第十七届全国大学生
智能汽车竞赛

技 术 报 告

学　　校：　广东工业大学

队伍名称：　霹雳火

参赛队员：　吴滨楠、刘骏帆、钟锐城

带队教师：　王日明、王嘉茉

关于技术报告和研究论文使用授权的说明

本人完全了解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：吴浩楠 刘强帆 钟锐城

带队教师签名：孙嘉荣

日　　期：2022 年 8 月 20 日

目录

第一章 引言

本文以第十七届全国大学生智能汽车竞赛为背景，介绍了基于摄像头的自主寻迹智能赛车开发流程。在该比赛中，我们采用组委会指定的 C 车模进行开发设计，以英飞凌的 TC264 芯片作为车模的核心处理器，结合 MT9V03X 灰度摄像头作为主要传感器，满足大赛要求的以最快最稳的速度在蓝白色赛道上进行巡线自主驾驶。整个系统涉及机械、硬件、软件等多个层面设计，赛车通过摄像头采集赛道信息并发送到 MCU，通过 MCU 对图像数据进行处理提取到赛道偏差和特殊路况，再通过控制算法计算出最优解并将控制信号发送给电机和舵机，实现对车模的自动化控制。

关键词：TC264，智能车，图像处理

第二章 机械结构设计

2.1 整车布局

本组选用C车模作为比赛用车。包括传感器在内车模长宽高分别为：275mm、175mm、350mm。全车重量1005克。整车由头到尾依次是：舵机、摄像杆、电池、主控板、电机和驱动板。

2.2 电路板安装

整车共两块电路板，主控板放置于车模中部。驱动板放置于电机上端。

2.3 电池安装

电池安放于整车中轴线上，主控板下。往届智能车多将电池放置于整车最后方。本组根据考虑决定不采取该方案，理由如下。电池为除电机外整车最重部件，且本组使用电池较长，若固定于车尾处，容易与后轮发生摩擦，并且增加车辆后端重量。

其次，C 车模为标准后驱车。根据部分车厂提供的后驱车数据，后轮驱车，令前轮可专注于转向，因此在转弯的时候更加敏捷。重量前后中立分配均匀，能拥有良好的操控稳定性和行驶平顺性，并且有利于延长轮胎的使用寿命。因此我们将电池放置在整车中心偏前，以接近整车前后端重量比例 1:1，同时减少后轮磨损程度。

最后是电池放置方向。电池放置还可分为南北走向与东西走向。因南北走向可将车辆重心尽可能往中心线收紧。东西走向在过弯时重心更容易发生左右偏移，会影响过弯姿态，增加不确定性，故采取南北走向。

2.4 舵机安装

舵机的主流安装方式有两种：立式安装和卧式安装。通过对比这两种安装方式，参考往届参赛C车的机械设计，我们发现卧式安装对转向灵敏度的提升并不显著，反而会导致车子的不符合阿克曼转向的原理，所以我们最终放弃了卧式安装，采用立式安装。立式安装的好处在于可以更方便的调整前轮的内八角和阿克曼转向半径，而阿克曼转向原理则可以让我们更科学合理的调整后轮差速来辅助转弯。最后我们将车模原装的舵机拉杆更换为金属拉杆，好处在于金属拉杆的固定更加稳固，从根本上杜绝了车模由于受到外力而使拉杆松动的现象。

2.5 前轮定位

束角对车辆的影响主要有对车辆直线的稳定性，和入弯时的转向反应，以及轮胎胎温的上升情况和磨损程度。前轮如果设置外八字，入弯转向会更灵敏。而过大角度的外八字则有转向过度的倾向。由于外八字不利于直线的行驶，最终决定将束角设为零。

假设我们的车辆四轮外倾角都为 0 ，那么在弯道中，我们外侧吃力轮胎，因为倾斜原因，会只有外侧与地面接触，从而减少了与地面接触面积，减少了抓地力。那么我们把外倾角设定为负数时，在弯道中车辆外侧吃力轮子与我们的地面获得而会最大面积的接触，从而获得更好的弯道抓地力。因此我们将前轮增加了 1° 的外倾角，从而增大过弯时的摩擦力。

2.6 电机与编码器安装

电机与编码器的安装方式是比较固定的，这部分存在三个齿轮的啮合，在安装的时候需要保证齿轮与齿轮之间啮合到位以避免发生空转的现象，更需要保证编码器在获取电机转速时不会受到过多杂波影响。更需要注意左右两套齿轮之间的啮合程度要尽可能相似，这里在安装固定到位之后可以适量使用润滑脂进行齿轮辅助啮合。

2.7 摄像头安装

本次摄像头四轮组的规则没有限制摄像头高度，为了能够更好的看到元素特征，我们选择了把摄像头高度调高至 33cm，安装在车前保证图像的前瞻性。同时在摄像头碳素杆后面增加两条支撑杆，保证小车在行驶过程这种摄像头不会因为太高而晃动。

第三章 硬件电路设计

3.1 整体方案

电路系统稳定是软件系统正常工作的前提。本组采用总钻风 MT9V03X 摄像头来完成所有赛道元素的识别与循迹，外接 TFT 屏幕、按键调试板、蓝牙、陀螺仪等模块辅助进行图像处理以及控制参数的调试，进而实现整车的功能。

基于可靠、高效的原则，并在满足对应电路需求的前提下，我们尽可能地将电路简洁化，并自行设计了主控、驱动和按键调试模块。

可靠性是电路设计的重中之重，需要对电路设计的所有环节进行稳定性模拟测试，解读相关芯片的数据手册，做好各部分的屏蔽滤波等工作，将模拟电路与数字电路分离，防止交叉部分形成干扰。

高效性是指能量损耗低。通过适当使用开关电源，提高电源效率。在驱动电路上，要求降低驱动 H 桥开关的损耗以及内阻，从而降低发热程度。

简洁是指在达到一定的可靠性的前提下，为了减轻整车重量，降低整车重心位置，通过更换更简易电路，减少元器件使用数量，改变排列、走线等方法对电路板布局进行压缩，以起到减轻电路板重量，增大电路集成度，减少高速信号传输干扰的作用。

3.2 主控板

智能车的主控板由四个部分组成：TC264DA 核心板，电源稳压电路，外设。

3.2.1 单片机最小系统

我们直接采用逐飞科技出品的 TC264DA 核心板。其是一款高性能、低功耗跨界处理器，具有 200MHz 和 DSP 功能的双三核，2.5MB 闪存，752KB 的 RAM，带 ECC 保护。下图为主板上设计核心板的接口。

