块

整个车模由 7.2V 电池（如图 4.2）提供总电源，配上开关、指示灯、及电容，原理图如图 4.2。电容应该尽可能放置在靠近电池电源处，以起到对电源的滤波作用。如图 4.2 所示：

(a)



(b)

图 4.2 电源模块 （a）7.2V 电池 (b) 电源模块电路原理图

4.2 5V 稳压模块

线性稳 5V 稳压芯片很多，比如 7805、TPS7350、LM2940-5.0、LM 1084-5.0 等，经过实践使用发现，LM2940 的线性度较为理想，且电路简单、稳定性好。所以 5V 稳压芯片我们选择 LM2940。

通过 LM2940 芯片对 7.2V 稳压得到 5V 电压，原理图及实物图如图 4.3，给放大电路、驱动电路等模块供电；另取一片 LM2940 芯片稳压得到 5V 电源，给编码器供电。此处编码器单独供电是为了避免编码器对放大和驱动电路产生干扰。

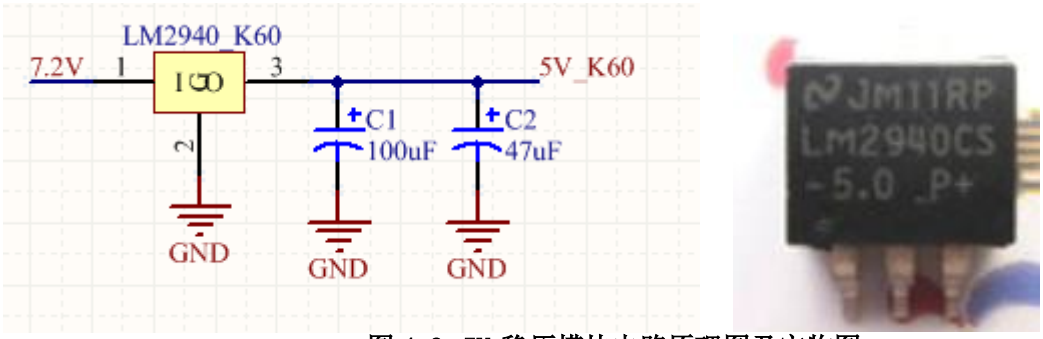


图 4.3 5V 稳压模块电路原理图及实物图

4.3 3.3V 稳压模块

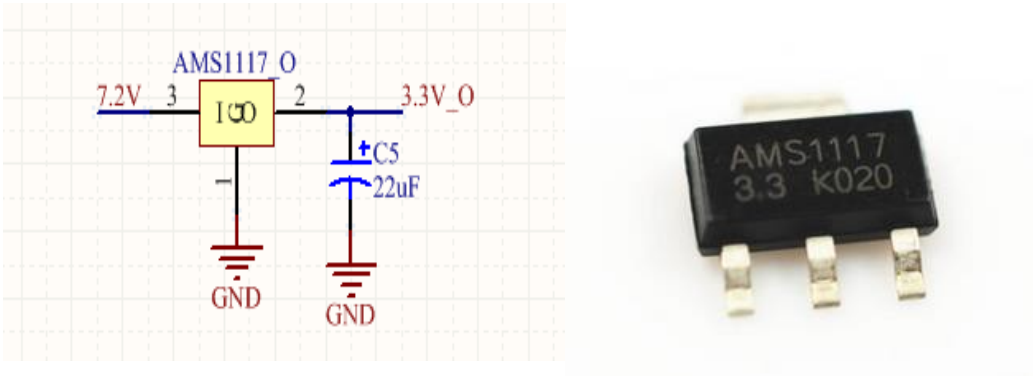


图 4.4 3.3V 稳压模块电路原理图及实物图

稳 3.3V 稳压芯片有线性电源：ASM1117-3.3、LM2940-3.3、LM1084-3.3 开关电源：LM2596。经过比较，ASM1117 的线性度较好且价格便宜，效率高，不易发热，但有纹波，电路比较复杂，适用于单片机供电。通过 LM1117 芯片对 5V 稳压得到 3.3V 电压，原理图及实物图如图 4.4，给单片机、液晶屏供电。此处采用对 5V 稳而非 7.2V 稳压是为了减少损失的电压，从而提高稳压的快速性和稳定性。

4.4 6V 稳压模块

6V 稳压芯片我们选择 LM2941。LM2941 芯片为可调稳压模块，输入电压为 7.2V，调节 R1 和 R2 的值可改变输出电压。本车将电压调到 6V 电压给舵机供电。原理图及实物图如图 4.5 所示：

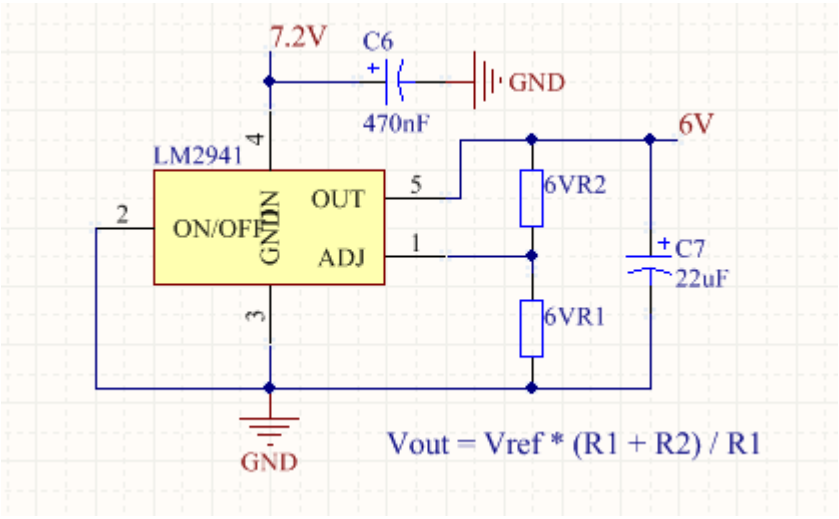
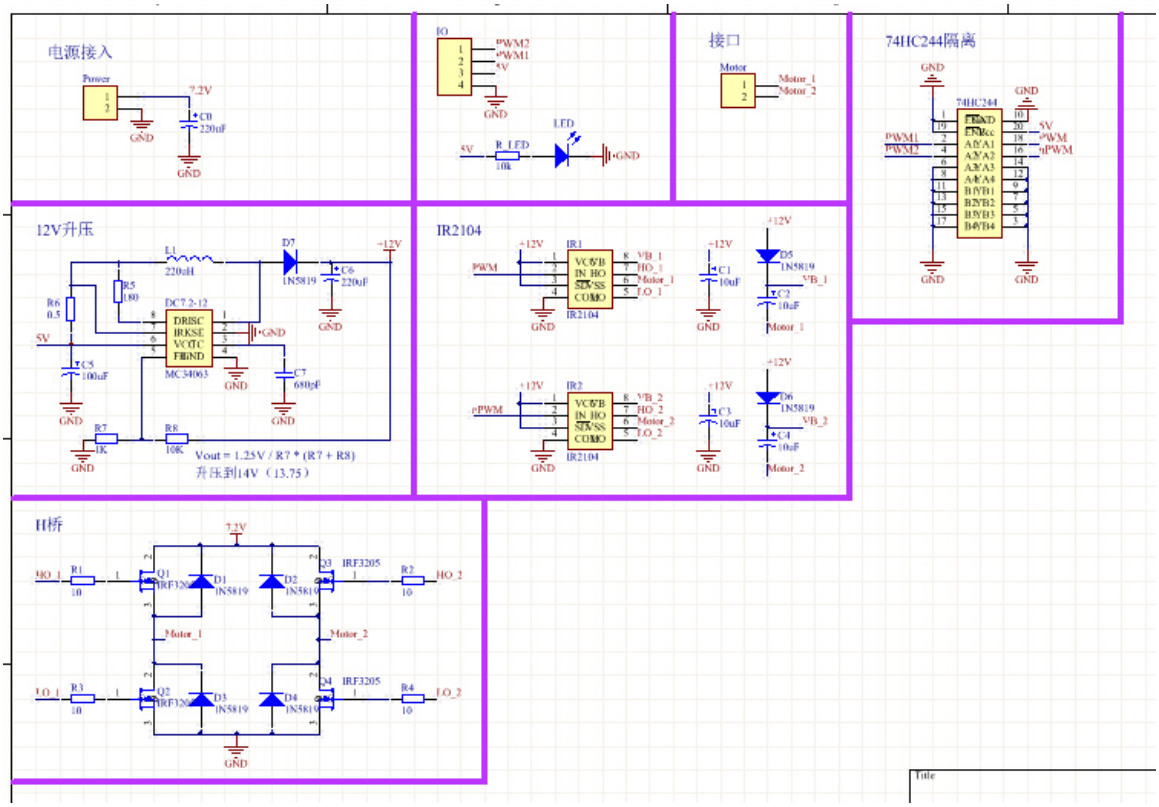


