第九届“飞思卡尔”杯全国大学生

智能汽车竞赛

技 术 报 告

学校：东南大学

队伍名称：Armageddon

参赛队员：王天宜

马文涛

带队老师：谈英姿 孙琳

关于技术报告和研究论文使用授权的说明

本人完全了解第九届“飞思卡尔”杯全国大学生智能汽车竞赛有关保留、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

“飞思卡尔”杯全国大学生智能汽车竞赛是一项以汽车电子为背景，涵盖控制、模式识别、传感技术、电子、电气、计算机、机械等多个学科的科技创意性比赛，由教育部高等学校自动化教学指导分委员会主办、飞思卡尔半导体公司协办。现在，该赛事已发展成为教育部面向全国大学生的大型专业竞赛之一，对培养学生的综合工程能力、创新精神、实践动手能力及团结协作精神均具有良好的促进作用。

Armageddon智能车主要由3个线性CCD传感器构成的道路检测子系统，驱动电机以及机械传动齿轮构成的动力子系统，连杆机构以及转向伺服电机构成的转向子系统，速度检测系统以及以MK60为中心的电路子系统构成。结合软件部分的PID控制理论完成了智能车自主循线，避障，过坡道和人字路的功能。在调试过程中主要利用TFT屏幕模块，结合MATLAB编写的上位机处理SD卡读取的赛道数据，使调试过程更加顺利。

本文主要讨论了Armageddon智能车的设计方案和原理。文章将从系统设计，机械结构，硬件电路，软件算法以及调试方法等几个方面全面介绍智能车的制作及调试过程。

引用的相关文献将在文章结尾列举。

关键词 Freescale 智能车 3线性CCD PID SD卡

第二章 系统设计

2.1 系统方案介绍

本参赛小队的智能车系统采用 K60 作为核心控制单元，由安装在智能车上的三个线性 CCD 传感器接收从比赛赛道反馈回来的信号并传到核心控制单元单片机中，由智能车 MCU 接收并处理从传感器反馈的信号后，PWM 模块产生相应的 PWM 波形，通过输出不同占空比控制智能车的转向舵机与直流电机，以实现控制智能车按照规定赛道行驶。为了使智能车能够更加快速稳定的行驶，智能车 MCU 必须把对赛道

路径的判断、转向舵机的角度控制以及对直流电机的控制紧密的联系在一起。不论是某一路采集信号的误判还是单片机对转向伺服电机控制的失当，都会引起智能车在行驶过程中产生抖动甚至偏离赛道。所以，对智能车系统设计必须稳定科学合理。

2.2 智能车系统模块设计

2.2.1 线性 CCD 模块

利用双线性 CCD 模块对比赛赛道信息进行采集并反馈给中央处理单元，由中央处理单元对采集数据进行处理。利用中间的线性 CCD 模块采集坡道部分的赛道信息，进行特殊处理。

2.2.2 电源模块

为各个电路模块提供稳定的直流电源，保证各个模块稳定工作。

2.2.3 速度检测模块

对智能车的实时速度检测并反馈，以实现对智能车速度的闭环控制，以便调整智能车在直到和弯道上的行驶速度，使其能够平稳快速的跑完全程。

2.2.4 舵机驱动模块

对智能车上的舵机进行驱动，以达到快速准确的控制智能车的行驶方向。

2.2.5 电机驱动模块

对直流电机进行驱动，控制赛车的速度。

2.2.6 陀螺仪模块

检测坡道，从而对赛车的速度的进行控制。

第三章 机械设计

根据大赛的要求，我们选择 B 型车模作为智能车制作的基础车模。在严格遵守比赛

对车模所做的要求的前提下，按照说明书所说的步骤对车模进行安装，并根据实际条件进行了一定的改装。

3.1 车后轮结构调整

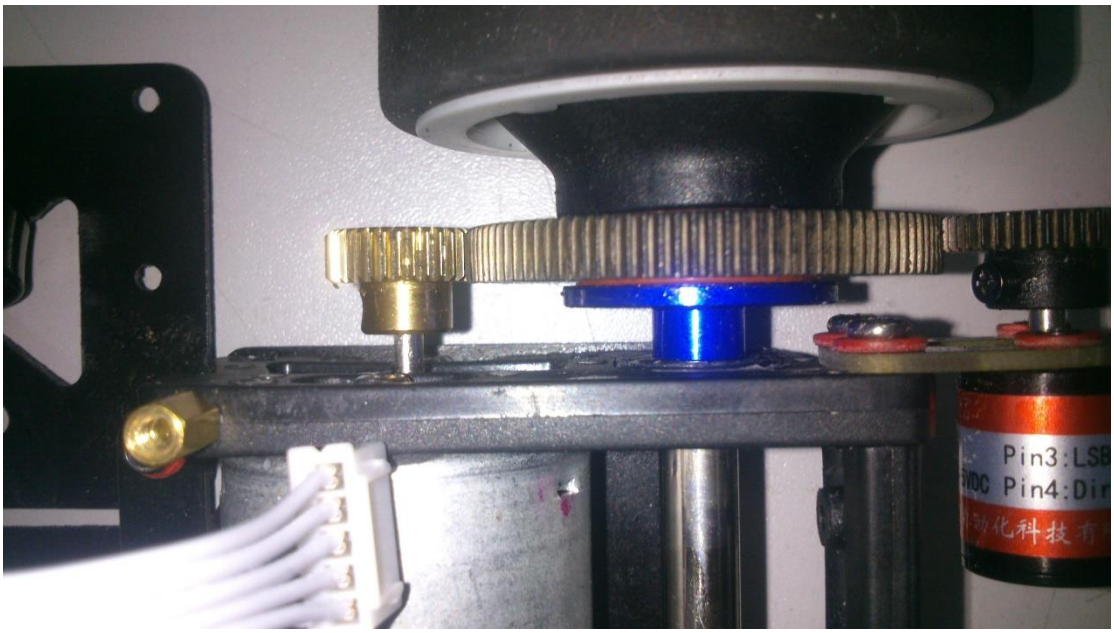
3.1.1 差速器

1 差速器配件包

差速器位置应该处于传动轴与左右半轴的交汇点，从变速箱输出的动力在这里被分配到左右两个半轴。汽车在直线行驶时左右两个驱动轮的转速是相同的，但在转弯过时两边车轮行驶的距离不是等长的，因此车轮的转速肯定也会不同。差速器的作用就在于允许左右两边的驱动轮以不同的转速运行。

直线行驶时的特点是左右两边驱动轮的阻力大致相同。从发动机输出的动力首先传递到差速器壳体上使差速器壳体开始转动。接下来要把动力从壳体传递到左右半轴上，我们可以理解为两边的半轴齿轮互相在“较劲”，由于两边车轮阻力相同，因此二者谁也掰不过对方，因此差速器壳体内的行星齿轮跟着壳体公转同时不会产生自转，两个行星齿轮咬合着两个半轴齿轮以相同的速度转动，这样汽车就可以直线行驶了！

假设车辆现在向左转，左侧驱动轮行驶的距离短，相对来说会产生更大的阻力。差速器壳体通过齿轮和输出轴相连，在传动轴转速不变情况下差速器壳体的转速也不变，因此左侧半轴齿轮会比差速器壳体转得慢，这就相当于行星齿轮带动左侧半轴会更费力，这时行星齿轮就会产生自传，把更多的扭矩传递到右侧半轴齿轮上，由于行星齿轮的公转外加自身的自传，导致右侧半轴齿轮会在差速器壳体转速的基础上增速，这样以来右车轮就比左车轮转得快，从而使车辆实现顺滑的转弯。



2 差速器

3.1.2 轴承

后轮轴承的过孔进行倒置处理，有利于降低重心，提高智能车稳定性。

3 轴承

3.2 车前轮结构调整

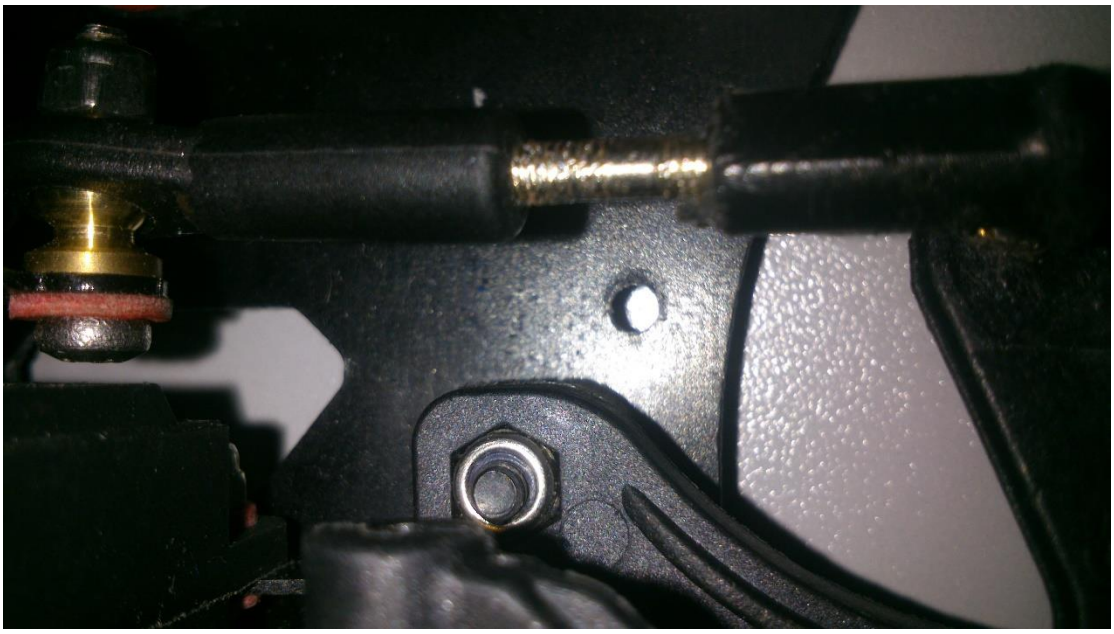
3.2.1 前轮外倾和前轮前束

主销后倾是指在汽车的纵向平面内（汽车的侧面）有一个向后的倾角，即主销轴线与地面垂直线在汽车纵向平面内的夹角。采用主销后倾角的原因是由于汽车在车轮偏转后会产生一回正力矩，纠正车轮的偏转。