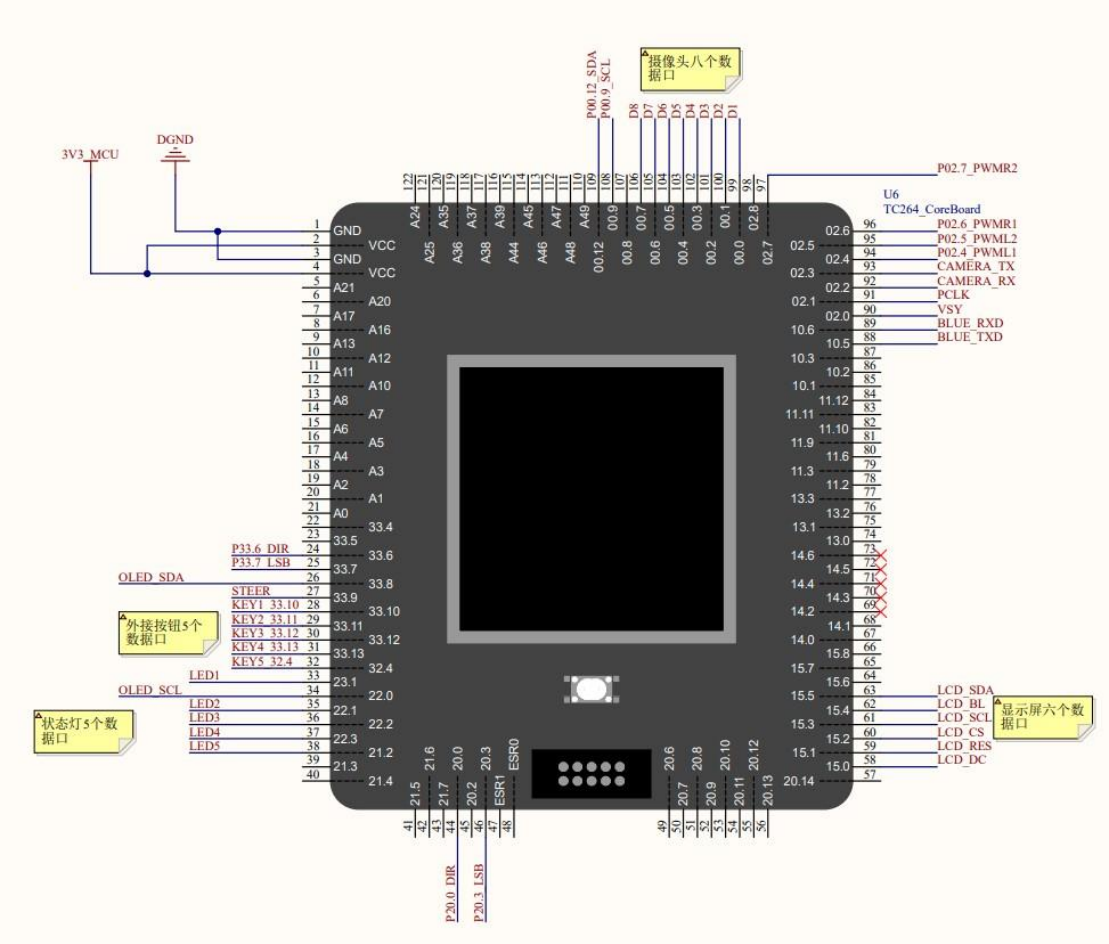

图 1 TC264 核心板

3.2.2 电源稳压电路

电源分为开关稳压电源与线性稳压电源。线性电源的电压反馈电路工作于线性状态，一般是将输出电压取样然后与参考电压送入电压比较放大器，以此放大器的输 出作为电压调整管的输入，用以控制调整管使其解电压随输入的变化而变化，从而调整其输出电压；开关电源是指用于电压调整的晶体管工作于饱和区或截至区，即开关状态，开关电源主要通过改变调整管开和关的时间，即占空比，从而改变输出电压的大小。

主控板稳压电路分别需要 6V，5V，3.3V 供电。5V 负责给电池 7.4V 与 3.3V 电压缓冲。3.3V 给单片机、摄像头、编码器、蓝牙、陀螺仪、TFT 供电。6V 给舵机供电。由于 6V 与 5V 电路功耗较大，且需要较高电源效率，故使用开关稳压电路。3.3V 用电器较多，且多数需要较小纹波，因而使用 3 枚 LDO 芯片，分