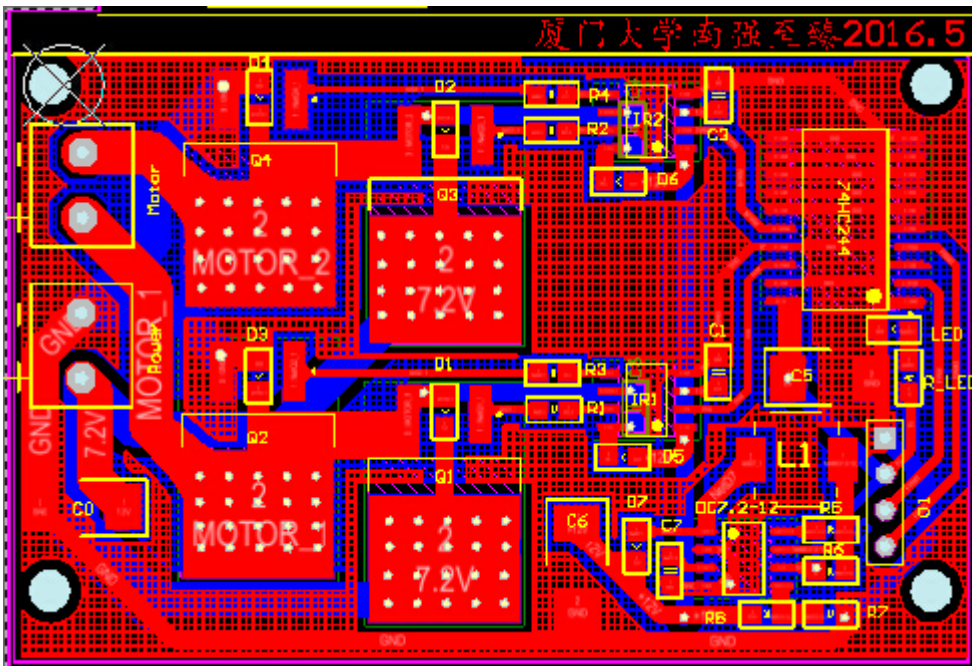
图 4.5 6V 稳压模块电路原理图及实物图

4.5 驱动电路模块

MOS 管最显著的特性是开关特性好，所以被广泛应用在需要电子开关的电路中，常见的如开关电源和马达驱动，也有照明调光。我们采用 MOS 管驱动电机，如图 4.6 所示：



(a)



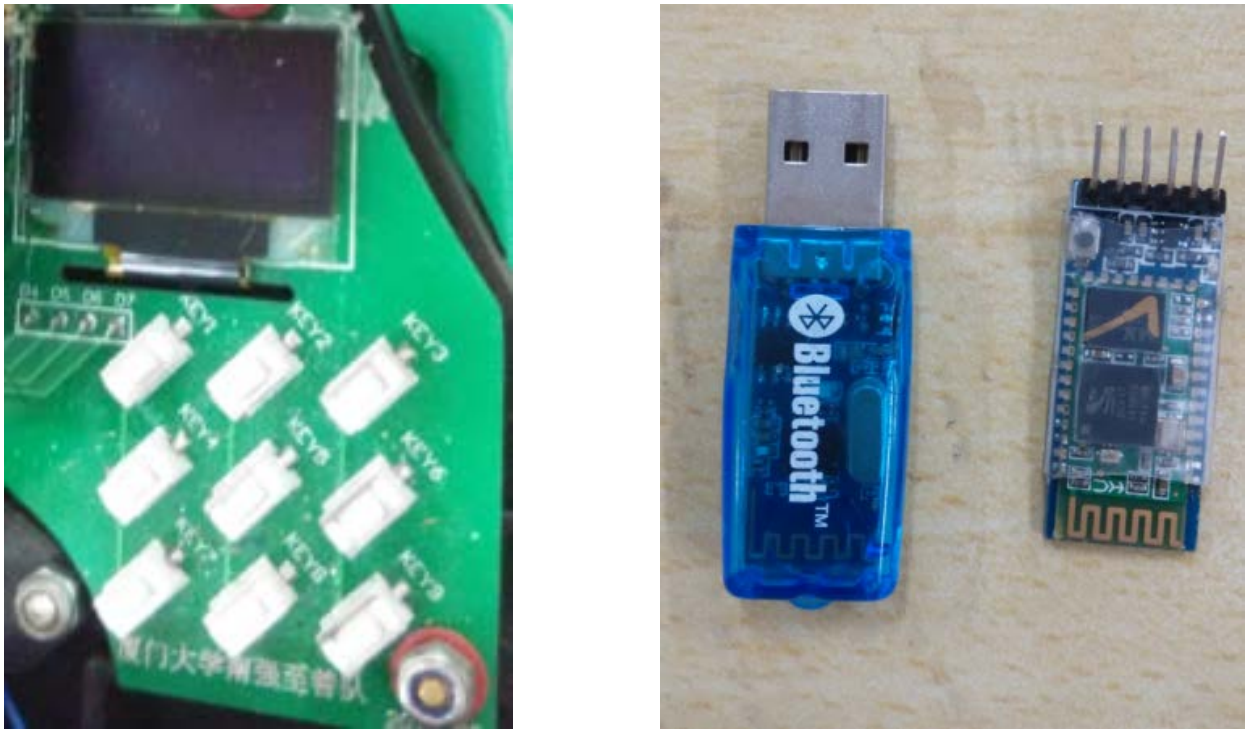
(b)

图 4.5 MOS 管驱动电路 (a)H 桥原理图

(a) 驱动电路原理图 (c)驱动电路 PCB 图

4.6 调试电路模块

在整个电路系统中，除上述几个模块外，为了方便调试，我们还加入液晶屏、键盘、蓝牙等模块，电路如图 4.8 所示。液晶屏和键盘使调车人员能对小车参数直接观察和修改；蓝牙模块则可将小车行进状况有关参数时事的返回到调车人员的电脑中，使调车人员可以时事监测小车的情况，方便调试。但是比赛时不能用蓝牙模块，故模块采用可直接插拔式，方便拿取。



(a) (b)

图 4.6 调试电路模块 (a)液晶键盘模块 (b)蓝牙模块

第五章 系统软件的设计与实现

高效稳定的软件程序是智能车平稳快速寻线的基础。我们设计的智能车使用鹰眼摄像头来获取赛道信息，图像采集处理就成了整个软件的核心内容。而单片机处理速度有限，如何高效的提取所需的赛道信息和利用所剩不多的单片机资源进行控制成为了我们研究的核心内容。进过不断的尝试和改 进，在图像信息采集方面，最终我们成功的把计算融入单片机采集图像的间隙 中，充分利用了单片机的资源。在智能车的转向和速度控制方面，我们使用了 鲁棒性很好的经典 PID 控制算法，配合使用理论计算和实际参数补偿的办法， 使在寻线中的智能车达到了稳定快速的效果。

5.1 系统软件流程图

软件设计部分主要包括：图像采集、图像处理、赛道判断、舵机打角、电机速度控制以及速度反馈处理，系统流程图如图 5.1 所示。

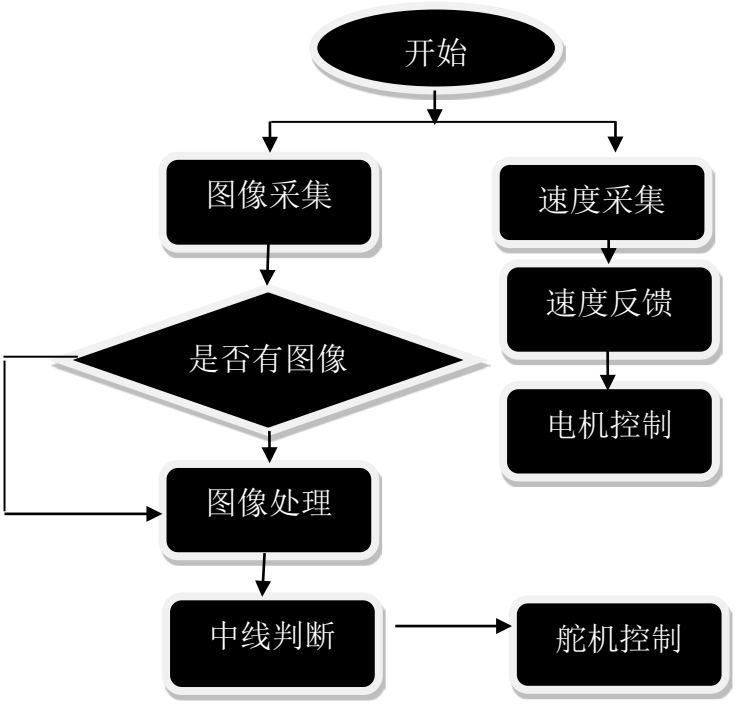


图 5.1　系统控制流程图

5.2 图像采集与处理

图像采集部分主要由 CMOS 摄像头来完成，但是由于光线，杂点，远处图像不清晰等因素干扰，图像效果会不理想。所以，在软件上进行处理，排除干扰因素，对赛道进行有效的识别。由于摄像头自身的特性，采集到的图像会产生梯形式变形和桶形式变形，使得得到的图像失真，不利于处理。因此我们还要对采集到的图像进行一系列的校正，使得到的图像能正确的反应赛道情况。对于图像信号的处理，我们的思路是分奇偶长来采集，采集行之间的间隔来进行图像处理。仔细分析可以发现，对于智能车系统而言，不需要逐行采集图像。