主销前倾是指在汽车的横向平面内向内倾斜的一个角度，即主销轴线与地面垂直线在汽车横向断面内的夹角。主销前倾也有使车轮自动回正的作用。

前轮外倾角是指通过车轮中心的汽车横向平面与车轮平面的交线与地面垂线之间的夹角。当轮胎呈“八”字形张开时称为“负外倾”，而呈现“V”字形张开时称为“正外倾”。前轮外倾角可以减少转向阻力，使汽车转向轻便。模型车提供了编号 B-210 前摆臂拉杆配件包配件用来调节前轮外倾的角度。

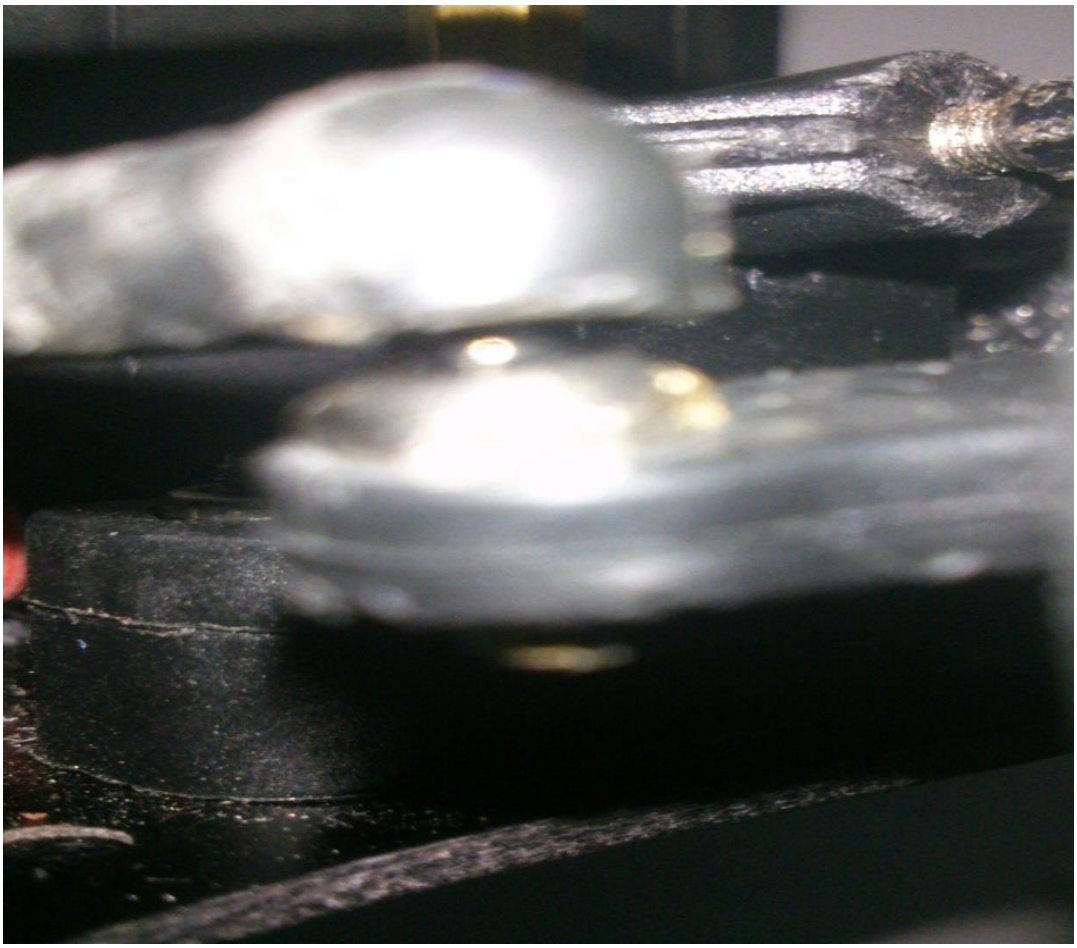
当车轮有了外倾角后，在滚动时就类似与圆锥滚动，从而导致两侧车轮向外滚开。由于转向横拉杆和车桥的约束使车轮不可能向外滚开，车轮将在地面上出现边滚边向内滑移的现象，从而增加了轮胎的磨损。在安装车轮时候，为了消除车轮外倾带来的这种不良后果，可以使汽车两前轮的中心面不平行，并使两轮前边缘距离小于后面的边缘距离，两者之差称为“前轮前束”。模型车是由舵机带动左右横拉杆实现转向的。主销在垂直方向的位置确定后，改变左右横拉杆的长度即可改变前束的大小。



4 横拉杆

3.2.2 垫座

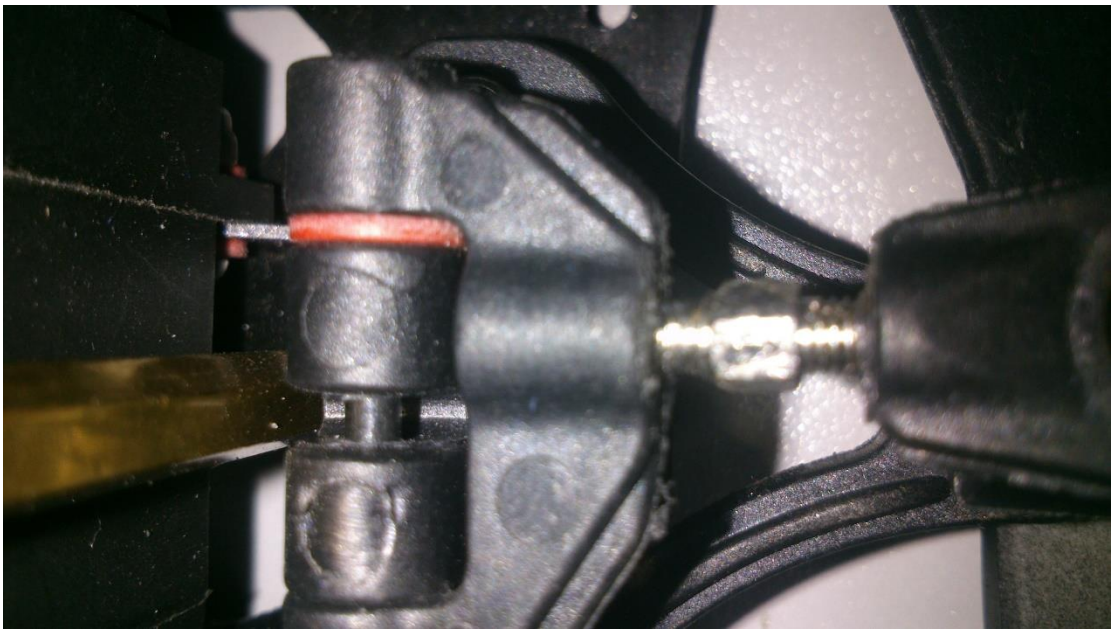
前轮支架与底盘之间添加垫座，有利于降低重心。



5 垫座

3.2.3 垫片

加垫片消除空程

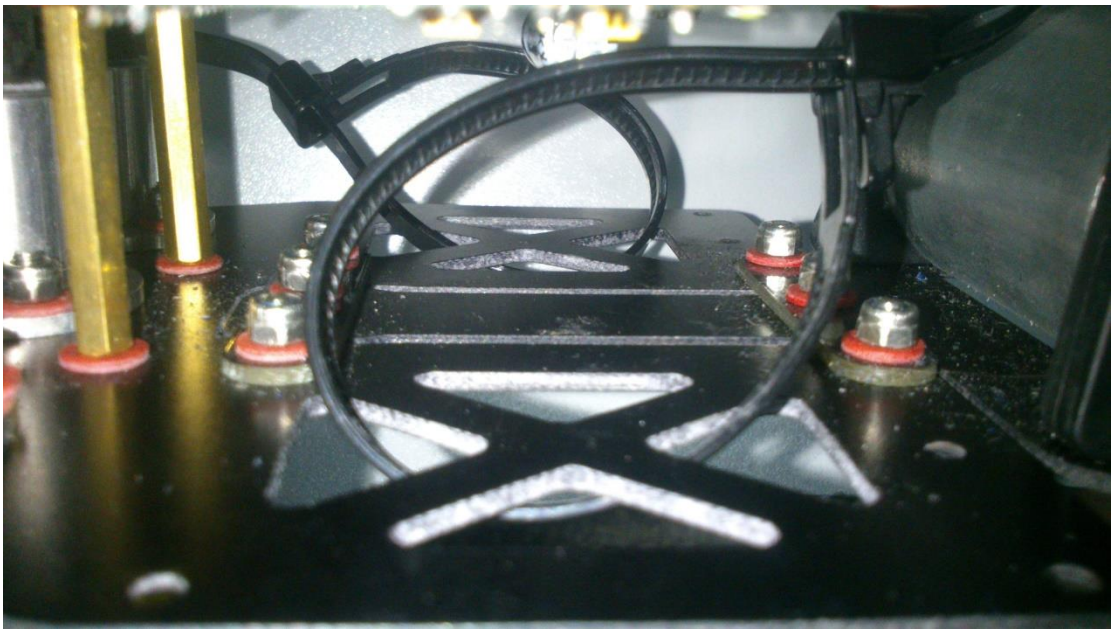


6 红色平垫片

3.3 其他机械结构调整

3.3.1 电池固定

为降低智能车重心，Armageddon 没有使用电池座，而是将电池用绑带直接固定于底盘之上，虽然这样的固定方式稳定性有所欠缺，但对于智能车来说仍能保证必要的稳定。



7 电池绑带

3.3.2 防撞垫

调试过程中，赛车冲出赛道的情况时有发生，一旦发生强烈撞击，将会损坏赛车的机械结构或部件。尤其是以较高速度失控，很可能对舵机造成损伤。为保护赛车安全，我们量身制作了防撞泡沫垫，实用且不失美观。