别给摄像头、单片机及其他外设供电。

开关稳压电路选取 TPS54202。TPS54202 具有完善的保护电路，包括过流，过压，电压反接保护。使用这个芯片只需要极少的外围元件就能构成高效稳压电路。

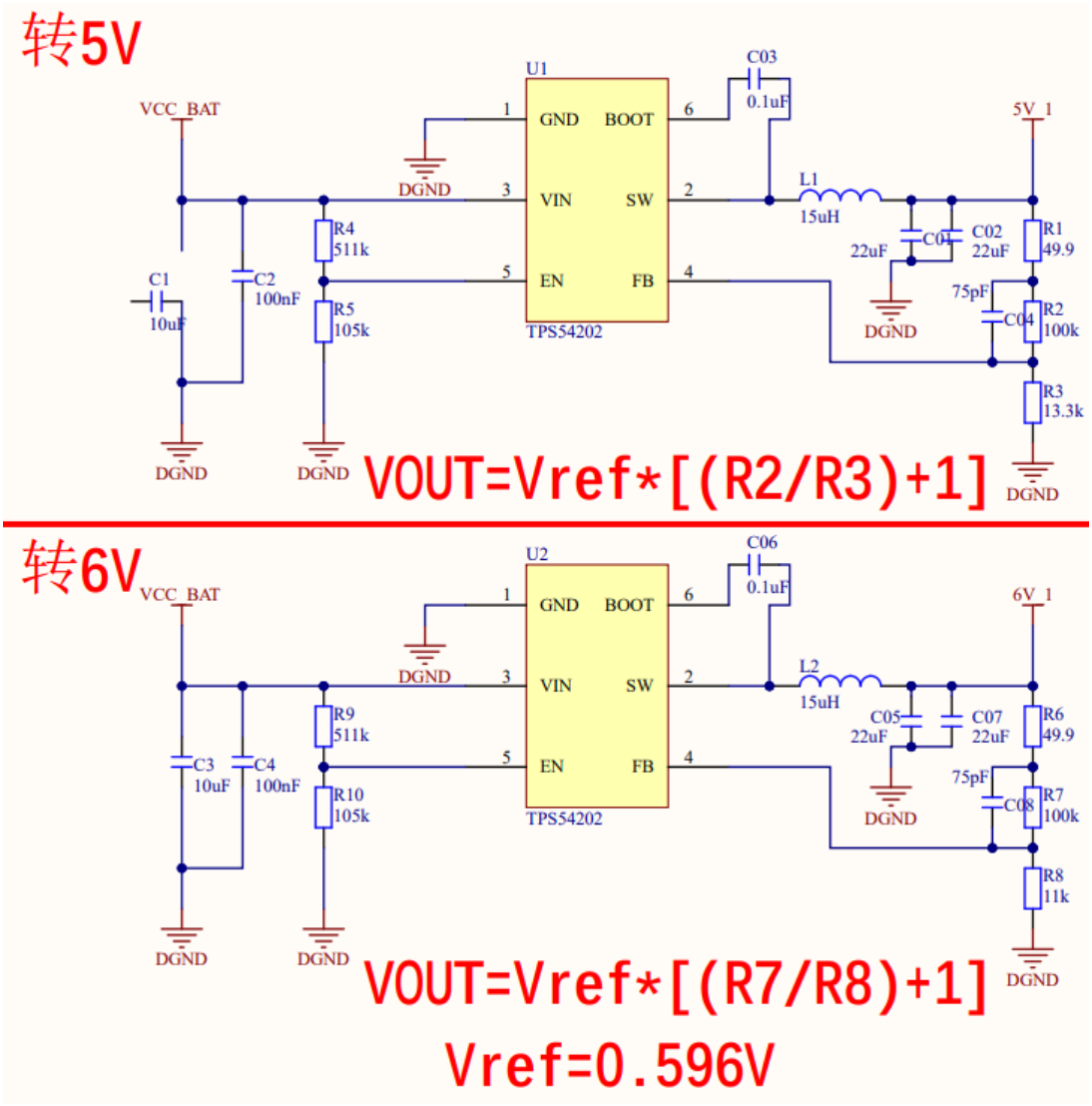

图 2 电源降压电路

线性稳压电路采用 AMS1117-3V3。AMS1117 的外围元件仅有 4 个电容，整体电路极为简便，同时也能满足使用需求。

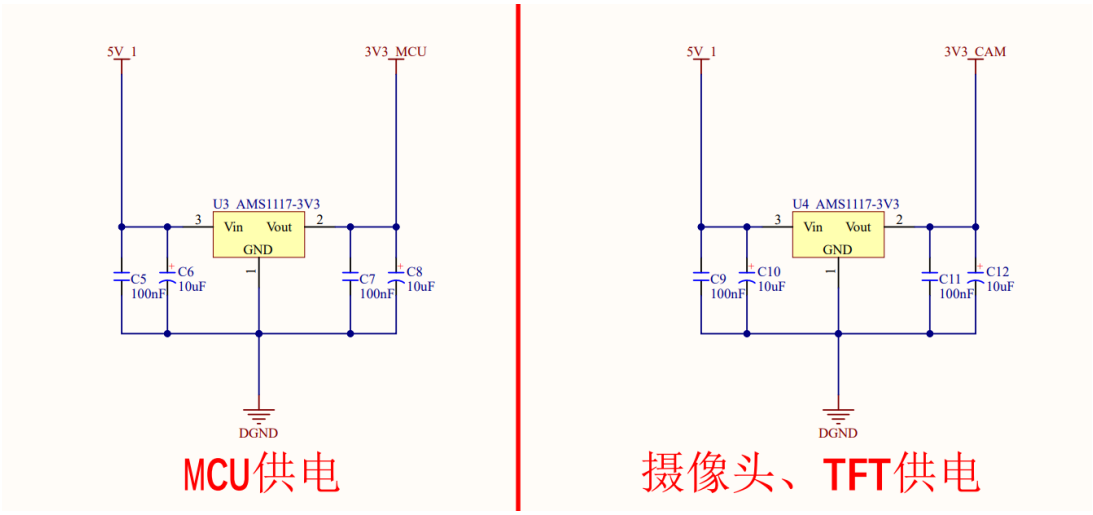

图 3 各模块稳压电路原理图

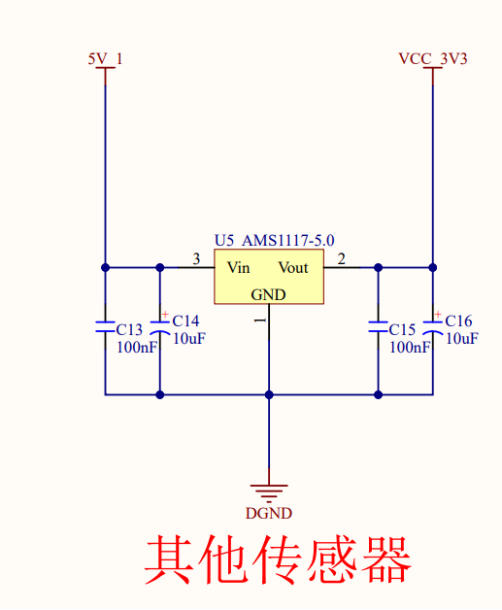

图 4 传感器线性稳压电路原理图

3.3 驱动板

3.3.1 电源稳压电路

驱动板部分分别需要 3.3V、5V、12V 供电。5V 与 3.3V 负责降压给隔离芯片供电。12V 负责给驱动芯片 IR2104 供电。隔离芯片电流要求较小，因而使用 AMS1117 进行降压。驱动芯片所需的 12V 由 SX1308 及相关外部电路组成升压电路进行供应。对应电路图入下所示。