那么图像采集的过程中会存在一些空闲时间，我们可以利用这些时间来对图像进行处理。这种方法的优势在于：

1）图像处理嵌入到图像采集的过程中，有效解决了单片机处理图像信号能力不足的缺陷。

2）图像采集的时间可以更多，可以充分获取赛道的信息。同时也能够为智能车的控制算法预留出更多的时间。

5.2.1 赛道中心提取

今年的赛道与上一届基本相同，寻迹线也是在赛道的两边。我们采用直接逐行扫描原始图像的方法，根据设定的阈值提取赛道左右两边的黑白跳变点。由于图像的连续性，利用前面已经求出的黑线中心位置判断黑线的趋。以上一场的最近行搜线中点为中心，左右搜线，确定本行黑线的扫描范围。这样不仅节约时间，而且排除了不在宽度范围内的黑线干扰。

根据采集到的黑线宽度，判断是否采集为真正赛道，滤除噪点造成的影响；由于图像是近处大远处小，所以我们采用由近到远的方法。近同时也使用了一些去噪处理，所以得到的两条边界基本不会出错。提取完两条赛道的边界以后，我们用求平均的方法来求得赛道的中心，若是弯道，则对中心进行一定的补偿校正。这样，对小车的控制就可以参考历届成熟的控制方案。

5.2.2 赛道路径选择

根据往届比赛的经验以及历届参考资料，赛车能否以最短的时间完成比赛，与赛车的速度和路径都有着密切的关系，因此，如何使赛车以一个最合理、最高效的路径完成比赛是提高平均速度的关键。我们主要通过架高摄像头高度，从而增加视场的长度。根据分析，如果采集到的赛道图像能够比较完美的覆盖一个完整的小 S 弯道时，通过加权计算出来的中心就能位于视场中心，使小车

能以较好的路径通过 S 弯。否则小车在过 S 弯时会出现较大的抖动使路径变差，速度大大降低。

其次，由于远处的黑线少，近处的黑线多，图像分布不均匀，在车子到达一定速度后会出现过弯侧滑等现象，影响路径。所以我们对原本简单的加权进行优化。我们对于参与加权计算的图像行数及权重进行了处理，加大远处的权重在行数范围内建立一个权重常值表。但由于赛道情况不同，在计算黑线中值，如果只对采集到的黑线进行加权求中值，就会出现数据不足，根据加权的思想，对提取的黑线也进行分段加权处理。

同时根据提取到的赛道中心，我们对赛道类型进行了判断，主要分为直道、小 S、弯道、直角，障碍、坡道。赛道类型的判断基本很难做到非常的准确，所以我们结合了多种方法来进行判断,其中最主要是根据赛道中心的一个增长趋势来进行判断。

5.2.3 起跑与停车检测

由于第十一届智能车大赛使用黑线停车，因此直接根据摄像头获取图像进行识别就可以判断起跑线，通过延时检测程序可以避免停车误判

5.3 舵机转向和速度调节的 PID 控制算法

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5.4：

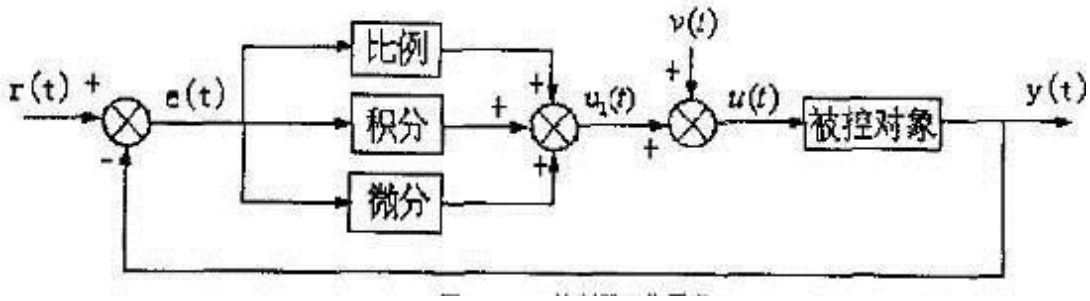


图5.4 PID控制器工作原理

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：

公式 1

$$u = k_p e + k_i \int e dt + k_d \frac{de}{dt}$$

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：

比例调节（P）作用：是按比例反应系统的偏差，系统一旦出现了偏差，比例调节立即产生调节作用用以减少偏差。比例作用大，可以加快调节，减少误 差，但是过大的比例，使系统的稳定性下降，甚至造成系统的不稳定。积分调节（I）作用：是使系统消除稳态误差，提高无差度。因为有误差，积分调节就进行，直至无差，积分调节停止，积分调节输出一常值。积分作用的强弱取

决与积分时间常数 Ti，Ti 越小，积分作用就越强。反之 Ti 大则积分 作用弱，加入积分调节可使系统稳定性下降，动态响应变慢。积分作用常与另 两种调节规律结合，组成PI调节器或PID调节器。

微分调节（D）作用：微分作用反映系统偏差信号变化率，具有预见性， 能预见偏差变化的趋势，因此能产生超前的控制作用，在偏差没有形成之前， 已被微分调节作用消除。因此，可以改善系统的动态性能。在微分时间选择合 适情况下，可以减少超调，减少调节时间。微分作用对噪声干扰有放大作用， 因此过强的加微分调节，对系统抗干扰不利。此外，微分反应的是变化率，而 当输入没有变化时，微分作用输出为零。微分作用不能单独使用，需要与另外 两种调节规律相结合，组成PD或PID控制器。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：

$$u_n = k_p\left(e_n + \frac{1}{T_i}\sum_{k=1}^{n} e_k T + T_d\frac{e_n - e_{n-1}}{T}\right)$$

其中 T 是采样时间，k_p、T_i、T_d 为三个待调参数，我们在实际代码实现算法时使用增量式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）：

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

这种算法用来控制步进电机特别方便，对直流电机也可以采用，其实如果对控制有更高的要求或者干扰因素较多，我们可以对 PID 算法做各种改进，比如用梯形法做数值积分以提高精度，将差分改成一阶数字滤波等等，在实际调车的过程中，我们确实遇到过由于自制码盘采样得到的脉冲上升下降沿不够陡，使得速度采样出现不稳定和失真，但由于这些附加处理比较耗费代码的运行时

间，出于代码效率和实际效果的比较，我们没有采用这些改进的方案另外可 以考虑加反向器来整波形得到较为理想的方波。

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，且方法简单，我们采用了这种方法，同时，我们先后实验了几种动态改变 PID 参数 的控制方法。

5.3.1 经典 PID　算法在本智能车上的应用

1）舵机控制

在小车跑动中，因为不需要考虑小车之前走过的路线，所以我们舍弃了 I 控制，将小车舵机的 PID 控制简化成 PD 控制。我们对舵机的控制采用了位置式 PID 算法，根据往届的资料和自己的测试，将图象经过算法处理后得到的赛道中心位置和对应的舵机 PID 参照角度处理成一次线性关系。另外，为了过弯更加的顺畅，我们将 Kp 值设为与偏移量成二次行数的关系。

2）速度控制

我们对速度的控制采用了增量式 PID 算法，我们在速度控制中采取的基本策略是弯道降速，直道提速。速度控制策略的好坏就决定小车取得能否优秀成绩，如何给速度是关键，什么样的赛道配合什么样的速度。比如过弯时要提前减速使赛车在入弯前达到一个安全速度以免冲出赛道，过 S 弯时要同一般弯道和直道区分开，经过实际测试我们将速度给定处理成二次曲线，这样达到的实际效果还是比较理想的。

第六章　调试环境软件说明

6.1 开发工具

软件开发工具选用的是 Embedded Workbench for ARM。Embedded Workbench for ARM 是 IARSystems 公司为 ARM 微处理器开发的一个集成开发环境(下面简称 IAR EWARM)。比较其他的 ARM 开发环境，IAR EWARM 具有入门 容易、使用方便和代码紧凑等特点。它为用户提供一个易学和具有最大量代码 继承能力的开发环境，以及对大多数和特殊目标的支持。嵌入式 IAR EmbeddedWorkbench 有效提高用户的工作效率，通过 IAR 工具，可以大大节省 软件调试时间。

6.2 上位机调试软件的设计

为了方便实时地监视小车在行进过程中的各种参数，我们用 VC++6.0 编写的一个上位机，并利用 SD 卡采集回来的图像信息传输到电脑上，用上位机处理图像，这样方便了图像处理，还大大提高了调试车子的效率。

6.3 蓝牙调试模块设计

在小车的调试过程,经常需要查看某些变量的值,通过串口线与上位机之 间进行通讯显然不现实，于是给小车设计了蓝牙通讯模块，使小车在行驶过程 种将某些重要变量通过串口发送到上位机进行观察、分析、处理。

6.4 TF 卡模块设计

由于 K60 自带 SPI 通信模块，而 TF 卡可通过 SPI 进行读写。所以，在无线调 试模块无法满足要求时，我们可以通过使用 TF 卡将需要记录的数据完整地保存 下来，以便后期处理。

第七章　车模参数

项目	参数
车模尺寸（长Ｘ宽Ｘ高）（ｍｍ）	约280*175*360
车模重量（带电池）（g）	1250g
电容重量（uF）	约500
传感器种类和个数	陀螺仪加速度传感器：1个 摄像头传感器：1个 测速传感器：1个 总计传感器：3个
除了车模原有的驱动电机、舵机外伺服机的个数（个）	0
赛道信息的检测精度（ｃｍ）	2
赛道信息检测频率(Hz)	50