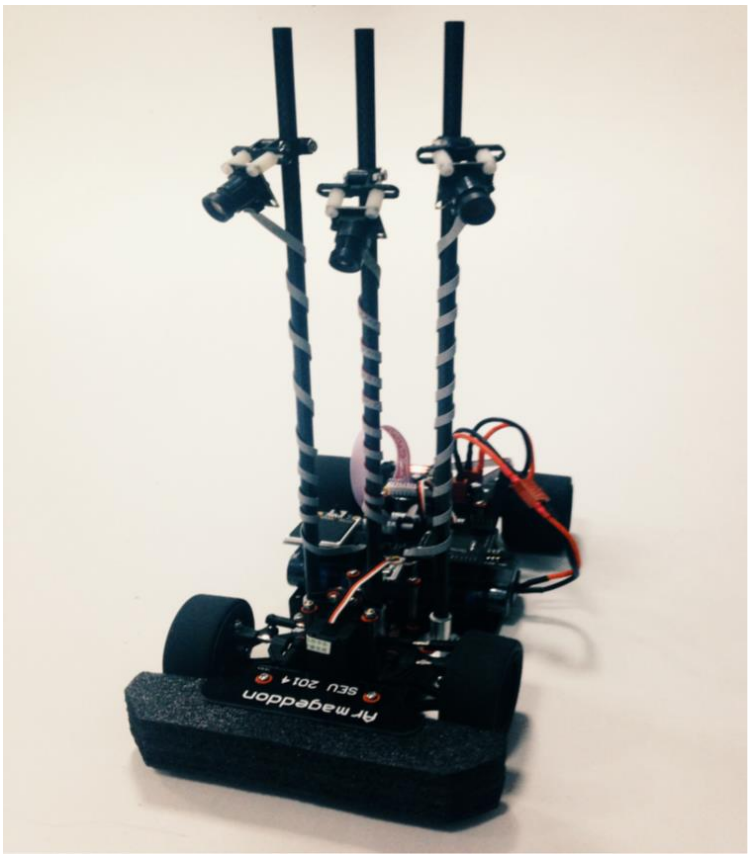
8 防撞垫

3.3.3 其他

其他机械结构包括轴承、转向杆等，要酌情调整使之松紧适当、转动自如；在关键的连接部位可以在完成调节后使用强力胶固定。各种螺丝一定要上紧，并经常检查，防止行驶过程中零件松动造成的损坏。

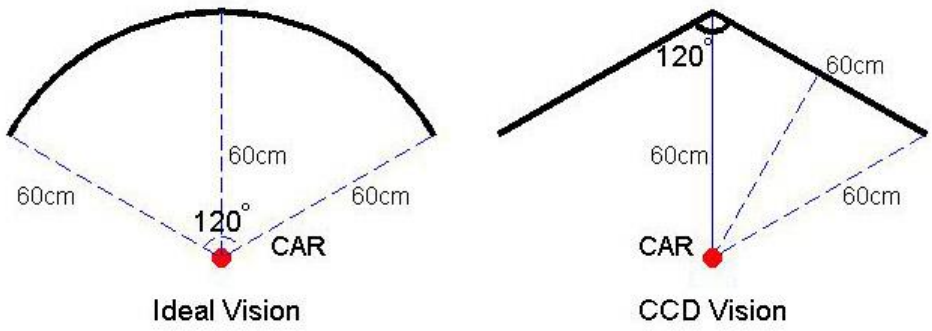
3.4 传感器的设计安装

针对第九届飞思卡尔智能车光电组的比赛，我们对于一种新型的线性 CCD 的组合方案进行了探索，确定了一种双线阵 CCD 排布的方式。从而获得了更加广阔的视野范围和较大的前瞻。基于此排布方式的算法得到了稳定的表现。



9 车模全身照

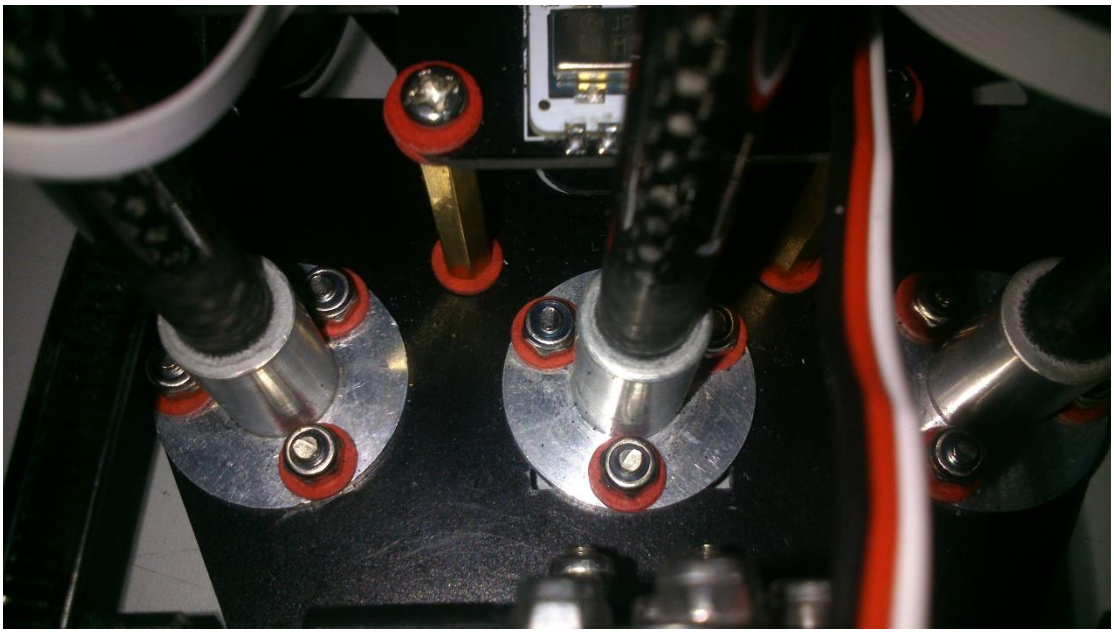
我们认为理想的前瞻范围应类似人眼，视野范围应当是距离镜头等距离的一段圆弧，此时视野范围比较开阔。为了接近这种效果，我们用两个 CCD 拼出两条相交直线。设计思路的理想排布比对效果如图示。



10 理想视野和 CCD 实际视野

本着简约稳定的设计原则，为本智能车量身定制了线性 CCD 碳素支撑杆和固定座，以及固定于底盘上的铝合金底座，效果图如图所示。

11 碳素支撑杆



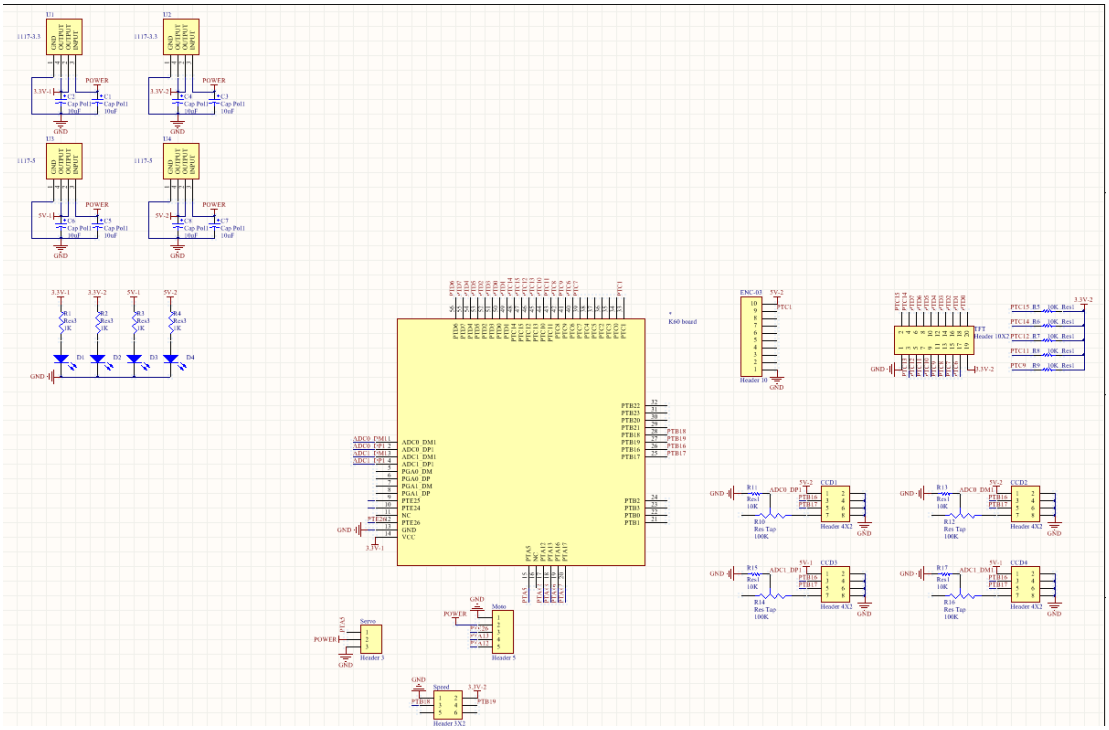
12 铝合金底座

第四章 硬件电路设计

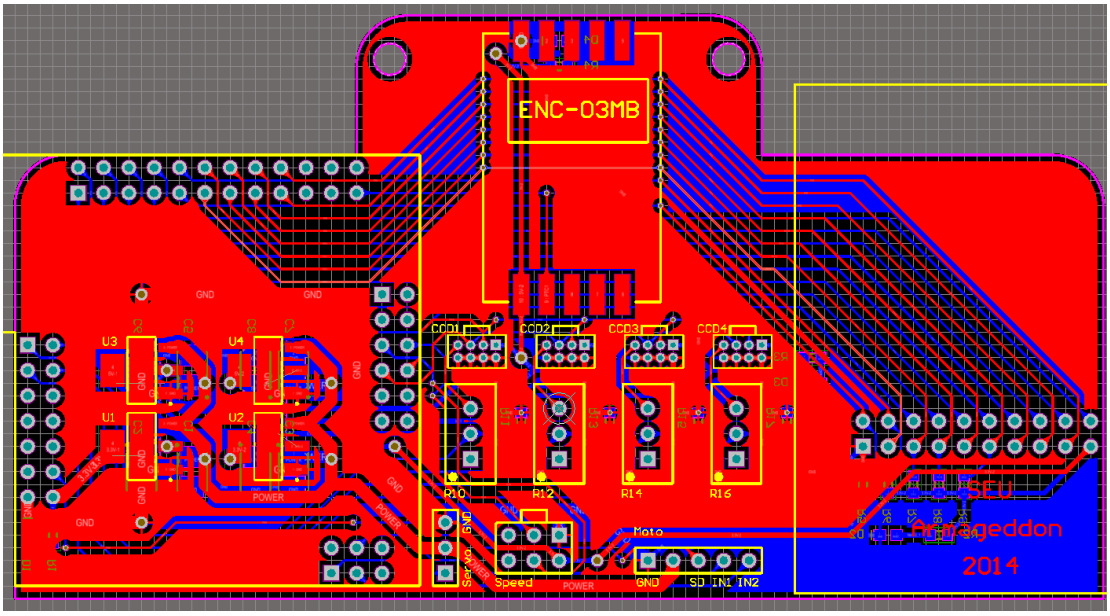
硬件电路部分遵循稳定高效简洁的原则，三个 CCD 支撑碳素杆占用了车中的一片横向区域，我们决定在有限的空间内，考虑到尽量减轻整车重量，降低车体重心位置，所以使电路设计尽量简洁，尽量减少元器件使用数量，缩小电路板面积，使电路部分重量轻，易于安装。