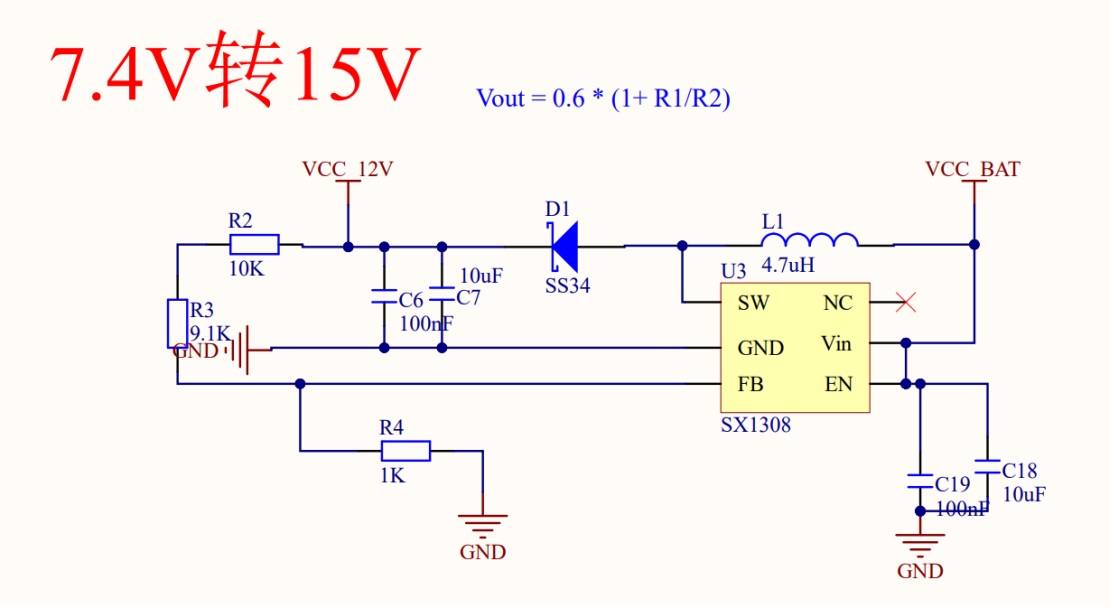

图 5 驱动板升压电路原理图

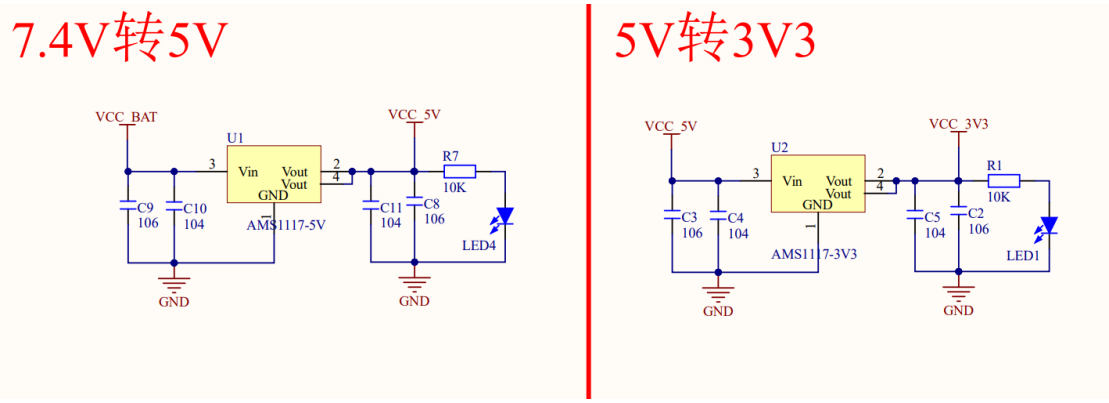

图 6 驱动板降压电流原理图

3.3.2 电机驱动电路

电机驱动电路由驱动芯片 IR2104、MOS 管 HSU3018B 以及电阻电容肖特二极管组成。

IR2104 引脚定义：SD 信号为使能信号，高电平有效，芯片工作。IN 为高电平时 HO 为高，LO 为低；IN 为低电平时，HO 为低，LO 为高电平。我们的 PWM

信号就是在这里输入（因为片内自带了 CMOS 和 LSTTL 电平兼容器，可以直接输入而不用考虑电平转换）。Vb 是高侧浮动电源输入脚，HO 是高侧门极驱动输出，Vs 是高侧浮动电源回流。这三个控制上半桥的 MOS 导通。Vcc 是低侧浮动及参考电源输入脚，LO 是低侧门极驱动输出，COM 是低侧回流。这三个控制下半桥的 MOS 导通。同时 IR2104 可以控制半桥的核心在于其 Vb 和 Vs 脚之间外接的“自举电容”。

MOS 管 HSU3018B 可以承受 30V110A 的冲击，远超电池所能提供的功率，因此提高了驱动电路的稳定性。

电路图如下所示：

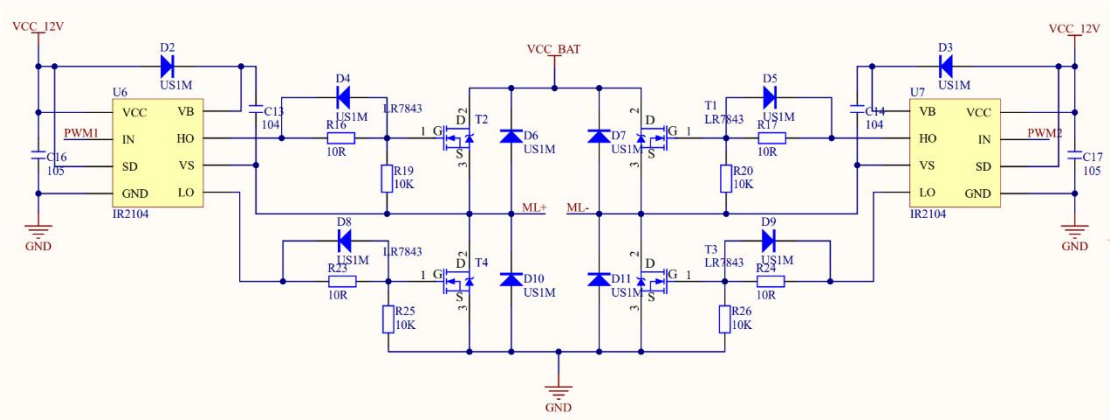

图 7 电机驱动电路原理图

3.4 按键调试模块

按键调试模块由五个按键及一块 0.96 寸 OLED 屏幕组成。负责车辆参数的调节。3D 图如下：

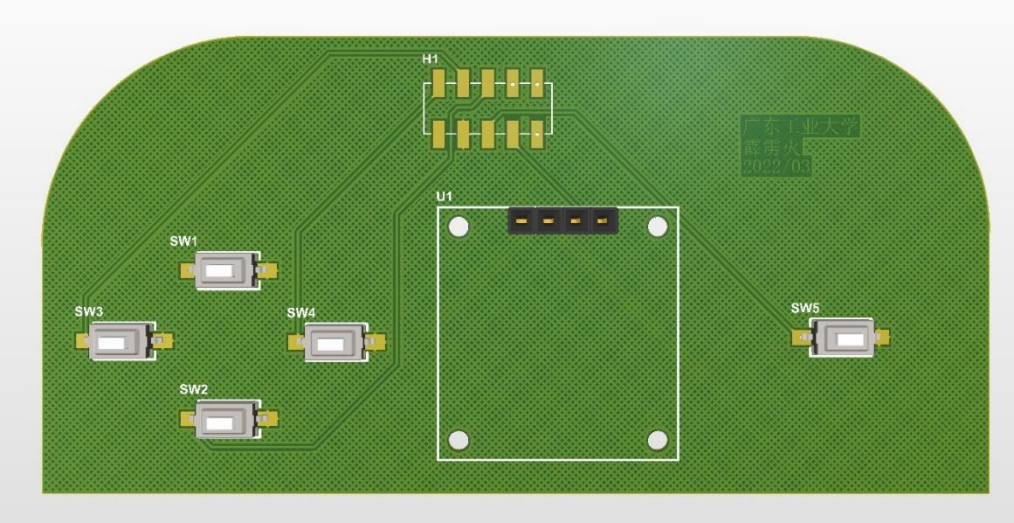

图 8 按键模块 3DPCB 图

第四章 软件设计

4.1 整体方案

基于英飞凌 TC264 单片机的优越性能，我们采用双核来处理程序，并对双核进行明确分工：性能较强劲的 1 核进行图像处理，而由 0 核进行电机控制并承担一部分陀螺仪数据处理的工作。在芯片启动之后，1 核会通过总钻风摄像头不断地进行图像数据的采集和处理，并将处理后的数据以全局变量的形式传递给 0 核，0 核通过 1 核传递过来的已经处理完的数据，可以很方便地对电机进行控制，其中包括一个舵机和两个直流有刷电机。而 1 核若发来陀螺仪数据处理的请求也是一样可以快速响应。

对于 1 核的图像处理，我们采用顺序执行的方式，即将所有的程序都放在主循环程序中，摄像头采集图像后触发并通过 DMA 中断传入单片机，单片机首先对灰度图像进行二值化处理，然后进行扫线、拐点提取等，最后进行赛道特殊元素判断并更新赛道偏差。

对于 0 核的电机控制和陀螺仪数据处理，我们采用中断出发的方式，即将所有的程序都放在定时器中断中，通过设置定时器定时地对电机进行控制，其中对舵机的控制周期是 20ms，对直流有刷电机的控制周期是 6ms，对陀螺仪的角度积分周期是 2ms，需要注意的是陀螺仪积分的开启与关闭由 1 核控制，并不会一直处于工作状态。