第八章　结论

自去年 8 月份报名参加飞思卡尔智能车队以来，至今已过去一年时间。一年时间弹指一挥间，很长也很短。一年的做车经历，舍弃了太多东西，对于其间承 受的酸楚与辛劳来说一年确实很长；但是对于成长，对于我们一直执着追求并信奉的东西而言，一年稍纵即逝。记得去年的今天我们还是一群什么都不懂的菜鸟，一年的摸爬滚打带给我们太多弥足珍贵的东西，无论是理论知识还是实践能力，都将是伴随一身的财富。

整份报告主要包括机械、电路以及控制算法等三个方面。机械我们主要针对前轮前束，主销内倾角的调整，以及重心的降低。我们经过查找资料，一系列的计算，使用 CAD 画图，制作舵机摆臂。在电路方面，主要分为主板、驱动电路、红外检测电路三个模块来完成。每个模块的电路外形，尺寸都经过精确的测量，与机械完美结合。在程序方面，主要使用 C 语言编程，利用比赛推荐的开发工具调试程序，同时利用上位机，蓝牙等辅助调试手段，经过小组成员不断讨论、改进，终于设计出一套比较通用稳定的程序。使车子能在较短的时间内以较快的速度稳定跑完。

经过一整年的努力，我们在电路、机械、程序方面都有所进步，但是还有许多问题有待解决，例如，电机的极限、轮胎摩擦力不够等等。这些问题还有待大家一起探讨解决。

在这几个月的备战中，在场地、经费方面都的到了学校和学院的大力支持，在此特别感谢一直支持和关注智能车比赛的学校和学院领导以及各位老师。同时也感谢比赛组委会能组织这样一项很有意义的比赛。

参考文献

[1]邵贝贝. 单片机嵌入式应用的在线开发方法[M]. 北京. 清华大学出版社. 2004

[2]张军. AVR 单片机应用系统开发典型实例. 北京：中国电力出版社，2005 [4]

王晓明. 电动机的单片机控制[M]. 北京：北京航空航天大学出版社. 2002

[5]Protel DXP 电路设计与制版实用教程.2 版 李小坚[等]编著. 人民邮电出版社 2009

[3]竞赛秘书处：第十届“飞思卡尔”智能车大赛比赛规则. 2015.

[4]孙炳达：自动控制原理. 北京：机械工业出版社，2011.

[5] Tinku Acharya 等. 数字图像处理——原理与应用.清华大学出版社，2007

[6]姜明国.阿克曼原理与矩形化转向梯形设计.长春汽车研究所.2009

[7]鲍然.智能车底盘与机械结构设计.北航智能车创新基地

附录：程序源代码

主函数和 PIT：

```
void  main(void)
{
    DisableInterrupts;  //关中断

    /********** 系统初始化 **********/
    System_Init();

    /********** 显示开机图形 **********/
    Scerrn_DisWord();
    Delay(6666,6666);

    /********** 读取拨码盘的值 **********/
    BMP_GetVal();

    /********** 根据拨码盘判断 **********/
    //判断要不要读取 DFlash
    if(g_BMP1 == BMP_OFF)
    {
        MyFlash_Read();
        g_Reset_Camera_CNST = 1;              //会重新初始执行一次 SCCB 设置 CNST
和 BRIGHT
    }
    //判断要不要初始化 TF 卡
    else if(g_BMP1 == BMP_ON)
    {
        sd_init();
    }
```

```
    /********** 串口初始化 **********/
    uart_init(UART4, 115200);



    EnableInterrupts;  //开中断

    while(1)
    {
        //Key_Control();
        //Screen_Menu();
        camera_get_img();

        if(g_BMP1 == BMP_ON && g_Key_PressVal == KEY_VAL7)
        {
            SD_Gather_Camera_Picture ();
        }

        Bluebooth_Push_Data();
    }
}



/*!
 *  @brief:      定时中断 0 的中断服务函数
 */
void PIT0_IRQHandler(void)
{
    /********** 速度控制 **********/
    if(g_BMP2 == BMP_ON)            //拨码盘 2 开的时候，延时发车
```

```
{
    if(g_Motor_Flg == 1)
    {
        Timer();
        if(g_Sec >= 0)
        {
            Speed_Control();
        }
        if(g_Sec >= g_Running_Time)
            g_Motor_Flg = 0;
    }
    else
    {
        //时间清-1
        g_Ms2 = -1;
        g_Sec = -1;
        g_Min = -1;
        //关闭电机后，对积分项进行清 0
         g_Speed_Left_Error_Sum = 0;
        g_Speed_Right_Error_Sum = 0;
        //脉冲清 0
        ftm_quad_clean(FTM1);
        g_Pulse_Right_Cnt = 0;
        //坡道次数清 0
        g_Ramp_Check_Time_Cnt = 0;
        //PWM 清 0
        ftm_pwm_duty(FTM0, FTM_CH4, 0);
        ftm_pwm_duty(FTM0, FTM_CH5, 0);
        ftm_pwm_duty(FTM0, FTM_CH6, 0);
        ftm_pwm_duty(FTM0, FTM_CH7, 0);
    }
```

```
    }
    else if(g_BMP2 == BMP_OFF)              //灯塔发车
    {
        if(g_Motor_Flg == 1)        //等待的时候，用脉冲计数的那个灯塔模块
        {
            //记录
            g_Ray_Flg_Last = g_Ray_Flg;
            //获得最新一个周期的数据，递推移动前面的数据
            for(signed int i = 19; i >= 1; i--)         //g_Ray_Arr[20]是最旧的
            {
               g_Ray_Arr[i] = g_Ray_Arr[i-1];
            }
            g_Ray_Arr[0] = g_Ray_Pulse_Cnt;
            g_Ray_Pulse_Cnt = 0;                        //清中断脉冲计数
            //求和
            g_Ray_Sum = 0;
            for(signed int i = 0; i < 20; i++)
            {
                g_Ray_Sum += g_Ray_Arr[i];
            }
            //判断有无检测到红外
            if(g_Ray_Sum > g_Ray_Yes_Cnt)           //大于某个值  有     可调
                g_Ray_Flg = 1;
            else
                g_Ray_Flg = 0;
        }

        //控制
        if(g_Ray_Flg_Last == 1 && g_Ray_Flg == 0 && g_Motor_Flg == 1)       //
等待的时候从有到无
        {
```

```
            //在开启了电机的情况下，上一次有，这一次没有，灯塔关，需要发车
            g_Motor_Flg = 2;                //已经在跑的情况
        }
        else if(g_Motor_Flg == 2 )  //在跑的时候从有到无
        {
            Timer();                        //计时
            Speed_Control();                //速度控制
            //加个延时停车
            if(g_Sec >= g_Running_Time)
            {
                g_Motor_Flg = 3;
            }
            else
            {
                //检测停车部分
                if(g_Sec >= g_Ray_Check_Delay_Time)       //时间超过延时的时候，
才开始检测停车
                {
                    if(g_Braking_Flg == 0)
                    {
                        g_Ray_Stop_IO_Read = gpio_get(PTB16);
                        if(g_Ray_Stop_IO_Read == 1)                //检测到有灯
塔，就得开始计数
                        {
                            g_Ray_Stop_Cnt++;
                            if(g_Ray_Stop_Cnt      >=      g_Ray_Stop_Threshold)
//连续 g_Ray_Stop_Threshold = 3 次检测
                            {
                                g_Ray_Stop_Cnt = 0;            //计数清 0
                                g_Braking_Flg = 1;             //开始减速，等待
过计时器灯塔熄灭
```

```
                        }
                    }
                    else                                    //没有检测到灯塔的时候，把计数的值清 0
                    {
                        g_Ray_Stop_Cnt = 0;                 //计数清 0
                    }
                }
                else if(g_Braking_Flg == 1)
                {
                    g_Ray_Stop_IO_Read = gpio_get(PTB16);
                    if(g_Ray_Stop_IO_Read == 0)             //检测到灯塔熄灭
                    {
                        g_Motor_Flg = 3;
                    }
                }
            }
        }
        else if(g_Motor_Flg == 3)
        {
            //刹车
            ftm_pwm_duty(FTM0, FTM_CH4, g_Braking_PWM);
            ftm_pwm_duty(FTM0, FTM_CH5, 0);
            ftm_pwm_duty(FTM0, FTM_CH6, g_Braking_PWM);
            ftm_pwm_duty(FTM0, FTM_CH7, 0);
            //计数
            g_Braking_Cnt++;
            if(g_Braking_Cnt >= g_Braking_Delay)        //延时反转中断次数
            {
```

```
                ftm_pwm_duty(FTM0, FTM_CH4, 0);
                ftm_pwm_duty(FTM0, FTM_CH5, 0);
                ftm_pwm_duty(FTM0, FTM_CH6, 0);
                ftm_pwm_duty(FTM0, FTM_CH7, 0);
                g_Motor_Flg = 0;
                g_Braking_Cnt = 0;
                g_Braking_Flg = 0;
            }
        }
        else
        {
            g_Ms2 = 0;
            g_Sec = 0;
            g_Min = 0;
            ftm_pwm_duty(FTM0, FTM_CH4, 0);
            ftm_pwm_duty(FTM0, FTM_CH5, 0);
            ftm_pwm_duty(FTM0, FTM_CH6, 0);
            ftm_pwm_duty(FTM0, FTM_CH7, 0);
        }
    }

    /********** 坡道延时 **********/
    if(g_Ramp_Flg == 1)             //延时清坡道标志，防止姿态模块没有清掉
    {
        g_Ramp_Time_Cnt++;          //大于 g_Ramp_Time_Threshole 后就会清标志
        //随便限幅，怕数据溢出
        if(g_Ramp_Time_Cnt > 10000)      g_Ramp_Time_Cnt = 10000;
    }
    if(g_Ramp_End_To_Start_Delay_Cnt > 0)           //坡道结束后，延时一段时间
在开启检测
    {
```

```
        g_Ramp_End_To_Start_Delay_Cnt++;
        if(g_Ramp_End_To_Start_Delay_Cnt > g_Ramp_Check_Delta)     //下坡后延时
2s（给定 2s）再次检测  g_Ramp_Check_Delta = 1000
            g_Ramp_End_To_Start_Delay_Cnt = 0;
    }

    /********** 读读累加的脉冲值 **********/
    //g_Speed_Left_Cnt = -ftm_quad_get(FTM1);     //读值
    //ftm_quad_clean(FTM1);                       //清 0

    //g_Speed_Right_Cnt = g_Pulse_Right_Cnt;
    //g_Pulse_Right_Cnt = 0;

    /********** 舵机赋值 **********/
    //限幅
    if(g_Steer_PWM < g_Steer_Min)          g_Steer_PWM = g_Steer_Min;
    if(g_Steer_PWM > g_Steer_Max)          g_Steer_PWM = g_Steer_Max;
    //输出 PWM
    ftm_pwm_duty(FTM2,FTM_CH1,g_Steer_PWM);

    /********** 按键双击 **********/
    if(++g_Key7Double_Cnt > 110)         g_Key7Double_Cnt = 0;
    if(++g_Key8Double_Cnt > 110)         g_Key8Double_Cnt = 0;

    /********** 电池电压 **********/
    g_Battery_ADC = adc_once(ADC1_SE14, ADC_16bit);

    /********** 求角度 **********/
    Angle_Clac();

    /********** 亮灯延时 **********/
```

```
    if(g_Shine_light_Cnt > 0)
    {
        PTB0_OUT = 0;
        g_Shine_light_Cnt++;
        if(g_Shine_light_Cnt > 50)
        {
            g_Shine_light_Cnt = 0;
            PTB0_OUT = 1;
        }
    }

    PIT_Flag_Clear(PIT0); //清中断标志位
}
```