4.1 单片机最小系统

控制系统主芯片为飞思卡尔公司的 MK60DN512VLL10，为了使整个控制电路板的体积最小化，我们自行设计了尺寸较核心只引出所需的管脚，图为最小系统板原理图及 PCB 图。



13 最小系统原理图

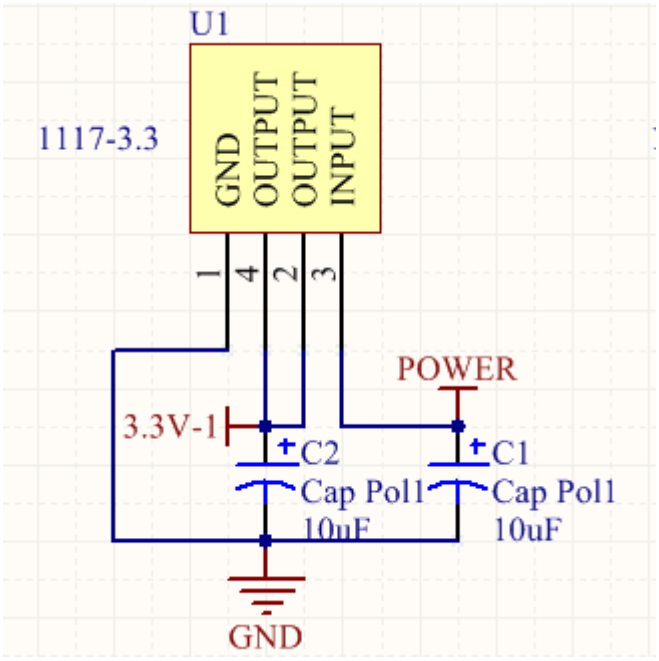


14 最小系统 PCB

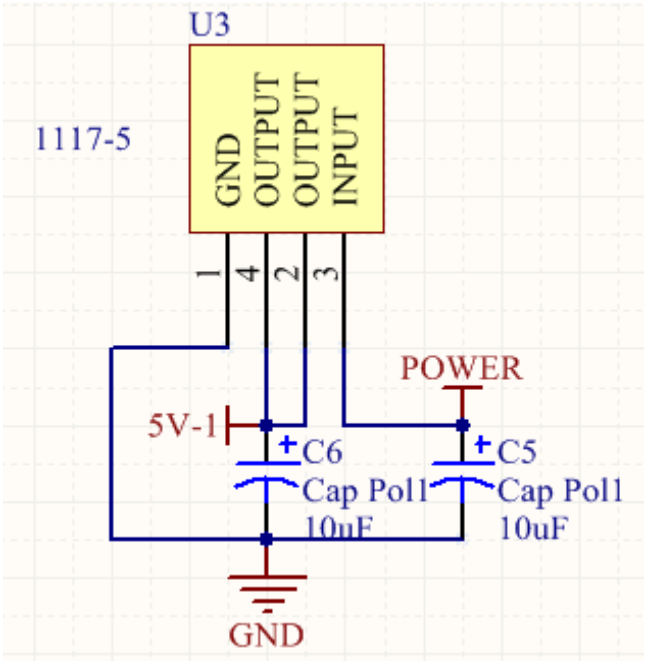
4.2 电源模块

电源模块为小车系统的其他各模块提供所需要的电源。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。本系统全部硬件电路的电源由 7.2V、2A/h 的可充电镍镉电池提供。由于电路中的不同电路模块所需要的工作电压和电流容量不相同，因此电源模块应该包含多个稳压电路，将充电电池电压转换成各个模块所需要的电压。

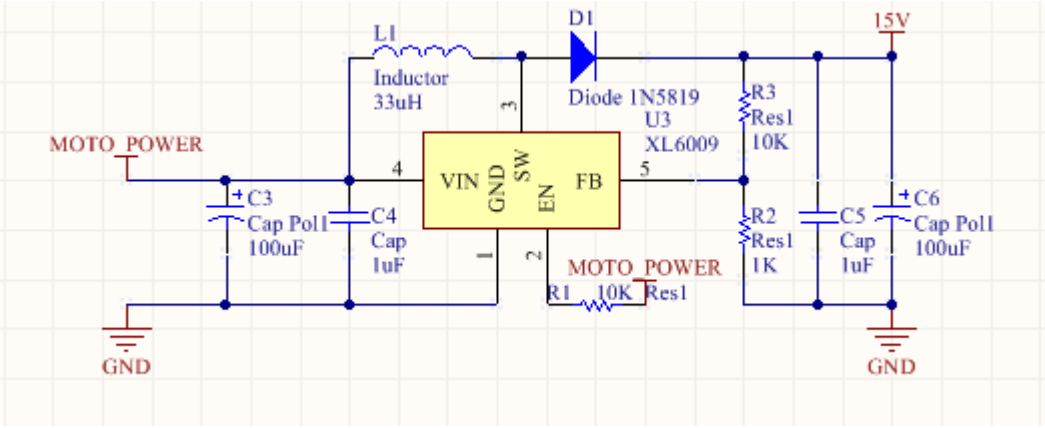


15 3.3V 供电电路



16 5.0V 供电电路

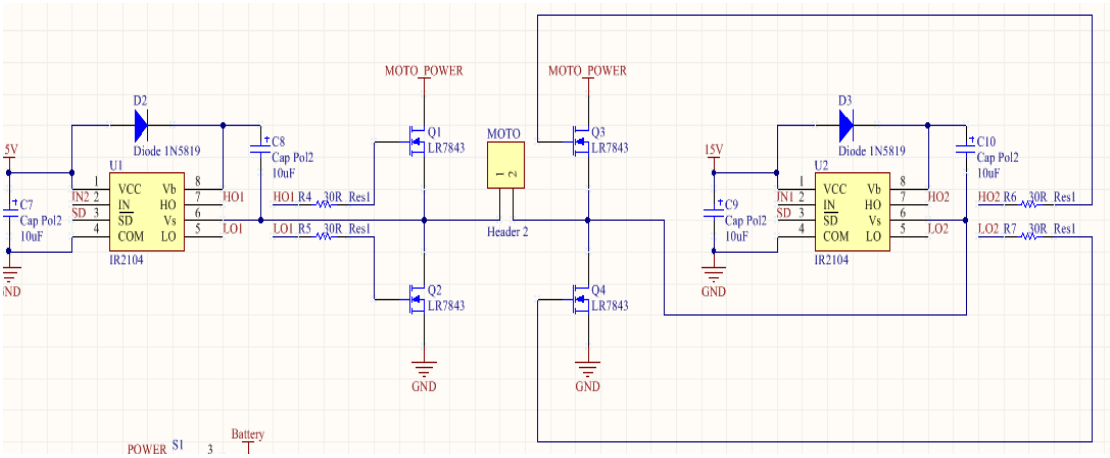
给驱动 MOS 管的 IR2104 供电的电路设计原理图



17 半桥驱动芯片供电电路

4.3 电机驱动模块

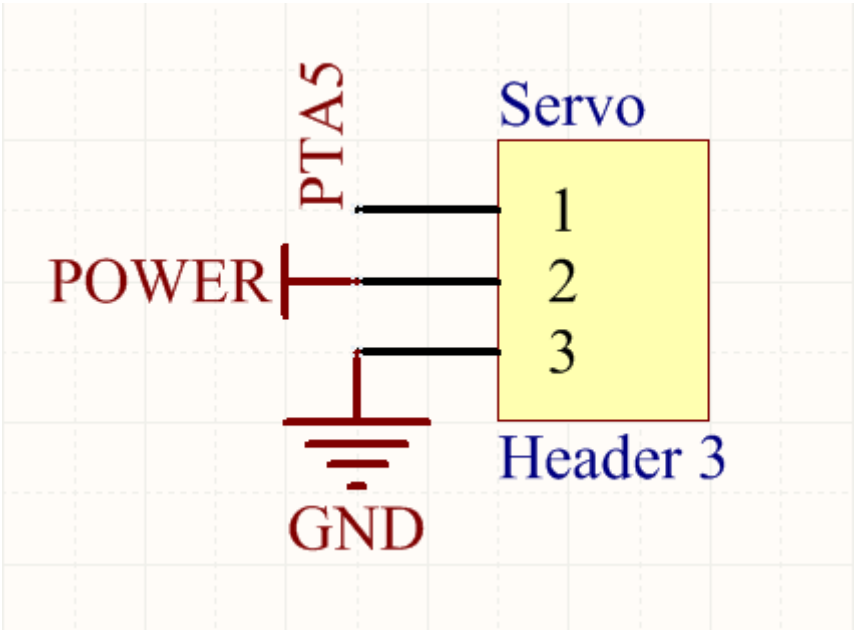
Armageddon 的电机驱动采用半桥电路，使用 IR2104 驱动两片 LR7843，上桥臂为电机提供动力，下桥臂进行制动。IR2104 使用单路 PWM 输入 IN，高端输出 HO 与 IN 同步，低端输出 LO 与 HO 反相，且 IR2104 内部有死区控制电路，有效地避免了两个管子同时导通损坏供电线路及电池。通过实际使用，我们认为对于光电组智能车来说，制动采用半桥驱动的能耗制动已经足够，且其制动能力随车速的提高而加强，相比于反接制动来说更加可靠，完全避免了刹停、刹反的可能性。同时 MOS 管控制电流小，减少 CPU 发热，而且内阻小、驱动电流大，有利于快速加速和制动。驱动电路原理图如图