4.2 图像处理算法

4.2.1 灰度图像二值化

从 MT9V03X 总钻风摄像头采集到的为灰度图像，即：图像为单通道类型图，二维数组代替了整个图像，每个元素的值即为这个像素的灰度值。由于比赛赛道为蓝布为底赛道为白，这两种颜色在灰度图中的灰度值相差比较大，蓝色的灰度值较小，白色的灰度值较大；那么我们在这里就可以使用固定阈值法对图像进行二值化：设定一个灰度值作为阈值，像素点的灰度值大于阈值我们则认为这个像素点是赛道把它设为白点灰度值为 255，像素点的灰度值小于阈值则认为这个像素点是赛道外把它设为黑点灰度值为 0。把全部的像素点进行了比较之后，就得到了非黑即白的二值化图像，从而能把赛道以及赛道外给分割出来。但是由于赛道每个地方的光线是不一样的，所以二值化的固定阈值是不好确定的，那么在这里我们选择了使用“大津法”二值化阈值算法，来对图像二值化的阈值进行求解得出适合的二值化阈值，使得图像二值化后不至于失真。

4.2.2 赛道边界处理

为了给车子一个寻迹的赛道偏差，我们对二值化之后的图像进行进一步处理，主要是对赛道边界进行提取，通过提取出赛道左右的边界线，从而得到赛道的中线来计算车子寻迹的赛道偏差。对赛道边界提取的方法是通过简单的左右扫线，即从图像中间位置开始，对整幅图像进行从下往上、从中间往两边的遍历，若我们遍历到图像像素点由白跳变到黑这样的阶跃点，即将该点视为赛道边界。在左、右都提取到赛道边界之后，将这样的点的坐标存储到数组中，并对左、右两个点的 X 坐标进行求均值的处理得到赛道中点的 X 坐标，也存储到数组中。且该 X 坐标作为下一行扫线起点的 X 坐标，这样的处理可以加快扫线速度。

以上就是正常情况下的扫线处理，但是还有一种意外情况：车子本身已经偏离赛道，即我们给出的扫线起点的 X 坐标在赛道外，这样会导致直接左右扫线寻找不到正确的赛道边界。我们给出的解决方法是对扫线起点进行像素点是黑或白的判断，如果起点本身是在赛道外的话，我们将认为此时车身已经偏离到赛道外了需要在扫线的时候进行纠正。具体纠正方法是通过向左右两边寻找是否有赛道的存在，这样的处理可以帮助车子在极端情况下返回到正确的赛道上，而不会进入未知状态。

在进行基础扫线的时候，除了基础的左中右三线数组的记录，我们还会记录下一些有用的数据用于在后面进行拐点、元素判断。我们将左右扫线寻找不到阶跃点的情况视为丢线，我们会在扫线的同时记录左右边界的丢线数。

4.2.3 车库元素

由于车库元素前面的斑马线是个非常明显的特征，所以车库元素的第一步应该识别车库前面的斑马线，来作为开启车库找点补线；而斑马线的识别可以通过横向方向上的黑白跳变来判断此处是否是斑马线；判断成功之后，在找寻车库元素的特征拐点：靠近赛道距离车子在远端的直角拐点，找到这个拐点之后，无论是不入库时还是入库时，都以这一点去进行补线操作即可。

4.2.4 十字回环元素

十字回环元素是在一个状态机中进行管理，该状态机的开启条件是识别到存在拐点，并且在拐点的上方存在一个黑洞（圆环），这时我们进行对拐点的路径进行补线直行并进入下一个状态。

第二个状态是使用陀螺仪辅助入环，持续对存在拐点的一边进行补线直行，直到积分达到目标角度（20°），认为车子已经进入圆环中，取消补线并关闭积分进入下一个状态。

第三个状态是车子在环中自主寻迹，并使用陀螺仪辅助环中状态，直到积分达到目标角度（200°），此时车子还未到达圆环出口，但已经接近圆环出口，关闭积分进入下一个状态。

第四个状态是通过摄像头对圆环出口进行识别判断，判断条件为检测到拐点的存在并且正前方不存在赛道。此时根据进入圆环时的方向进行对应方向的补线实现出环，同时开启陀螺仪积分辅助出环，当车子已经完全出环，此时关闭积分状态机结束，车子返回正常赛道行驶并准备下一元素的识别判断。

4.2.5 三岔元素

三岔元素中最关键的元素特征即为三岔的上 Y 顶点，只需要找到了上 Y 顶点即可识别出三岔并且补线进入三岔了，同理出三岔也是一样的。由于车子进入三岔前的路径不好或者三岔前面接了一个弯道，所以三岔会有正入斜入的情况；但是无论如何只需要抓住三岔的最显著特征，Y 拐点即可很好的识别并且处理到三岔路口。所以对此我们队将三岔分为了四种情况：正入、入三岔中丢失了左右拐点、左斜入、右斜入。使用丢线数和拐点是否存在，来先简单区分出这四种情况之后，在由这四个情况找到自己设定的左右下拐点去搜寻上拐点即可。

4.2.6 环岛元素

环岛元素是在一个状态机中进行管理，该状态机的开启条件与十字回环状态机的开启条件比较相似，唯一的不同点是加入了车子前方必须是直道这一条件，

当开启条件满足后与十字回环相似，先进入补线状态。与此同时开启编码器测距功能，当检测到小车行走了 0.3m 后，此时小车已通过环岛的出口，我们开始识别环岛中部这一特征。这里的特征是下方存在一个黑洞（圆环，根据左右环岛情况位置会不同，存在于左右两边）。当成功识别到这一特征之后，进行下一状态。

第二个状态此时车子行驶到环岛中部，开始判断环岛入口并补线入环。判断环岛入口主要是存在拐点且车子位于直道上，需要根据拐点的位置进行补线辅助车子完成入环动作，同时开启陀螺仪积分，直到积分达到目标角度（30°），关闭陀螺仪积分进入下一状态。

第三个状态时小车在环中进行自主寻迹，这里主要靠开启陀螺仪积分并直到积分达到目标角度（180°），关闭积分进入下一状态。

第四个状态时小车已经接近环岛出口，开始进入环岛出口的识别，识别环岛出口和识别十字回环的出口基本一致，在识别到出口后进行补线并开启陀螺仪积分辅助出环，当积分达到目标角度（60°）取消补线，关闭陀螺仪进入下一状态。

第五个状态是小车已经出环，但是会再次经过环岛出口，此时就需要我们手动补线让小车直行通过，这里也是运用编码器测距的功能来确定跳出该状态。

4.3 电机控制算法

4.3.1 直流有刷电机

C 车模上共有两个相同型号的直流有刷电机，我们使用两个增量式 PI 控制器分别对其进行速度闭环控制。对于 PID 算法，网上已经有各种各样的教程可以参考。简单描述一遍，PID 控制器是一个线性控制器，该控制器共有 P（比例）、I（积分）、D（微分）三个系数，分别对输入量进行处理后线性叠加，得到输出量。我们在电机的速度闭环控制上选用了增量式 PI 控制器，即对传统 PID 公式进行求微分运算，得到其微分表达式，该表达式计算出来的值是系统输出的增量，叠加在原输出量上，而不是直接输出一个全新的值来控制电机。使用增量式控制的好处在于电机的转速在车子运动过程中只会有小幅度的变化，其输出值不会有很大幅度的变动。只使用 PI 两个参数对电机进行控制，而不使用 D 项也是这个原因。