摄像头处理：

```
void Camera_Dispose(unsigned char imgbuff[])
{
    //摄像头标志位初始化
    Camera_Flg_Init();

    /*********************************************************************/
    /*最近行检测和处理                                                   */
    /*********************************************************************/
    g_Left[g_Behind_Line] = Camera_Scan_Left_Boundary(imgbuff, g_Behind_Line,
g_Mid_Last0);
    g_Right[g_Behind_Line] = Camera_Scan_Right_Boundary(imgbuff, g_Behind_Line,
g_Mid_Last0);
    if( (g_Left[g_Behind_Line] != -1) && (g_Right[g_Behind_Line] != -1) )
```

```
        g_Mid[g_Behind_Line]    =    (signed    int)((g_Left[g_Behind_Line]    +
g_Right[g_Behind_Line]) / 2);
    else
        g_Mid[g_Behind_Line] = g_Mid_Last0;

    /**********最近行检测**********/
    Camera_Check_First(imgbuff);

    /**********最近行处理**********/
    Camera_Dispose_First(imgbuff);

    /**********剩余处理**********/
    Camera_Dispose_Last(imgbuff);

    /**********交接弯偏差偏移**********/
    g_Alter_Bend_Error_Offset        =        Camera_Dispose_Alter_Bend(imgbuff,
g_Behind_Line);

    /**********检测坡道**********/
    if(g_Sec >= g_Ramp_Check_Delay_Time)         //判断是否开启检测坡道
    {
        Check_Ramp_All();
    }
    else
    {
        g_Ramp_Flg = 0;
    }
}

/*!
 *  @brief:     摄像头最近行检测
```

```
 *  @param:     imgbuff[]       采集回来的数据数组
 *  @return:    void
 *  @note:      void
 *  @since:     version 1: 05_25 ---> Done;
                version 2: 06_19 ---> 修改变量名;
 */
void Camera_Check_First(unsigned char imgbuff[])
{
    signed int tmp_left=0,tmp_right=0;

    //检测最近 30 行有没有单线
    g_Check_Into_Single_Flg =
        Camera_Check_Straight_Into_Single_Line(imgbuff,          g_Behind_Line,
g_Mid_Last0);
    ///**/" 初 始 30 行 搜 单 线 %d 单 线 行 %d\n\r",
g_Check_Into_Single_Flg,g_Check_Into_Single_Line - FRONT_LINE);
    if(g_Check_Into_Single_Line != g_Behind_Line)
    {
        //找到最近行的左右边线，利用上一场的中点搜最近行的左右边界
        tmp_left    =    Camera_Scan_Left_Boundary(imgbuff,    g_Behind_Line,
g_Mid_Last0);
        tmp_right   =   Camera_Scan_Right_Boundary(imgbuff,    g_Behind_Line,
g_Mid_Last0);
    }
    if(g_Check_Into_Single_Flg == -1)//初始行无单线
    {
        if( (tmp_left != -1) && (tmp_right != -1) )
        {
            if(tmp_right - tmp_left < IMG_BOTTOM_WIDTH - 40)//可能为胶带可能为
障碍
            {
```

```
                g_Bottom_Narrow = 1; //在向上搜最高行后再判断是胶带还是障碍
            }
            if(tmp_right - tmp_left > IMG_BOTTOM_WIDTH + 40)
            {
                g_Bottom_Broad = 1;//需要清掉，以找到出单线的情况
            }
            g_Left[g_Behind_Line] = tmp_left;
            g_Right[g_Behind_Line] = tmp_right;
        }
        else //初始搜线中点为黑点，说明为出胶带,需要向下补线不管初始行
        {
            g_Check_Bend_Out_Tape_Flg = 2;
        }

        //全部列竖直向上搜黑点
        if(g_Check_Bend_Out_Tape_Flg == -1)
        {
            Camera_Scan_All_Vertical(imgbuff, g_Vertical_Black, g_Behind_Line,
IMG_BLACK, tmp_left, tmp_right);
            //更新赋值到 g_Upright[]加减速可能用，后面 g_Vertical_Black 可能会被
更替
            for(signed int i = tmp_left; i <= tmp_right; i++)
            {
                g_Upright[i] = g_Vertical_Black[i];
            }
            //初始行过窄，判断为出胶带还是障碍内（分为初始行在胶带前或后）
            if( g_Bottom_Narrow == 1 )
            {
                ///**/"初始行过窄\t\t%d\n\r", tmp_right - tmp_left);

                //判断赛道趋势,或者哪边需要修线
```

```
                Camera_Check_Tendency(g_Vertical_Black, g_Behind_Line, tmp_left,
tmp_right);

                ///**/"行%d 赛道趋势-->%d\n\r", g_Tendency_Line - FRONT_LINE,
g_Tendency);

                ///**/"向左-->%d,向右-->%d,全丢-->%d\n\r", 1, 2, 3);

                ///**/"左丢-->%d,右丢-->%d\n\r", 4, 5);

                g_Check_Bend_Out_Tape_Flg                                    =
Camera_Check_Bend_In_Tape(g_Vertical_Black,g_Behind_Line,tmp_left,tmp_right);
                if(g_Check_Bend_Out_Tape_Flg == 1)
                {
                    ///**/"  初  始  行  在  胶  带  内  \t\t%d\n\r",
g_Check_Bend_Out_Tape_Flg);
                }
                if(g_Check_Bend_Out_Tape_Flg == -1)
                {
                    ///**/"  判  断  是  障  碍  还  是  胶  带  \t%d\n\r",
g_Check_Bend_Out_Tape_Flg);

    Camera_Check_Out_Barrier_Or_Tape(imgbuff,g_Vertical_Black,g_Behind_Line,tm
p_left,tmp_right);
                }
            }
            else  //判断情况：赛道趋势，入胶带，单线，障碍，斜入十字，
            {
                //判断赛道趋势,或者哪边需要修线
                ///**/"判断赛道趋势-->%d\n\r", g_Tendency);
```

```
                Camera_Check_Tendency(g_Vertical_Black, g_Behind_Line, tmp_left,
tmp_right);

                if(g_Tendency < 3)
                {
                    //判断障碍
                   Camera_Check_Block(imgbuff, g_Vertical_Black, g_Behind_Line,
tmp_left, tmp_right);
                    if(g_Check_Barrier_Flg == -1)
                    {
                        //判断单线
                        Camera_Check_Single(imgbuff,          g_Vertical_Black,
g_Behind_Line, tmp_left, tmp_right);
                        if( g_Check_Into_Single_Flg == -1 )
                        {
                            //判断入胶带
                            Camera_Check_Tape(    imgbuff,    g_Vertical_Black,
g_Behind_Line, tmp_left, tmp_right);
                        }
                    }
                }
                else//初始行有丢线
                {
                    if(g_Tendency != 3)
                    {
                        //判断单线
                        Camera_Check_Single(imgbuff,          g_Vertical_Black,
g_Behind_Line, tmp_left, tmp_right);
                        if( g_Check_Into_Single_Flg == -1 )
                        {
                            //判断斜入十字，斜出十字
```

```
                        Camera_Check_Bend_Crossroad(imgbuff,
g_Vertical_Black, g_Behind_Line, tmp_left, tmp_right);
                        if(     (g_Bend_Crossroad_Flg     ==     -1)     &&
(g_Straight_Crossroad_Out_Flg == -1) )
                        {
                            //判断入胶带
                            Camera_Check_Tape(  imgbuff,  g_Vertical_Black,
g_Behind_Line, tmp_left, tmp_right);
                        }
                    }
                }
                else
                {
                    //判断出十字
                    Camera_Check_Straight_Crossroad_Out(imgbuff,
g_Vertical_Black, g_Behind_Line);
                    if(g_Straight_Crossroad_Out_Flg == -1)
                    {
                        //判断入胶带
                        Camera_Check_Tape(    imgbuff,    g_Vertical_Black,
g_Behind_Line, tmp_left, tmp_right);
                    }
                }
                /*

                //判断入胶带
                Camera_Check_Tape( imgbuff, g_Vertical_Black, g_Behind_Line,
tmp_left, tmp_right);
                if(     (g_Check_Bend_Into_Tape_Flg     ==     -1)     &&
(g_Check_Straight_Into_Tape_Flg == -1) && (g_Big_Bend_Flg == -1) )
                {
```

```
                        if(g_Tendency == 3)//初始行全白
                        {
                            //判断出十字
                            Camera_Check_Straight_Crossroad_Out(imgbuff,
g_Vertical_Black, g_Behind_Line);
                        }
                        else //初始行非全白
                        {
                            //判断单线
                            Camera_Check_Single(imgbuff,      g_Vertical_Black,