18 电机驱动电路

4.4 舵机供电电路

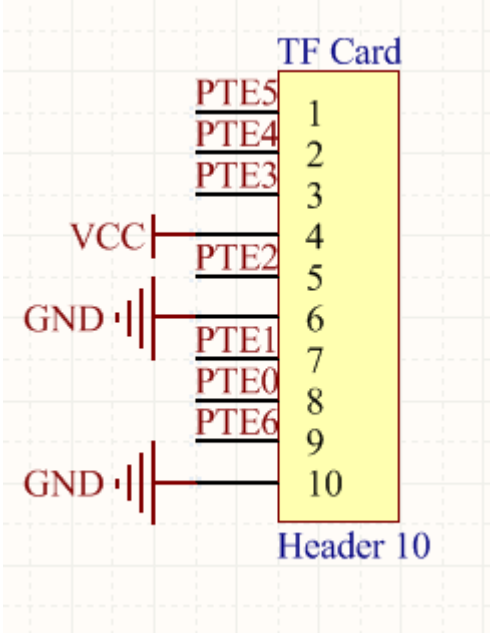
Armageddon 使用电池电压为舵机供电，以期得到更快的响应速度。



19 舵机供电电路

4.5 SD 卡电路

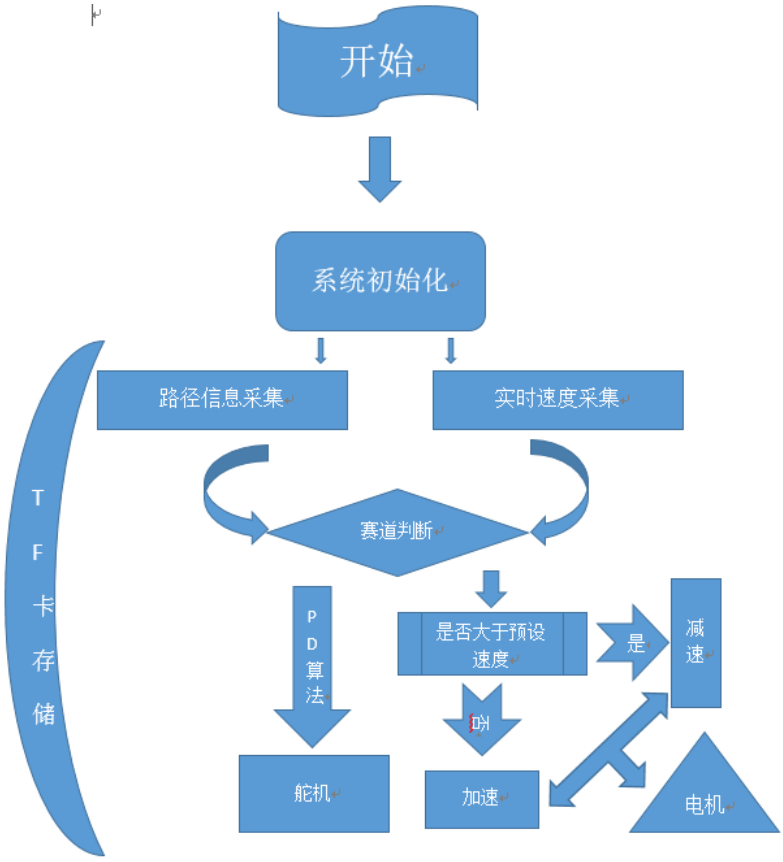
为方便记录行驶数据，Armageddon 今年搭载了 SD 卡模块，将行驶数据保存在 TF 卡中方便对算法进行分析。主板 TF 卡插座如图。



20 SD 卡电路

第五章 软件系统设计

本次比赛要求小车能够自主识别路线，按照设定的轨迹行走，实现避障，过人字路，坡道的功能，并在不冲出跑道的前提下追求速度。因此，系统软件的设计基本原则是：首先考虑小车的稳定性，在此基础上完成各项功能要求，尽量提高小车的速度。整个软件系统采用模块化的程序结构。从结构上看，系统程序主要包括一个主体循环程序、转向控制程序、中断服务程序、图像处理算法程序、速度控制算法程序以及其他一些如写SD卡等控制程序组成。



21 程序流程图

5.1 控制算法

控制策略为伺服电机采取分段式 PD 控制，直流电机为增量式 PID 控制。

PID 控制算法是控制系统中技术比较成熟，应用最广泛的一种控制算法，它的结构简单、相比来说，参数容易调整，不一定需要系统的确切数学模型，因此在各个领域内都有广泛应用。PID 控制算法即将偏差的比例（P）、积分（I）和微分（D）通过线性组合构成控制量，用这一控制量对被控对象进行控制。在 PID 控制算法中有四个比较重要的参量：采样周期、比例系数、积分系数和微分系数。采样周期 T 在

计算机控制系统中是一个很重要参量，从信号的保真来考虑， 采样周期 T 不宜太长，从控制性能来考虑，采样周期 T 应尽可能的短，也就是采样角频率ωs 应尽可能的高，但采样频率越高，对单片机的运算速度要求越快，存储器的容量要求越大，计算机的工作时间和工作量随之增加。另外采样频率高到一定的程度，对系统性能的改善已经不显著了。本系统限于赛道图像的采集频率的限制，采用 20ms 的控制周期，从控制效果来说，也已经足够。比例调节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。微分调节反映系统偏差信号的变化率，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，在偏差还没有形成之前，已被微分调节作用消除。因此，可以改善系统的动态性能。

PID 控制可以分为位置式 PID，增量式 PID 和速度 PID 控制。位置式 PID 数字调节器的输出 u(kT)是全量输出，是执行机构所应到达的位置，数字调节器的输出 u(kT)跟过去的状态有关，运算工作量大，需要对 e(kT)作累加。现在增量式 PID 控制有着广泛的应用，其数字调节器的输出只是增量。增量式算法的优点有：数字调节器只输出增量，计算机误动作时造成的影响比较小；手动-自动切换冲击小；算式中不需要累加，增量只跟最近的几次采样值有关，容易获得较好的控制效果。由于式中无累加，消除了当偏差存在时发生饱和的危险。

增量式 PID 控制的算法为：

$\Delta u(n) = kp\,[e(n) - e(n-1)] + kIe(n) + kD\,[e(n) - 2e(n-1) + e(n-2)]$

$\Delta u(n)$为 n 时刻的输出量，kp 为比例放大系数，kI 为积分放大系数，kD 为微分放大系数，e(n)为 n 时刻的偏差量，e(n-1)为 n −1 时刻的偏差量，e(n-2)为 n −2 时刻的偏差量

5.2 转向控制

根据线性 CCD 图像中提取的道路方向信息，经过组合之后，以道路中心线为基础，确定模型车与中心线的最佳位置关系，在此基础上，比较线性 CCD 组合图像中获得模型车预期位置与最佳位置的关系，从而确定前轮转向角度。遇到障碍时，增大 P 值，有增大摆幅的效果，可顺利通过。

其控制算法原理为：

```
ServoValue[0]=ServoCenter-(int32)(P*direction_error[0]+
D*(direction_error[0]-direction_error[1]));
```

5.3 速度控制

由于在小车的控制过程中，虽然速度的精度要求不高，但是速度的响应要快。

尤其在直道高速转入弯道，上下坡，过人字路，避障时，这点更加突出。但是考虑到速度采样频率和电机响应速度的限制，在设计速度控制器时，采用线性CCD获得跑道区域信息，然后进行区域控制，避免过多地对电机动作影响小车的均速，也避免了上面机构响应慢，突出了调速地快速和平稳性。通过将道路方向信息确定即将驶入的区域的最佳运行速度送入PID控制器，PID控制器通过bang-bang方式与pid相结合改变电枢电压，并在必要时启动反转制动工作方式，实现调速跟随，达到了闭环控制的效果，大大简化速度控制的软件编程，只需根据直道判定算法判别出弯、直道，并根据不同的路况设置不同的速度即可。

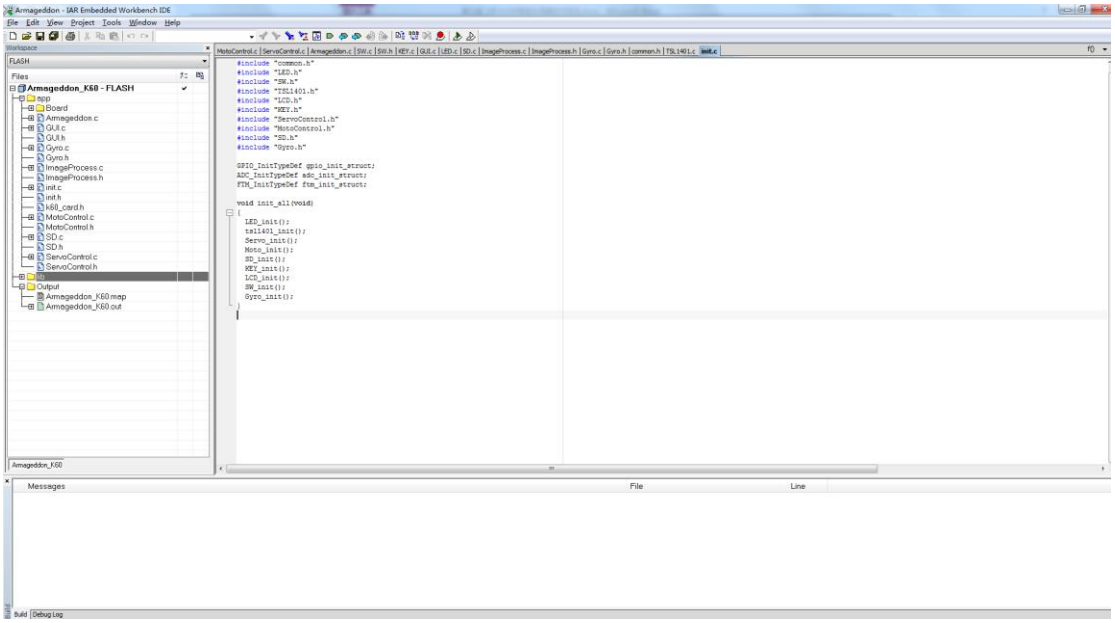
5.4 限速及刹车

将双线性 CCD 引入路径识别后，小车能提前约 60cm 感知弯道来临，即可提前转弯。但在高速行驶状态下，这个提前量远远不够。换句话说，若不引入刹车，小车在弯道冲出跑道的可能性就会非常大。但是刹车时机的选择非常重要，经过实验，我们采取在程序判定出前方 60cm 处的黑线为非直线时进行刹车，效果很好。此外，对人字路引入了适当的反刹控制。

第六章 开发工具和调试

6.1 系统开发工具

开发工具使用的是 IAR Embedded Workbench 开发环境。它能够为单片机 K60 提供与之配套的应用程序开发模块。在目标程序的下载方面，通过 J-Link 与单片机之间的连接下载程序。IAR Embedded Workbench 功能强大，界面简洁严整，内置的 C 编译器编译产生的代码优化度高，执行效率也高。同时 IAR Embedded Workbench 具有强大的在线调试功能，可以充分满足智能车软件系统开发的需要。软件界面如图所示。



22 IAR 开发环境

6.2 上位机

Armageddon 的上位机采用 matlab 开发，用于处理 SD 卡存储的二进制文件，功能包括每一场线性 CCD 图像显示，绘制速度曲线，绘制左、中、右线，绘制舵机转角曲线，绘制陀螺仪曲线，显示全局图像对照等。