选择 PI 控制器之后，就是调参环节，我们对电机的速度响应要求首先是稳（稳定在目标速度），然后是快（目标速度改变时电机快速跟上）。调参有两种方法：一是根据工程经验直接上手，二是对控制对象进行建模。这里主要简述第二

种方法，对直流减速电机进行建模。电机的建模方法有两种，一种是手动测量电机的各项参数，然后使用 MATLAB 的电机仿真工具进行仿真；第二种也是我采用的方法，直接测量电机的输入和输出之间的关系，然后使用 MATLAB 的系统辨识函数，对电机系统进行函数辨识，直接得出电机的系统传递函数。这里需要注意的是，在测量电机的输入和输出数据的时候，最好是将车子放置在赛道上，保证测量时电机转动受到的阻力和比赛时是一样的。在得到电机的系统传递函数后，再使用 MATLAB 中专门对 PID 调参的工具，PID Tuner，可以很方便地得到一套基础参数，然后再到赛道上进行简单微调，得到最终的参数。

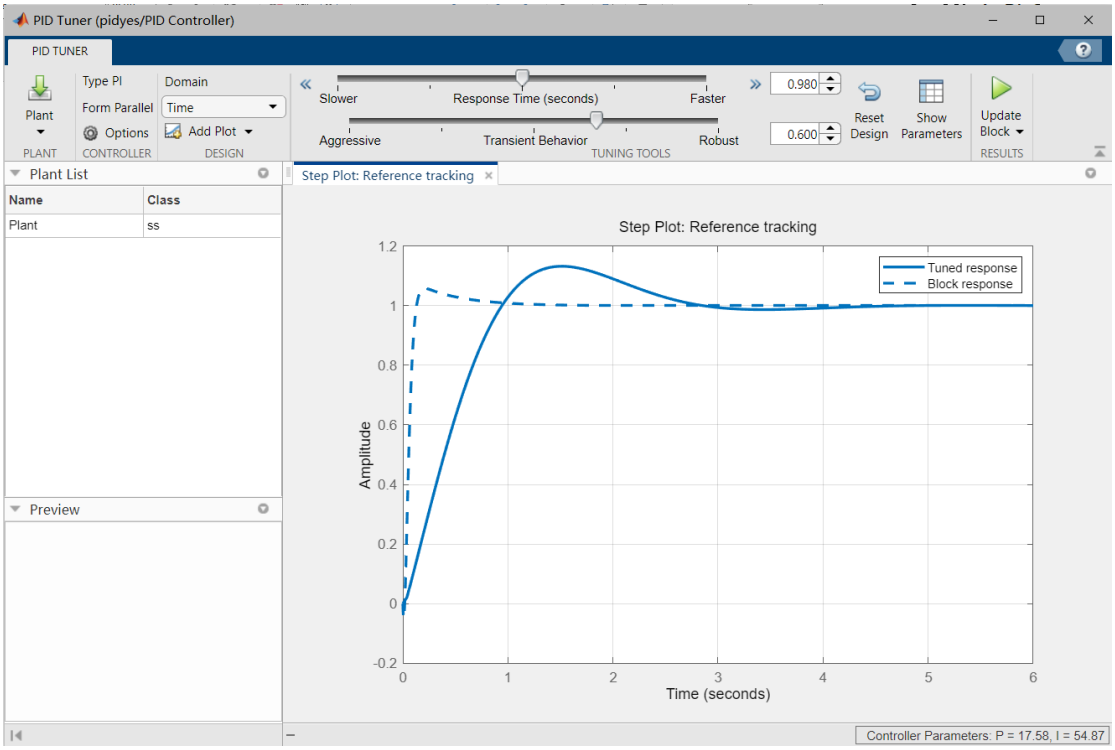

图 9 Matlab 仿真电机速度环结果图

4.3.2 模拟舵机

舵机我们采用的是型号为 S3010 的模拟舵机，所以 PWM 输出频率最高只能设定在 50Hz，即 20ms 一个控制周期。舵机连接的是车子的前轮，主导车子的转向。主要是使用赛道偏差来作为输人值对舵机的转向环进行控制，这里同样使用 PID 算法，具体使用位置式 PD 控制器对赛道偏差进行处理。

舵机的控制归根结底是对赛道偏差的处理，我们根据赛道偏差和舵机的打角映射关系可以得到基础 P 参数，再根据车子运动时的姿态来调整 D 参数。其中 P 项主导转弯打角大小，D 项辅助入弯提前响应以及出弯时避免超调。

4.4 陀螺仪传感器数据采集

我们的车子的传感器除了摄像头之外，还有一个型号为 ICM20602 的陀螺仪加速度计。这是一个六轴的传感器模块，我们使用到了其中一个轴，即绕 Z 轴的角速度，我们对这个角速度进行积分，得到车子在水平方向上转动的角度。由于 ICM20602 是一个六轴的传感器，没有地磁计可以校准，所以计算得到的角度和实际车子转动的角度会存在一定的误差。我们使用采集一段陀螺仪数据来校准零漂，加上我们对陀螺仪的积分时间较短且精度要求不高，最终计算出来的角度还是比较满意的。

4.5 人机交互模块

我们的人机交互主要由三部分组成：一是屏幕和 LED 灯显示，二是按键调参，最后是蓝牙发送模块。我们的主板上留有一个 1.8 寸 LCD 屏幕的接口，在平时调试图像处理的时候可以显示图像，便于我们直接在赛道上进行代码测试和调试。而在车子运动的时候是不需要屏幕的，这时我们也可以很方便的将屏幕取下，防止屏幕对车子运动的姿态造成干扰。主板上还预留着五颗不同颜色的可编程 LED 灯和一个蓝牙发送模块的接口，这些 LED 和蓝牙模块的作用在于车子运动的时候也可以为我们提供车子的一些信息，弥补只有屏幕显示的不足。最后的人机交互，也是比赛时最重要的，就是按键模块。我们的按键模块供有五个按键和一个 OLED 屏幕，在车子发车前可以外接该模块对车子的某些参数进行修改，便于我们在赛场上随机应变。

第五章 系统调试

5.1 开发工具

对于软件程序的编译和运行，我们使用英飞凌官方提供的 AURIX-studio 软件对 TC264 进行开发。该软件拥有友好的编程界面，同时也拥有强大的 Debug 功能。

在对车模的控制进行调试的过程中，我们会通过蓝牙模块采集数据到计算机进行分析，其中对于计算机端的串口接收我们使用免费的高自由度上位机 VOFA+。该上位机拥有极致简约的协议设计，同时拥有多种绘图插件，对于我们 PID 的参数调试起到很大作用。

5.2 制作、安装过程

车模的硬件部分是由一位队友单独设计、焊接、测试的。而软件部分的图像处理和控制算法，则由另外二人一起合作完成。每人分工合作，首先将自己的任务独立开到一个系统中进行测试，当功能测试完毕之后再归并到车模系统上进行测试，这样的组件式开发有利于队友之间明确分工，高效开发。