g_Behind_Line, tmp_left, tmp_right);
                            if( g_Check_Into_Single_Flg == -1 )
                            {
                                //判断斜入十字，斜出十字
                                Camera_Check_Bend_Crossroad(imgbuff,
g_Vertical_Black, g_Behind_Line, tmp_left, tmp_right);
                            }
                        }
                    }*/
                }
            }
        }
        if(g_Check_Bend_Out_Tape_Flg != -1)  //初始行在胶带内,搜线中点为黑点
        {
            ///**/"初始行出胶带-->%d\n\r", g_Check_Bend_Out_Tape_Flg);
            Camera_Check_Bend_Out_Tape(imgbuff,
g_Behind_Line,tmp_left,tmp_right);
        }
    }
}
```

```
/*!
 *  @brief:     摄像头最近行处理
 *  @param:     imgbuff[]       采集回来的数据数组
 *  @return:    void
 *  @note:      void
 * @since:      version 1: 05_25 ---> Start;
                version 2: 06_24 ---> Done;
 */
void Camera_Dispose_First(unsigned char imgbuff[])
{
    g_Scan_Loop = g_Behind_Line;
    if(g_Check_Into_Single_Flg != -1)//单线
    {
        if(g_Check_Into_Single_Flg == 0)//初始三十行入单线
        {
            g_Scan_Loop     =     Camera_Dispose_Straight_Into_Single(imgbuff,
g_Behind_Line);
            g_Out_Single_Line = g_Scan_Loop;
        }
        else
        {
            g_Scan_Loop = Camera_Dispose_Into_Single(imgbuff, g_Behind_Line);
            g_Out_Single_Line = g_Scan_Loop;
        }
    }
    if(g_Check_Out_Barrier_Flg != -1)//初始行在障碍内
    {
        g_Scan_Loop = Camera_Dispose_Out_Barrier(imgbuff, g_Behind_Line);
        if(g_First_Black_Line != FRONT_LINE - 1)//处理结束的直接跳出
        {
            return;
```

```
        }
    }
    if(g_Check_Barrier_Flg != -1)//初始行看到障碍
    {
        g_Scan_Loop = Camera_Dispose_Block(imgbuff, g_Behind_Line);
        if(g_First_Black_Line != FRONT_LINE - 1)//处理结束的直接跳出
        {
            return;
        }
    }
    if(g_Check_Bend_Out_Tape_Flg != -1)//初始行在胶带内,搜线中点为黑点,初始行在胶带外
    {
        g_Scan_Loop = Camera_Dispose_Out_Tape(imgbuff, g_Behind_Line);
    }
    if( (g_Check_Straight_Into_Tape_Flg != -1) || (g_Check_Bend_Into_Tape_Flg != -1) )//入胶带
    {
        g_Scan_Loop = Camera_Dispose_Tape(imgbuff, g_Behind_Line);
    }
    if( g_Straight_Crossroad_Out_Flg != -1 )//出十字
    {
        g_Scan_Loop = Camera_Dispose_Crossroad_Out(imgbuff, g_Behind_Line);
        g_Bottom_Broad = -1;
    }
    if( g_Bend_Crossroad_Flg != -1)//十字
    {
        g_Scan_Loop = Camera_Dispose_Bend_Crossroad(imgbuff, g_Behind_Line);
    }
    if( g_Big_Bend_Flg != -1 )//大弯
    {
```

```
        g_Scan_Loop = Camera_Dispose_Big_Bend(imgbuff, g_Behind_Line);
    }
    if(g_Bottom_Broad != -1)//初始行为出单线，需要处理，不知是否需要结束
    {
        ///**/"出单线处理-->%d\n\r", g_Bottom_Broad);
        g_Scan_Loop = Camera_Dispose_Bottom_Out_Single(imgbuff, g_Behind_Line);
        if(g_First_Black_Line != FRONT_LINE - 1)//处理结束的直接跳出
        {
            return;
        }
        g_Bottom_Broad = -1;
    }
    if(g_Check_S_Into_Tape_Flg != -1)//小弯接胶带处理
    {
        ///**/"小弯接胶带处理-->%d\n\r", g_Check_S_Into_Tape_Flg);
        g_Scan_Loop = Camera_Dispose_S_Into_Tape(imgbuff, g_Behind_Line);
    }
    if(g_Check_Into_Single_Flg != -1)//出单线
    {
        g_Scan_Loop = Camera_Dispose_Out_Single(imgbuff, g_Behind_Line);
        if( (g_Left_Lost_Line != -1) || (g_Right_Lost_Line != -1) )//处理弯道
        {
            ///**/"单线接弯-->%d\n\r", g_Scan_Loop - FRONT_LINE);
            g_Scan_Loop = Camera_Dispose_Bend(imgbuff, g_Behind_Line);
            ///**/"单线接弯-->%d\n\r", g_Scan_Loop - FRONT_LINE);
        }
        if(g_First_Black_Line != FRONT_LINE - 1)//处理结束的直接跳出
        {
            return;
        }
    }
```

```
    if( g_Bend_Into_S_Flg != -1 )//小S
    {
        g_Scan_Loop = Camera_Dispose_Bend_Into_S(imgbuff, g_Behind_Line);
        if(g_First_Black_Line != FRONT_LINE - 1)//处理结束的直接跳出
        {
            return;
        }
    }
    if(g_First_Black_Line != FRONT_LINE - 1)//处理结束的直接跳出
    {
        return;
    }
}



/*!
 *  @brief:     摄像头最后的处理
 *  @param:     imgbuff[]       采集回来的数据数组
 *  @return:    void
 *  @note:      void
 *  @since:     version 1: 05_25 ---> Done;
                version 2: 06_25 ---> Start;
 */
void Camera_Dispose_Last(unsigned char imgbuff[])//双胶带，直道，小弯，斜入十字，直入十字，斜入障碍
{
    //后面的正常搜线
    signed int tmp_cross  = -1;
    signed int tmp_left = -1,tmp_right = -1,tmp_mid = -1;
    signed int tmp_cross_value, reget_line = -1;
```

```
    if(g_First_Black_Line != FRONT_LINE - 1)//处理结束的直接跳出
    {
        return;
    }
    if( (g_Straight_Crossroad_Out_Flg != -1) || (g_Bend_Crossroad_Flg != -1) )
    {
        g_Check_Cross_Times++;
    }
    //清标志
    g_Left_Lost_Line = -1;
    g_Right_Lost_Line = -1;
    g_Left_Reget_Line = -1;
    g_Right_Reget_Line = -1;
    tmp_mid = (g_Left[g_Scan_Loop] + g_Right[g_Scan_Loop]) >> 1;
    tmp_cross = g_Scan_Loop;
    for(; tmp_cross >= g_Front_Line; tmp_cross--)//处理小弯，斜入十字，直入十字，
剩余胶带
    {
        if(g_First_Black_Line != FRONT_LINE - 1)//处理结束的直接跳出
        {
            return;
        }
        if(reget_line != -1)
        {
            if( reget_line <= tmp_cross )
            {
                continue;
            }
            else
            {
                tmp_mid = g_Mid[reget_line+1];
```

```
                if(tmp_mid > 319)
                    tmp_mid = 319;
                else if(tmp_mid < 0)
                    tmp_mid = 0;
                reget_line = -1;
            }
        }
        if(tmp_mid > 319)
            tmp_mid = 319;
        else if(tmp_mid < 0)
            tmp_mid = 0;
        tmp_left = Camera_Scan_Left_Boundary(imgbuff, tmp_cross, tmp_mid);
        tmp_right = Camera_Scan_Right_Boundary(imgbuff, tmp_cross, tmp_mid);
        //找丢线，竖直判断是否十字，弯道
        //内跳是否是障碍
        if( (tmp_left == -1) || (tmp_right == -1) )//找截止行，判断是否是胶带，
障碍
        {
            ///**/"剩余次数%d\n\r", 1 - g_Check_Cross_Times);

            ///**/"左丢线行%d\n\r", g_Left_Lost_Line - FRONT_LINE);

            ///**/"右丢线行%d\n\r", g_Right_Lost_Line - FRONT_LINE);
            g_Left_Into_Tape_Line = -1;
            g_Left_Out_Tape_Line = -1;
            g_Right_Into_Tape_Line = -1;
            g_Right_Out_Tape_Line = -1;
            if( (tmp_cross > g_Front_Line + 10) && (g_Left_Lost_Line == -1) &&
(g_Right_Lost_Line == -1) &&
                (g_Check_Into_Single_Flg == -1) && (g_Check_Barrier_Flg == -1) )
            {
```

```
                g_First_Black_Line     =     Camera_Dispose_Final_Tape(imgbuff,
tmp_cross);
                break;
            }
            g_First_Black_Line = tmp_cross - 1;
            break;
        }
        tmp_mid = (tmp_left + tmp_right)>>1;
        g_Left[tmp_cross] = tmp_left;
        g_Right[tmp_cross] = tmp_right;
        g_Mid[tmp_cross] = tmp_mid;
        if( (tmp_cross < g_Scan_Loop - 1) && (tmp_cross >= g_Front_Line + 20) )//
判断丢线
        {
            if( (g_Left_Lost_Line == -1) && (g_Right_Lost_Line == -1) )//最多找
两次十字
            {
                if( (tmp_cross >= 35 + g_Front_Line) && (g_Bottom_Broad ==
-1) )//08_11
                {
                    if(tmp_left - g_Left[tmp_cross + 1] > 10)//08_11
                    {
                        if(   Camera_Jump_In_Check_Block(imgbuff,   tmp_cross,
tmp_left, 1) == 1 )
                        {
                            g_Scan_Loop     =     Camera_Dispose_Block(imgbuff,