上位机编写采用 GUIDE 创建图形界面。步骤如下：

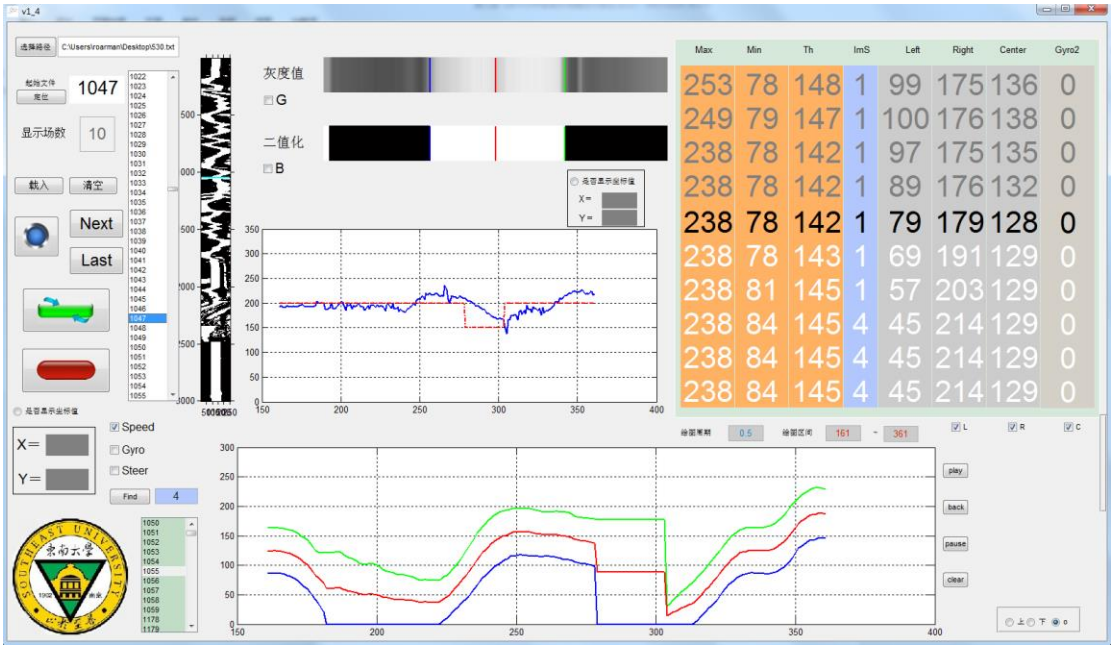
（1） GUI 对象布局

（2） 打开对象的属性查看器，设置对象相应的属性

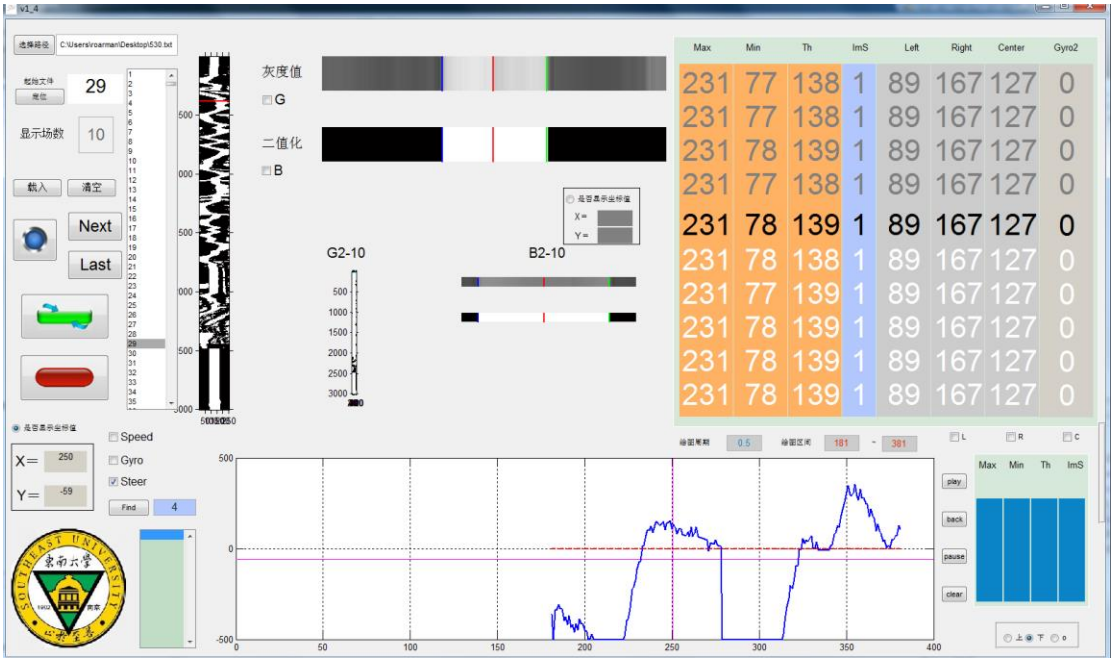
（3） 编写必需的回调函数

若要生成独立可执行的*.exe 文件，还要进行 mcc 编译。

我们量身编写的 matlab 读文件的应用程序图形界面如图：



23 上位机示例图 1



24 上位机示例图 2

第七章 智能车主要技术参数说明

模型车的主要技术参数汇总说明：

a) 改造后的车模总体重量，长、宽、高等基本尺寸参数。

b) 电路功耗，所有电容总容量；
c) 传感器种类以及个数；
d) 除了车模原有的驱动电机、舵机之外伺服电机个数；
e) 赛道信息检测精度、频率。
整理表格如下：

车模主要技术参数名称	参数
车模总质量	1.0kg
车模长度	32cm
车模宽度	18cm
车模高度	36cm
电路总功耗	10W
电路电容总容量	1279.9μF
传感器种类	线性CCD 陀螺仪 编码器
传感器个数	线性CCD*3 陀螺仪*1 编码器*1
赛道信息检测精度	<5mm
赛道信息检测频率	100次/s

25 Armageddon 智能车技术参数表

第八章 总结

自我校的校赛后历时五个多月的制作和不断调试，我们 Armageddon 队最终实现了本届智能车竞赛要求的基本功能以及避障，过坡道和人字路的技术功能，通过华东赛顺利进入全国决赛。在这五个多月的备战过程中，我们汲取前人经验，团结进取，大胆创新，打破固有模式，几经尝试，勇于攻克一个个技术难题，突破性地实现了 3 个线性 CCD 传感器检测赛道的想法，并取得了不错的效果，使我们理论联系实际的能力，动手实践能力，创新能力以及团队协作能力都得到了极大地提高。

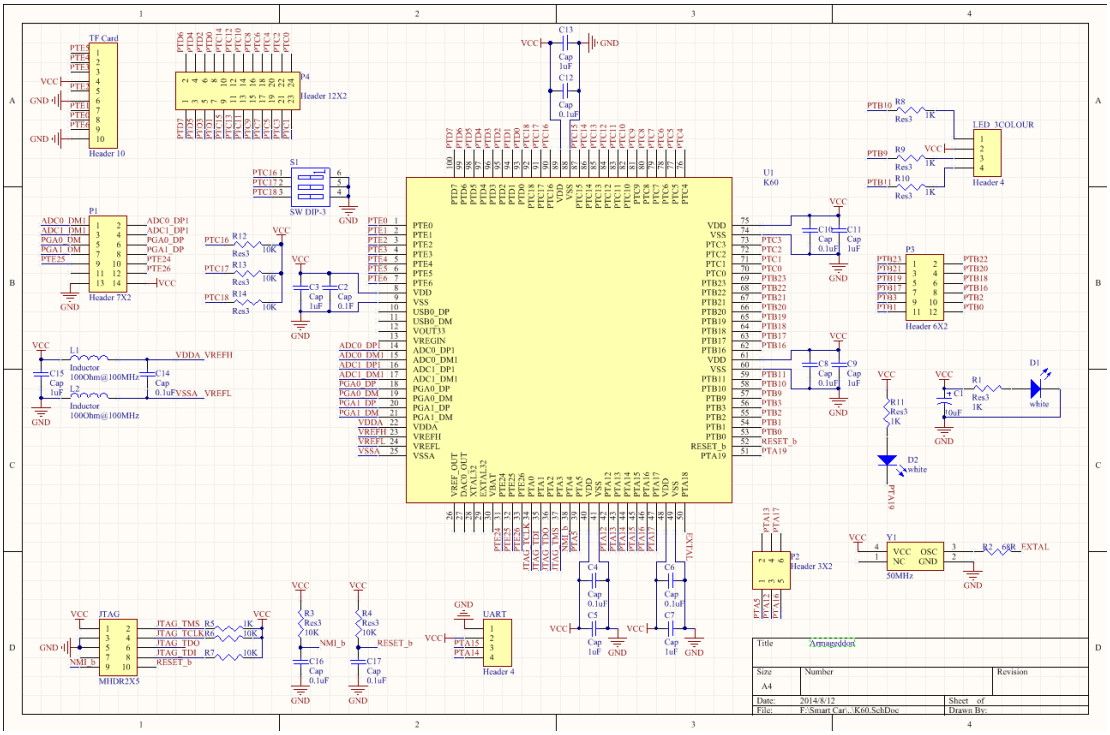
在本次比赛中，我们创造了全新的智能车构架，开始阶段感到压力很大，信心不足，但在两位老师谈英姿、孙琳的倾力支持下，鼓足了坚持的勇气和动力；校队的各位前辈用他们丰富的比赛经验和技术积累帮助我们攻克了许多技术难关、我们的技术方案得以顺利实施，速度的瓶颈实现了飞速突破；东南大学自动化学院提供的训练场地和资金支持让我们有了坚实的后盾和开拓创新的物质基础。其实，比赛的意义不仅仅是拿奖而已，我们在这个过程中收获了更多的知识，信任，扶持，肯定，这些才最为宝贵。最终无论能否捧回 “飞思卡尔”杯，我们都会满怀感激的向曾经支持我们、帮助我们的朋友、老师致以最诚挚的感谢，感谢一路上有你们，Armageddon 承载着我们的希望，你们的祝福，期待它争得属于大家的荣光。