车模的系统设计并不是一开始就固定下来一成不变的，而是伴随着开发的进程，速度由低到高，

5.3 系统调试

对于整个车模系统来说，可分为机械、硬件、软件，我们对这三大模块各自进行稳定性测试后再组合到一起，而一旦出现车模跑飞的情况，我们会首先检查机械，其次是软件，再是硬件。并且每次在对车模修改时，都保证只修改到一处，而不是多处同时修改，这样可以保证单独变量进行调试而不会被其他变量干扰到。

整个系统中又以软件为重，软件中又分为图像处理和控制算法，当车模姿态不稳定或者跑飞时，我们亦是单独对其中的某一变量进行处理。对于图像处理，我们将车模停在赛道某处，通过 LCD 屏幕显示摄像头采集到的图像信息和处理后的数据进行调试。对于控制算法的优化，我们通常将赛道的特殊元素挡住，使赛道形成一个单路径的、简单的循环赛道。将车模置于赛道上，关闭特殊元素的识别，只保留巡线功能进行 PID 调参。在这个过程中，同时使用蓝牙模块发送赛道偏差、电机速度、舵机打角等数据到 VOFA+上位机，通过上位机的二维绘图功能可以清晰直观地了解车模的运行姿态以及参数调整方向。

第六章 主要技术参数

车模类型	C 车模
车模整体尺寸（长、宽、高）	275mm、175mm、350mm
传感器种类及数量	摄像头 MT9V03X*1 陀螺仪 ICM20602*1
微处理器型号及数量	英飞凌 TC264*1
电池规格及数量	3000mAh、7.4V 锂电池*1
电路板数量及功能	主控板*1、驱动板*1
车模零件更换改装情况	更换舵机摆臂拉杆球头为金属材质

第七章 总结

第十七届智能车比赛就要过去了，第一次参加智能车比赛，选择了最有激情的四轮组，由于是小白选手，加上没有太多前人经验，我们的做法和方案大多是从网上的一些开源方案开始写起来的。能够一路走到全国总决赛，这一个赛季也没有遗憾了。图像处理算法和控制算法都是比较传统做法。摄像头四轮组，图像处理能够得到一条比较好的中线之后，不用太精调控制即可以将车速提到3m/s，写图像处理的时候有上位机的拿上位机调试没上位机就像我们写图像一样，对着屏幕手推车修图像就好，不要凭空想象的去写，会写出一堆 BUG。车体机械方面，我们队伍对于机械方面的调整也不是很专业，只能参考前人的技术报告去做对应的处理。最后感谢智能车交流群的车友们，大家的相互分享才能促进共同的成长，其实还有很多其他高效稳定的处理方案我们还没来得及去尝试，就要接着去提速而还是选择了之前的方案去接着写，这一点是我觉得比较遗憾的一点，如果我都去尝试了也许会更加的有趣吧。