g_Behind_Line);
                            return;
                        }
                    }
                    if(g_Right[tmp_cross + 1] - tmp_right > 10)//08_11
```

```
                    {
                        if(   Camera_Jump_In_Check_Block(imgbuff,   tmp_cross,
tmp_right, 2) == 1 )
                        {
                            g_Scan_Loop     =     Camera_Dispose_Block(imgbuff,
g_Behind_Line);
                            return;
                        }
                    }
                }
                if( ((g_Left[tmp_cross + 2] - tmp_left > 10) || (tmp_left == 0))
&&
                    ((tmp_right - g_Right[tmp_cross + 2] > 10) || (tmp_right ==
319)) && (g_Check_Cross_Times < 1))//直入十字
                {
                    g_Left_Lost_Line = tmp_cross + 2;
                    g_Right_Lost_Line = tmp_cross + 2;
                    if( g_Left_Lost_Line < g_Front_Line + 20)
                    {
                        Camera_Dispose_Mid_Rate(imgbuff,g_Left_Lost_Line);
                        ///**/"丢线行过远斜率补线%d\n\r", g_Left_Lost_Line -
FRONT_LINE);
                        return;
                    }
                    ///**/"直入十字丢线行%d\n\r", tmp_cross + 2 - FRONT_LINE);

                    g_Check_Cross_Times++;
                    tmp_cross_value = g_Right_Lost_Line;
                    tmp_cross_value                                          =
Camera_Dispose_Straight_Into_Cross(imgbuff, g_Right_Lost_Line);
```

```
                    if(     (tmp_cross_value     <     g_Right_Lost_Line)     &&
(g_First_Black_Line == FRONT_LINE - 1) )
                    {
                        tmp_cross = tmp_cross_value;
                        tmp_mid = g_Mid[tmp_cross_value];
                        tmp_left = g_Left[tmp_cross];
                        tmp_right = g_Right[tmp_cross];
                        g_Left_Lost_Line = -1;
                        g_Left_Reget_Line = -1;
                        g_Right_Lost_Line = -1;
                        g_Right_Reget_Line = -1;
                    }
                }
                if( ((g_Left[tmp_cross + 2] - tmp_left > 10) || (tmp_left == 0))
&&
                    ((tmp_right - g_Right[tmp_cross + 2] <= 10) && (tmp_right !=
319)) )//左丢线
                {
                    if(g_Left[tmp_cross + 2] > g_Left[tmp_cross + 1])
                        g_Left_Lost_Line = tmp_cross + 2;
                    else
                        g_Left_Lost_Line = tmp_cross + 1;
                    if(g_Left_Lost_Line > g_Behind_Line - 10)
                    {
                        reget_line        =        Camera_Lost_verify(imgbuff,
tmp_mid,tmp_cross + 2);
                        if(reget_line != -1)
                        {
                            g_Left_Lost_Line = -1;
                        }
                    }
```

```
                    if(g_Left_Lost_Line != -1)
                    {
                        ///**/"左丢线行%d\n\r", tmp_cross + 2 - FRONT_LINE);
                        if( g_Left_Lost_Line < g_Front_Line + 20)
                        {
                            Camera_Dispose_Mid_Rate(imgbuff,g_Left_Lost_Line);
                            ///**/"丢线行过远斜率补线%d\n\r", g_Left_Lost_Line
- FRONT_LINE);
                            return;
                        }
                        if( (g_Check_Cross_Times == -1) && (g_Bend_Crossroad_Flg
== -1) &&
                            (g_Straight_Crossroad_Out_Flg == -1) )//未处理过十
字，判断斜入十字，交接弯，大弯
                        {
                            if( ( ((g_Left[g_Left_Lost_Line] < g_Tendency_Column
- 1) && (g_Tendency_Line - g_Left_Lost_Line > 2))
                                || (g_Tendency_Column == g_Left[g_Behind_Line] +
1) ) && ((g_Tendency == 1)||(g_Tendency == 3))
                                && (g_Left_Lost_Line < g_Behind_Line - 5)  &&
(g_Cross_Fuzz_Flg == -1) )
                            {
                                ///**/" 判 断 左 弯 入 十 字 或 弯 %d\n\r",
g_Left_Lost_Line - FRONT_LINE);
                                tmp_cross_value = g_Left_Lost_Line;
                                tmp_cross_value                                =
Camera_Dispose_Bend_Cross_And_Curve(imgbuff, g_Left_Lost_Line);
                                if(  (tmp_cross_value  <  g_Left_Lost_Line)  &&
(g_First_Black_Line == FRONT_LINE - 1) )
                                {
                                    tmp_cross = tmp_cross_value;
```

```
                    tmp_mid = g_Mid[tmp_cross_value];
                    tmp_left = g_Left[tmp_cross];
                    tmp_right = g_Right[tmp_cross];
                    g_Left_Lost_Line = -1;
                    g_Left_Reget_Line = -1;
                    g_Right_Lost_Line = -1;
                    g_Right_Reget_Line = -1;
                }
            }
            else
            {
                ///**/" 判 断 左 入 十 字 或 弯 %d\n\r",
g_Left_Lost_Line - FRONT_LINE);
                tmp_cross_value = g_Left_Lost_Line;
                tmp_cross_value                                =
Camera_Dispose_Almost_Straight_Into_Cross(imgbuff, g_Left_Lost_Line);
                if( (tmp_cross_value < g_Left_Lost_Line) &&
(g_First_Black_Line == FRONT_LINE - 1) )
                {
                    tmp_cross = tmp_cross_value;
                    tmp_mid = g_Mid[tmp_cross_value];
                    tmp_left = g_Left[tmp_cross];
                    tmp_right = g_Right[tmp_cross];
                    g_Left_Lost_Line = -1;
                    g_Left_Reget_Line = -1;
                    g_Right_Lost_Line = -1;
                    g_Right_Reget_Line = -1;
                }
            }
        }
```

```
                    else if(g_Check_Cross_Times == 0)//处理过一次十字，判断
十字，大弯
                    {
                        ///**/"判断左入十字或弯%d\n\r", g_Left_Lost_Line -
FRONT_LINE);

                        tmp_cross_value = g_Left_Lost_Line;
                        tmp_cross_value                                    =
Camera_Dispose_Almost_Straight_Into_Cross(imgbuff, g_Left_Lost_Line);
                        if(   (tmp_cross_value   <   g_Left_Lost_Line)   &&
(g_First_Black_Line == FRONT_LINE - 1) )
                        {
                            tmp_cross = tmp_cross_value;
                            tmp_mid = g_Mid[tmp_cross_value];
                            tmp_left = g_Left[tmp_cross];
                            tmp_right = g_Right[tmp_cross];
                            g_Left_Lost_Line = -1;
                            g_Left_Reget_Line = -1;
                            g_Right_Lost_Line = -1;
                            g_Right_Reget_Line = -1;
                        }
                    }
                }
            }
            if( ((g_Left[tmp_cross + 2] - tmp_left <= 10) && (tmp_left != 0))
&&
                ((tmp_right - g_Right[tmp_cross + 2] > 10) || (tmp_right ==
319)) )//右丢线
            {
                ///**/"右丢线行%d\n\r", tmp_cross + 2 - FRONT_LINE);
```

```
                    if(g_Right[tmp_cross + 1] > g_Right[tmp_cross + 2])
                        g_Right_Lost_Line = tmp_cross + 2;
                    else
                        g_Right_Lost_Line = tmp_cross + 1;
                    if(g_Right_Lost_Line > g_Behind_Line - 10)
                    {