参考文献

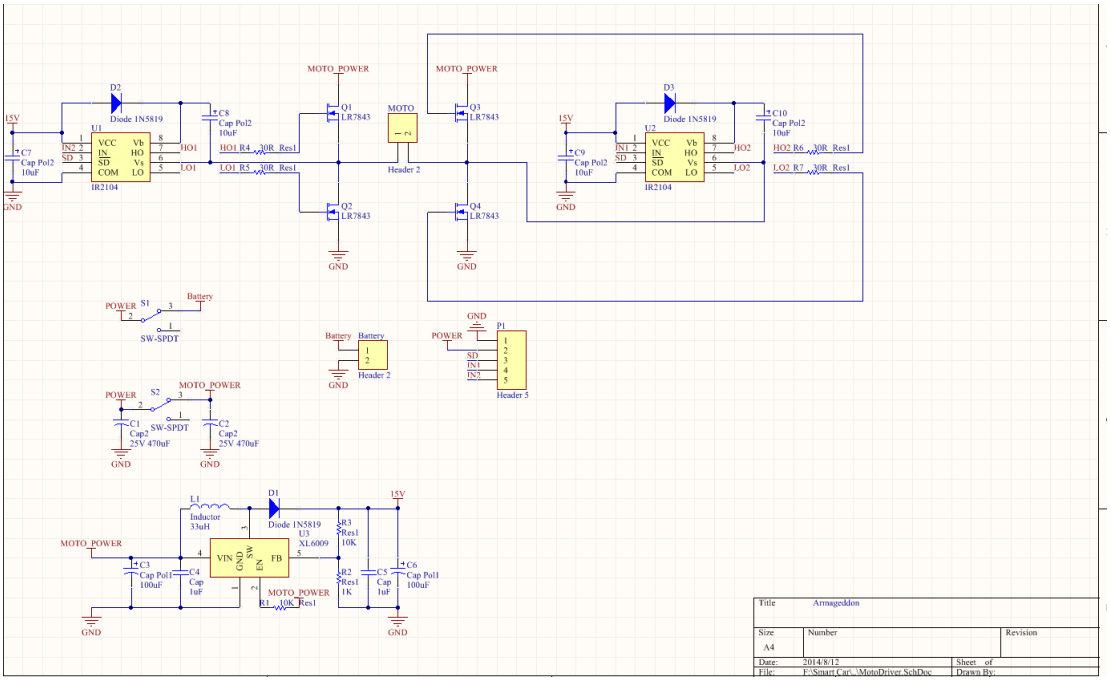
【1】 张贤明编著《MATLAB 语言及应用案例》，东南大学出版社
【2】 秦襄培编著《MATLAB 图像处理与界面编程宝典》，电子工业出版社
【3】 罗华飞编著《MATLAB GUI 设计学习手记》，北京航空航天大学出版社
【4】 卓晴等《学做智能车——挑战“飞思卡尔”杯》，北京航空航天大学出版社
【5】 臧杰，阎岩编《汽车构造》，机械工业出版社
【6】 蔡兴旺编著《汽车构造与原理》，机械工业出版社
【7】 邵贝贝编著《单片机嵌入式应用的在线开发方法》，清华大学出版社
【8】 东南大学“至善”队技术报告
【9】 东南大学“PoWerFect”队技术报告

附录：

附录 A：电路图

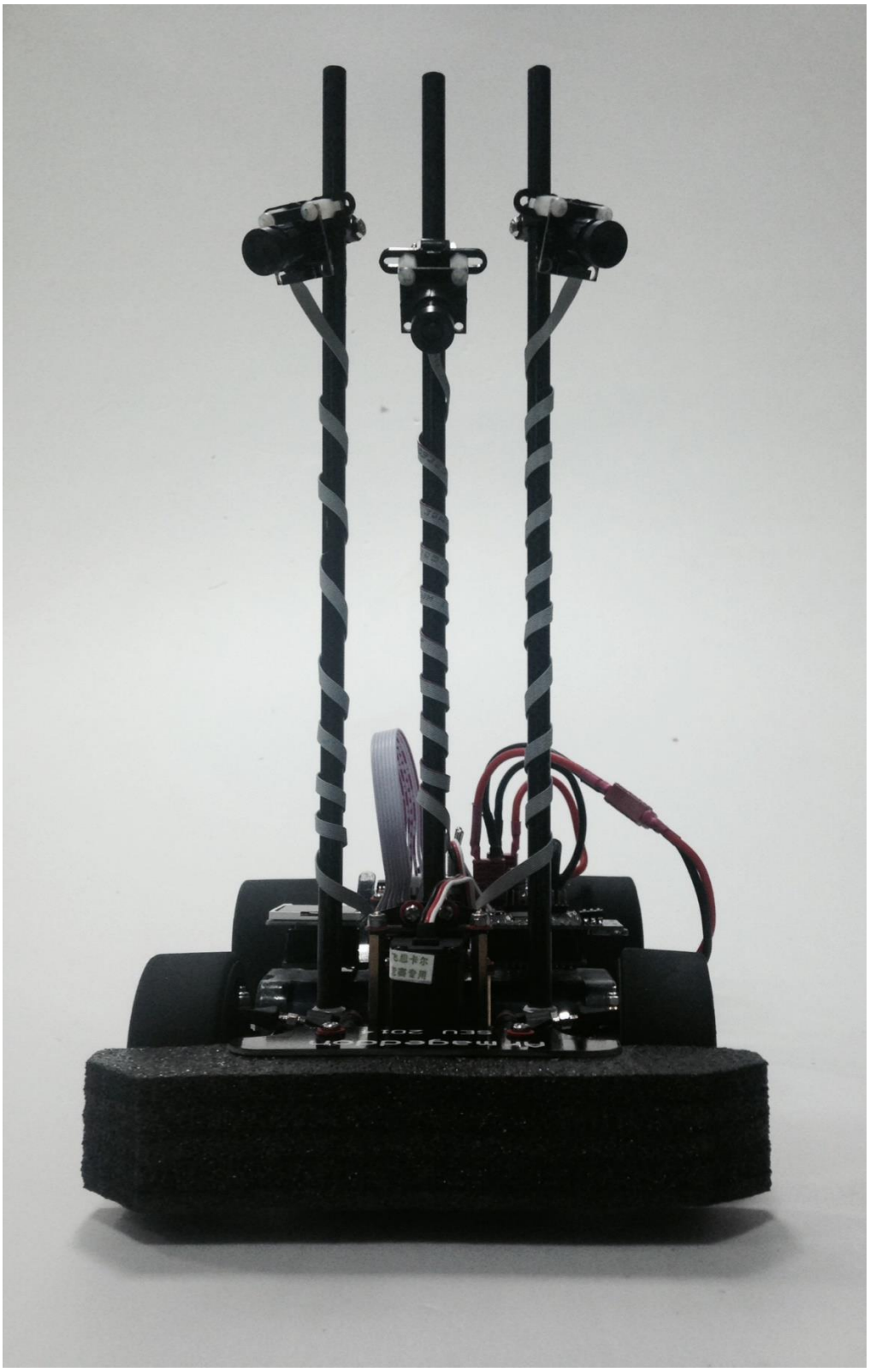


26 核心板原理图


MOTO_POWER
MOTO_POWER
D2
Diode 1N5819
D3
Diode 1N5819
15V
U1
IR2104
U2
IR2104
C7
Cap Pol2
10uF
C8
Cap Pol2
10uF
C9
Cap Pol2
10uF
C10
Cap Pol2
10uF
HO1 R4 30R Res1
LO1 R5 30R Res1
HO2 R6 30R Res1
LO2 R7 30R Res1
Q1
LR7843
Q2
LR7843
Q3
LR7843
Q4
LR7843
MOTO
Header 2
GND
POWER
S1
SW-SPDT
Battery
S2
C1
Cap2
25V 470uF
C2
Cap2
25V 470uF
P1
Header 5
L1
Inductor
33uH
D1
Diode 1N5819
U3
XL6009
R3
Res1
10K
R2
Res1
1K
C3
Cap Pol1
100uF
C4
Cap
1uF
C5
Cap
1uF
C6
Cap Pol1
100uF
R1 10K Res1
Title
Armageddon
Size
A4
Number
Revision
Date:
2014/8/12
Sheet of
File:
F:\Smart Car\..\MotoDriver.SchDoc
Drawn By:

26 电机驱动原理图

附录 B：车模正面照及线性 CCD 镜头选用试验结果



27 车模正面照

对于 Armageddon 智能车 3 线性 CCD 的排布方式，需要选出合适的镜头，使得既能保证它尽量不丢线（十字弯全白），又可以保证较大的前瞻和开阔的视野。经多次试验我们最终确定以下这种镜头选用方案。