                        reget_line        =        Camera_Lost_verify(imgbuff,
tmp_mid,tmp_cross + 2);
                        if(reget_line != -1)
                        {
                            g_Right_Lost_Line = -1;
                        }
                    }
                    if(g_Right_Lost_Line != -1)
                    {
                        if( g_Right_Lost_Line < g_Front_Line + 20)
                        {

Camera_Dispose_Mid_Rate(imgbuff,g_Right_Lost_Line);
                            ///**/"丢线行过远斜率补线%d\n\r", g_Right_Lost_Line
- FRONT_LINE);
                            return;
                        }
                        ///**/"  右  跳  变  列  %d  初  始  边  %d\n\r",
g_Tendency_Column,g_Right[g_Behind_Line]);

                        if( (g_Check_Cross_Times == -1) && (g_Bend_Crossroad_Flg
== -1) &&
                            (g_Straight_Crossroad_Out_Flg == -1) )//未处理过十
字，判断斜入十字，交接弯，大弯
                        {
```

```
                        if(     (     ((g_Right[g_Right_Lost_Line]     >
g_Tendency_Column + 1) && (g_Tendency_Line - g_Right_Lost_Line > 2))
                            || (g_Tendency_Column == g_Right[g_Behind_Line]
- 1) ) && ((g_Tendency == 2) || (g_Tendency == 4))
                            && (g_Right_Lost_Line < g_Behind_Line - 5)  &&
(g_Cross_Fuzz_Flg == -1) )
                        {
                            ///**/" 判 断 右 弯 入 十 字 或 弯 %d\n\r",
g_Right_Lost_Line - FRONT_LINE);
                            tmp_cross_value = g_Right_Lost_Line;
                            tmp_cross_value                                =
Camera_Dispose_Bend_Cross_And_Curve(imgbuff, g_Right_Lost_Line);
                            if( (tmp_cross_value < g_Right_Lost_Line) &&
(g_First_Black_Line == FRONT_LINE - 1) )
                            {
                                tmp_cross = tmp_cross_value;
                                tmp_mid = g_Mid[tmp_cross_value];
                                tmp_left = g_Left[tmp_cross];
                                tmp_right = g_Right[tmp_cross];
                                g_Left_Lost_Line = -1;
                                g_Left_Reget_Line = -1;
                                g_Right_Lost_Line = -1;
                                g_Right_Reget_Line = -1;
                            }
                        }
                        else
                        {
                            ///**/" 判 断 右 入 十 字 或 弯 %d\n\r",
g_Right_Lost_Line - FRONT_LINE);
                            tmp_cross_value = g_Right_Lost_Line;
```

```
                                tmp_cross_value                                 =
Camera_Dispose_Almost_Straight_Into_Cross(imgbuff, g_Right_Lost_Line);
                                if( (tmp_cross_value < g_Right_Lost_Line) &&
(g_First_Black_Line == FRONT_LINE - 1) )
                                {
                                    tmp_cross = tmp_cross_value;
                                    tmp_mid = g_Mid[tmp_cross_value];
                                    tmp_left = g_Left[tmp_cross];
                                    tmp_right = g_Right[tmp_cross];
                                    g_Left_Lost_Line = -1;
                                    g_Left_Reget_Line = -1;
                                    g_Right_Lost_Line = -1;
                                    g_Right_Reget_Line = -1;
                                }
                            }
                        }
                        else if(g_Check_Cross_Times == 0)//处理过一次十字，判断
十字，大弯
                        {
                            ///**/"判断右入十字或弯%d\n\r", g_Right_Lost_Line -
FRONT_LINE);
                            tmp_cross_value = g_Right_Lost_Line;
                            tmp_cross_value                                     =
Camera_Dispose_Almost_Straight_Into_Cross(imgbuff, g_Right_Lost_Line);
                            if( (tmp_cross_value < g_Right_Lost_Line) &&
(g_First_Black_Line == FRONT_LINE - 1) )
                            {
                                tmp_cross = tmp_cross_value;
                                tmp_mid = g_Mid[tmp_cross_value];
                                tmp_left = g_Left[tmp_cross];
                                tmp_right = g_Right[tmp_cross];
```

```
                                g_Left_Lost_Line = -1;
                                g_Left_Reget_Line = -1;
                                g_Right_Lost_Line = -1;
                                g_Right_Reget_Line = -1;
                            }
                        }
                    }
                }
            }
        }
        if(g_First_Black_Line != FRONT_LINE - 1)//处理结束的直接跳出
        {
            return;
        }
    }
}



/*!
 *  @brief:     摄像头标志位初始化
 *  @param:     void
 *  @return:    void
 *  @note:      void
 *  @since:     version 1: 05_25 ---> Done;
 */
void Camera_Flg_Init(void)
{
    //Scan
    g_First_Black_Line = FRONT_LINE - 1;        //截止行，默认为最远行在往前一
行
```

```
    g_Scan_Line = BEHIND_LINE;                    //正确搜线的行数
    g_Scan_Loop = BEHIND_LINE;                    //搜线循环变量

    Camera_Clear_All_Flg();
    for(signed int i = g_Behind_Line; i >= g_Front_Line; i--)
    {
        g_Left[i] = -1;
        g_Right[i] = -1;
        //g_Mid 不变
    }
    for (signed int j = IMG_LEFT_LIMIT; j <= IMG_RIGHT_LIMIT; j++)
    {
        g_Vertical_Black[j] = g_Behind_Line;
        g_Vertical_Black2[j] = g_Behind_Line;
    }

    //更新搜线中点，视情况，不一定都需要更新
    if(g_Mid[g_Behind_Line] < IMG_LEFT_LIMIT)
        g_Mid_Last0 = IMG_LEFT_LIMIT;
    else if(g_Mid[g_Behind_Line] > IMG_RIGHT_LIMIT)
        g_Mid_Last0 = IMG_RIGHT_LIMIT;
    else
        g_Mid_Last0 = g_Mid[g_Behind_Line];
}

void Camera_Clear_All_Flg(void)
{
    //Flg
    g_Tendency = -1;
    g_Tendency_Line = -1;
    g_Tendency_Column = -1;
```

```
g_Check_Out_Barrier_Flg = -1;
g_Check_Straight_Into_Tape_Flg = -1;
g_Check_Bend_Into_Tape_Flg = -1;
g_Check_Bend_Out_Tape_Flg = -1;
g_Check_Barrier_Flg = -1;
g_Bottom_Narrow = -1;
g_Bottom_Broad = -1;
g_Check_Into_Single_Flg = -1;
g_Big_Bend_Flg = -1;
g_Straight_Crossroad_Out_Flg = -1;
g_Bend_Crossroad_Flg = -1;
g_Bend_Into_S_Flg = -1;
g_Check_S_Into_Tape_Flg = -1;
g_Check_Out_Curve_Single_Flg = -1;
g_single_side = -1;
g_Single_Same_Curve = -1;
g_Check_Cross_Times = -1;
g_Out_Single_False_Flg = -1;                //单线找回错误行
g_Cross_Fuzz_Flg = -1;                   //十字易误判
g_Unreliable_Flg = -1;              //判断另一边也不可信
g_Curve_Tape_Into_Cross = -1;               //弯入胶带接十字
g_Single_Fauls_Get = 0;
g_Check_Cross_Times = -1;                     //检测十字次数
//Line
g_Check_Into_Single_Line = -1;
g_Check_Straight_Crossroad_Out_Line = -1;
g_Block_Narrow_Line = -1;
g_Block_Up_Line = -1;
g_Block_Down_Line = -1;
g_Big_Bend_Line = -1;
g_Bend_Endline = -1;
```

```
    g_Left_Into_Tape_Line = -1;
    g_Right_Into_Tape_Line = -1;
    g_Left_Out_Tape_Line = -1;
    g_Right_Out_Tape_Line = -1;
    g_Left_Lost_Line = -1;
    g_Left_Reget_Line = -1;
    g_Right_Lost_Line = -1;
    g_Right_Reget_Line = -1;
    g_Out_Single_Line = -1;
    g_Bend_Into_S_Line = -1;
    g_Check_S_Into_Tape_Line = -1;
    g_Potential_Left_Lost_Line = -1;
    g_Potential_Right_Lost_Line = -1;
    g_Check_False_Single_Line = -1;
    g_Check_False_Single_Column = -1
}
```