CCD	镜头度数	实际前瞻	备注
左	60°	52cm	
中	90°	15cm	仅检测坡道用
右	60°	52cm	

28 镜头具体参数

附录 C：源代码

```
#include "common.h"

#include "init.h"
#include "LED.h"
#include "TSL1401.h"
#include "GUI.h"
#include "ImageProcess.h"
#include "ServoControl.h"
#include "MotoControl.h"
#include "SD.h"
#include "Gyro.h"
//////////////////////////////////////////////////////////////////////////////

//宏定义
#define control_period  10   //控制周期10ms
#define GUI_period      40   //GUI周期40ms
//变量定义
uint8 program_flag=0;  //0:程序完成 1:程序未完成

uint8 PIT_counter=0;

//函数声明
void pit_init(void);
void pit0_isr(void);

//初始化变量定义
PIT_InitTypeDef pit0_init_struct;
/*****************************主函数*********************************/
void main (void)
{
  init_all();
  CalculateCompensationValue();
  pit_init();
  while(1)
  {
    if(program_flag==0)
    {
      ImageProcess();
      Servo_control();
      if(GUI_flag==0)
      {
        Moto_ON;
        Moto_control();
        Display();
        if(SD_flag==1)
        {
          WriteData();
        }
      }
      else
      {
        Moto_OFF;
        GUI();
      }
      program_flag=1;
    }
  }
}
```

```
void pit0_isr(void)
{
  PIT_counter++;

  if(GUI_flag==0) //控制状态
  {
    if(PIT_counter==(control_period-exposure_time))//复位TSL1401
    {
      tsl1401_reset();//复位TSL1401
    }
    if(PIT_counter==control_period) //曝光结束 一整个周期也结束 采集新数据
    {
      PIT_counter=0;
      if(program_flag==0) //程序未完成
      {
        WARNING_LED(LED_ON);
      }
      else                //程序已完成
      {
        if(SD_flag==1)
        {
          WARNING_LED(LED_OFF);
        }
      }
        //舵机赋值
        LPLD_FTM_PWM_ChangeDuty(FTM0, FTM_Ch2, pulse_width_to_duty(ServoValue[0]));
        //电机赋值
        LPLD_FTM_PWM_ChangeDuty(FTM1, FTM_Ch0, MotoValue1);
        LPLD_FTM_PWM_ChangeDuty(FTM1, FTM_Ch1, MotoValue2);
        //传感器数据采集
        tsl1401_get_grey();
        Gyro_get_value();
        Speed = -LPLD_FTM_GetCounter(FTM2);//获取FTM2的正交解码计数值
        LPLD_FTM_ClearCounter(FTM2);//清空计数器
        //标志位
        program_flag=0;

    }
  }
  else //GUI状态
  {
    if(PIT_counter==(GUI_period-exposure_time))//复位TSL1401
    {
      tsl1401_reset();//复位TSL1401
    }
    if(PIT_counter==GUI_period) //曝光结束 一整个周期也结束 采集新数据
    {
      PIT_counter=0;
      if(program_flag==0) //程序未完成
      {
        WARNING_LED(LED_ON);
      }
      else                //程序已完成
      {
        WARNING_LED(LED_OFF);
        //舵机赋值
        //LPLD_FTM_PWM_ChangeDuty(FTM0, FTM_Ch2, pulse_width_to_duty(ServoValue[0]));
```

```
        LPLD_FTM_ClearCounter(FTM2);//清空计数器
        //标志位
        program_flag=0;
      }
    }
  }
}
//////////////////////////////////////////////////////////////////////////////
void pit_init(void)
{
  //配置PIT0参数
  pit0_init_struct.PIT_Pitx = PIT0;
  pit0_init_struct.PIT_PeriodMs = 1;  //定时周期 1ms
  pit0_init_struct.PIT_Isr = pit0_isr;//设置中断函数
  //初始化PIT0
  LPLD_PIT_Init(pit0_init_struct);
  //使能PIT0
  LPLD_PIT_EnableIrq(pit0_init_struct);
}
```

```
#include "common.h"
#include "LED.h"
#include "SW.h"
#include "TSL1401.h"
#include "LCD.h"
#include "KEY.h"
#include "ServoControl.h"
#include "MotoControl.h"
#include "SD.h"
#include "Gyro.h"

GPIO_InitTypeDef gpio_init_struct;
ADC_InitTypeDef adc_init_struct;
FTM_InitTypeDef ftm_init_struct;

void init_all(void)
{
  LED_init();
  tsl1401_init();
  Servo_init();
  Moto_init();
  SD_init();
  KEY_init();
  LCD_init();
  SW_init();
  Gyro_init();
}
```

```c
void tsl1401_get_grey(void)
{
    uint8 i = 128;
    uint8 j;

    TSL1401_SI(1);//SI  = 1
    asm("nop");

    TSL1401_CLK(1);//CLK = 1
    asm("nop");

    TSL1401_SI(0);//SI  = 0
    tsl1401_delay();

    while(i--)
    {
        j=127-i;

        TSL1401_CLK(0);//CLK = 0

        GpixelMain[j]    =LPLD_ADC_Get(ADC0, DAD1);//CCD1
        GpixelUp[j]      =LPLD_ADC_Get(ADC0, AD20);//CCD2;
        GpixelDown[j]    =LPLD_ADC_Get(ADC1, DAD1);//CCD3
        GpixelMain[j+128]=LPLD_ADC_Get(ADC1, AD20);//CCD4;

        TSL1401_CLK(1);//CLK = 1
        asm("nop");
    }

    TSL1401_CLK(0);//CLK = 0
}

//TSL1401复位，用于调整曝光时间
void tsl1401_reset(void)
{
    uint8 i=128;

    TSL1401_SI(1);//SI  = 1
    asm("nop");

    TSL1401_CLK(1);//CLK = 1
    asm("nop");

    TSL1401_SI(0);//SI  = 0
    asm("nop");

    while(i--)
    {
        TSL1401_CLK(0);//CLK = 0
        asm("nop");

        TSL1401_CLK(1);//CLK = 1
        asm("nop");
    }

    TSL1401_CLK(0);//CLK = 0
}
```

```
void ImageProcess(void)
{
  Update();
  Compensation();
  /*********************************************************/
  if(ImageStateMain[0]==0)//第一场
  {
    MaxMain=FindMax(GpixelMain,0,255);
    MinMain=FindMin(GpixelMain,0,255);
    ThresholdMain[1]=CalculateThreshold(MaxMain,MinMain,binaryzation_parameter);
  }
  Binaryzation(256,ThresholdMain[1],GpixelMain,BpixelMain);
  ImageProcessMain();
  if(ImageStateMain[0]==1)
  {
    MaxMain=FindMax(GpixelMain,TrackLeftMain[0],TrackRightMain[0]);
    MinMain=FindMin(GpixelMain,0,255);
    ThresholdMain[0]=CalculateThreshold(MaxMain,MinMain,binaryzation_parameter);
  }
}
```

```
//高电平脉宽（单位us）转换为占空比函数
uint32 pulse_width_to_duty(uint32 pulse_width)
{
  double period=1000000/ServoFreq;          //周期 单位us
  return (uint32)(10000*((double)(pulse_width)/period)); //转化为占空比 0%~100%
}

uint32  ServoCenter=1500; //舵机中心值 左小右大
//舵机初始化
void Servo_init(void)
{
  ftm_init_struct.FTM_Ftmx = FTM0;       //使能FTM0通道
  ftm_init_struct.FTM_Mode = FTM_MODE_PWM;       //使能PWM模式
  ftm_init_struct.FTM_PwmFreq = ServoFreq;       //PWM频率

  LPLD_FTM_Init(ftm_init_struct);

  LPLD_FTM_PWM_Enable(FTM0, //使用FTM0
                      FTM_Ch2, //使能Ch2通道
                      pulse_width_to_duty(ServoCenter), //初始化
                      PTA5, //使用Ch2通道的PTA5引脚
                      ALIGN_LEFT         //脉宽左对齐
                      );
}

#define ServoMax 500

/*****************转向舵机控制*********************/

int32  DeltaServoValue=0;
int32  AllWhiteCounter=0;

int16  direction_error[2]={0,0};
uint32 ServoValue[20];

void Servo_control(void)
{
  uint8 i;

  float P,D;

  direction_error[1]=direction_error[0];
```

```
    if(ImageStateMain[1]==ImageStateMain[0])//上一场跟这一场状态一样
    {
      D=24;
      //if((direction_error[0]-direction_error[1])<0) D=D*1.2;

      ServoValue[0]=ServoCenter-(int32)(P*direction_error[0]+D*(direction_error[0]-direction_error[1]));
    }
    else //上一场跟这一场状态不同 只用P
    {
      ServoValue[0]=ServoCenter-(int32)(P*direction_error[0]);
    }
      D=0;
      MotoValue1+=P*(Speed_error[0]-Speed_error[1])+I*Speed_error[0]+D*(Speed_error[0]-2*Speed_error[1]+Speed_er

      if(MotoValue1>9000)       MotoValue1=9000;
      else if(MotoValue1<0)     MotoValue1=0;
      MotoValue2=0;
    //}
  //}
}
```

```
void Gyro_get_value(void)
{
  uint32 temp=0;
  uint8 i;

  i=9;
  while(i--)
  {
    Gyro_value_original[i+1]=Gyro_value_original[i];
    Gyro_value[i+1]=Gyro_value[i];
  }
  Gyro_value_original[0]=LPLD_ADC_Get(ADC0, AD15);

  i=10;
  while(i--)
  {
    temp+=Gyro_value_original[i];
  }
  Gyro_value[0]=temp/10;
}
```

```
void WriteData(void)
{
  uint16 i,j;

  i=256;
  while(i--)
  {
    j=255-i;
    sdhc_dat_buffer1[i]=GpixelMain[j];
  }
  i=128;
  while(i--)
  {
    j=127-i;
    sdhc_dat_buffer1[256+i]=GpixelUp[j];
  }
  i=128;
  while(i--)
  {
    j=127-i;
    sdhc_dat_buffer1[384+i]=GpixelDown[j];
  }

  i=256;
  while(i--)
  {
    j=255-i;
    sdhc_dat_buffer1[512+i]=BpixelMain[j];
  }
  i=128;
  while(i--)
  {
    j=127-i;
    //sdhc_dat_buffer1[768+i]=BpixelUp[j];
  }
  i=128;
  while(i--)
  {
    j=127-i;
    sdhc_dat_buffer1[896+i]=BpixelDown[j];
  }
  sdhc_dat_buffer1[1024]=MaxMain;
  sdhc_dat_buffer1[1025]=MinMain;
  sdhc_dat_buffer1[1026]=ThresholdMain[0];
  sdhc_dat_buffer1[1027]=ImageStateMain[0];
  sdhc_dat_buffer1[1028]=TrackLeftMain[0];
  sdhc_dat_buffer1[1029]=TrackRightMain[0];
  sdhc_dat_buffer1[1030]=TrackCenterMain[0];
```

```
void LED_colour(uint8 colour)
{
  switch (colour)
  {
    case 0:
      LPLD_GPIO_Output_b(PTB, 9, 1);
      LPLD_GPIO_Output_b(PTB, 10, 1);
      LPLD_GPIO_Output_b(PTB, 11, 1);
    break;

    case 1:
      LPLD_GPIO_Output_b(PTB, 9, 1);
      LPLD_GPIO_Output_b(PTB, 10, 0);
      LPLD_GPIO_Output_b(PTB, 11, 1);
    break;

    case 2:
      LPLD_GPIO_Output_b(PTB, 9, 0);
      LPLD_GPIO_Output_b(PTB, 10, 1);
      LPLD_GPIO_Output_b(PTB, 11, 1);
    break;

    case 3:
      LPLD_GPIO_Output_b(PTB, 9, 1);
      LPLD_GPIO_Output_b(PTB, 10, 1);
      LPLD_GPIO_Output_b(PTB, 11, 0);
    break;

    case 4:
      LPLD_GPIO_Output_b(PTB, 9, 0);
      LPLD_GPIO_Output_b(PTB, 10, 0);
      LPLD_GPIO_Output_b(PTB, 11, 1);
    break;

    case 5:
      LPLD_GPIO_Output_b(PTB, 9, 1);
      LPLD_GPIO_Output_b(PTB, 10, 0);
      LPLD_GPIO_Output_b(PTB, 11, 0);
    break;

    case 6:
      LPLD_GPIO_Output_b(PTB, 9, 0);
      LPLD_GPIO_Output_b(PTB, 10, 1);
      LPLD_GPIO_Output_b(PTB, 11, 0);
    break;

    case 7:
      LPLD_GPIO_Output_b(PTB, 9, 0);
      LPLD_GPIO_Output_b(PTB, 10, 0);
      LPLD_GPIO_Output_b(PTB, 11, 0);
    break;
  }
}
```