参考文献

[1] 薛安克 彭冬亮 陈雪亭. 自动控制原理[M] 西安电子科技大学出版社, 2001

[2] 李全钊. 单片机原理及接口技术[M]. 高等教育出版社，2009

[3] 闫琪 王江 熊小龙 朱德亚 邓飞贺 朱锐 金立. 智能车设计[M]. 北京航空航天大学出版社，2016

[4] Rafael C. Gonzalez Richard E. Woods 阮秋琦 阮宇智. 数字图像处理[M]. 电子工业出版社，2020

附录 A 电路主要原理图

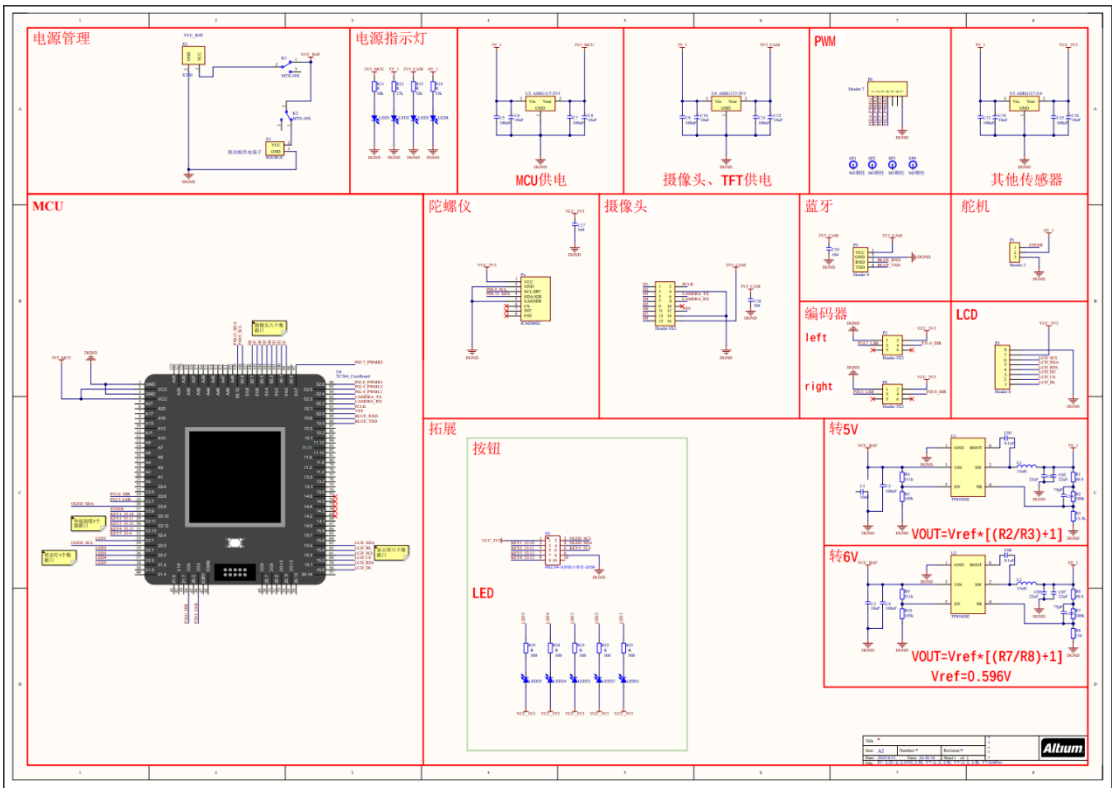

图 10 主控板原理图

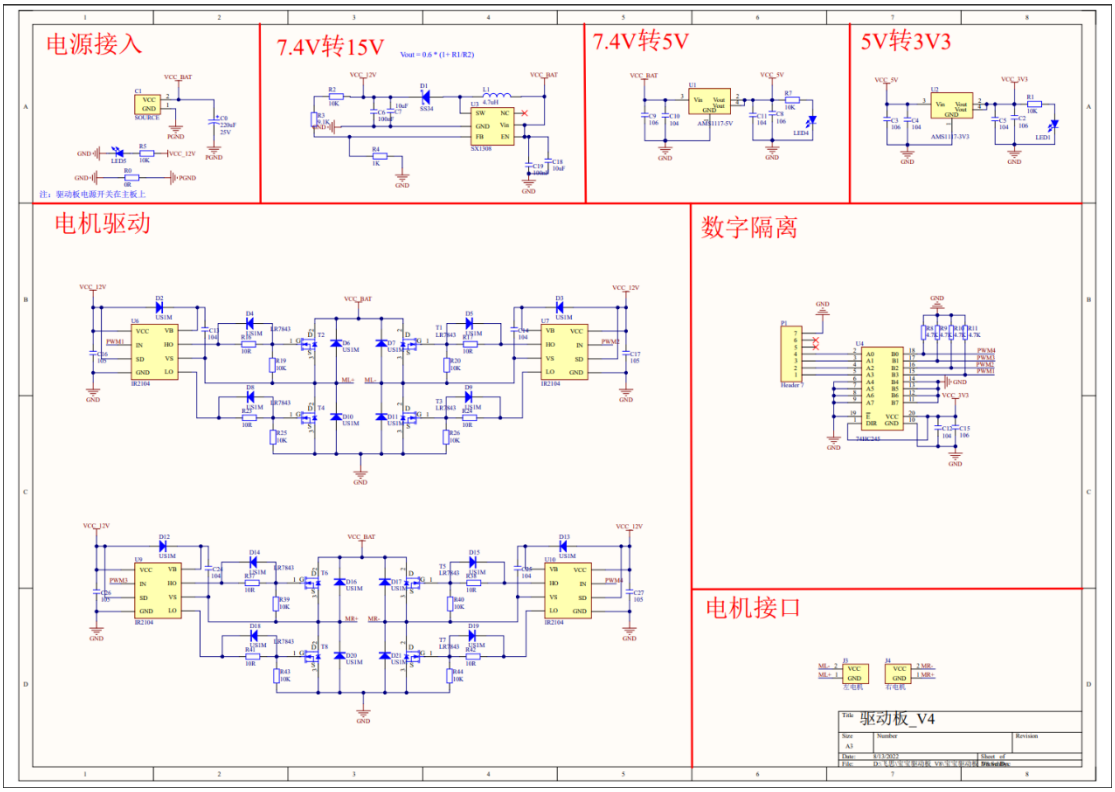

图 11 驱动板原理图

附录 B 算法核心代码

```
/*************************************************************
 ** 函数功能: 寻找三岔左边上顶点，防止因为扫线扫到三岔另外一边使得Bias 太小
 ** 参      数: uint8 row,uint8 cloumn:去找寻虚拟补线折现上点的基准点坐标
 **           Point *LeftUpPoint:左边的特殊点
 ** 返 回 值: 无
 ** 说      明: 用完这个函数之后应该再补一条左边的垂直线和右边从上拐点补到左上角拐点
 ** 作      者: 刘骏帆

*************************************************************/
typedef struct SeedGrowAqueue
{
    uint8 front;//队头
    uint8 rear;//队尾
}SeedGrowAqueue;//找到拐点之后再次八领域扫线的种子生长的队列
uint8 const l_data[8] = {0,1,2,3,4,5,6,7};//左种子标号队列数据域
uint8 const r_data[8] = {4,3,2,1,0,7,6,5};//右种子标号队列数据域
void ForkSearchLineAgain(int *UpInflectionY,Point left_seed,Point right_seed)
{
    SeedGrowAqueue seed_queue;//左右线用于标号转移的循环队列
    seed_queue.front = 0; seed_queue.rear = 7;
    //左种子生长:第三个条件是代表周围八个领域不是白色
    int left_seed_grow_starcolumn=left_seed.X-1;
```

```
for(;left_seed.X>left_seed_grow_starcolumn && left_seed.Y>bias_endline-1 && left_seed.Y<*UpInflectionY+1 && (seed_queue.front + 1) % 8 != seed_queue.rear;)
    {
        uint8 break_for_flag = 0;//用于判断是否找到黑色的区域从而不继续标号查看生长
        //判断种子是丢线还是已经到了前边的大片黑色区域
        if (left_seed.X + 1 >= MT9V03X_W && BinaryImage[left_seed.Y][left_seed.X] == IMAGE_BLACK)//左种子长到了右边，并且还是黑色说明这一段已经步入了全黑,赛道外了
        {
            LeftLine[left_seed.Y] = 0;
            left_seed.Y--;
        }
        else if (left_seed.X - 1 <= 0 && (BinaryImage[left_seed.Y][left_seed.X] == IMAGE_WHITE|| BinaryImage[left_seed.Y-1][left_seed.X] == IMAGE_WHITE))//左种子一直在左边边界，并且是白色，说明是正常丢线
        {
            LeftLine[left_seed.Y] = 0;
            left_seed.Y--;
        }
        else
        {
            for (; (seed_queue.front + 1) % 8 != seed_queue.rear; seed_queue.front++)
            {
                switch (l_data[seed_queue.front])
```

```
                {
                case 0:
                    if (BinaryImage[left_seed.Y][left_seed.X + 1] ==
IMAGE_BLACK)
                    {
                        left_seed.X++;
                        break_for_flag = 1;
                    }
                    break;
                case 1:
                    if (BinaryImage[left_seed.Y - 1][left_seed.X + 1] ==
IMAGE_BLACK)
                    {
                        left_seed.X++; left_seed.Y--;
                        break_for_flag = 1;
                    }
                    break;
                case 2:
                    if (BinaryImage[left_seed.Y - 1][left_seed.X] ==
IMAGE_BLACK)
                    {
                        left_seed.Y--;
                        break_for_flag = 1;
                    }
                    break;
                case 3:
                    if (BinaryImage[left_seed.Y - 1][left_seed.X - 1] ==
```

```
IMAGE_BLACK)
                    {
                        left_seed.X--; left_seed.Y--;
                        break_for_flag = 1;
                    }
                    break;
                case 4:
                    if (BinaryImage[left_seed.Y][left_seed.X - 1] ==
IMAGE_BLACK)
                    {
                        left_seed.X--;
                        break_for_flag = 1;
                    }
                    break;
                case 5:
                    if (BinaryImage[left_seed.Y + 1][left_seed.X - 1] ==
IMAGE_BLACK)
                    {
                        left_seed.X--; left_seed.Y++;
                        break_for_flag = 1;
                    }
                    break;
                case 6:
                    if (BinaryImage[left_seed.Y + 1][left_seed.X] ==
IMAGE_BLACK)
                    {
```

```
                left_seed.Y++;
                break_for_flag = 1;
            }
            break;
        case 7:
            if (BinaryImage[left_seed.Y + 1][left_seed.X + 1] ==
IMAGE_BLACK)
            {
                left_seed.X++; left_seed.Y++;
                break_for_flag = 1;
            }
            break;
        default:
            break;
        }
        //在这里判断一下前面的 switch 有没有找到黑色区域
        if (break_for_flag == 1)
        {
            char temp = seed_queue.front;
            if (temp - 2 < 0)//判断是否-2 会小于 0，会的话就下一
次从 0 号找
            {
                seed_queue.front = 0;
            }
            else
            {
                seed_queue.front = temp - 2;
```

```
                }
                seed_queue.rear = (seed_queue.front + 7) % 8;//重置队
头
                break;//跳出这次的 8 领域寻找，进入下一个种子 8 领
域搜索
            }
        }
        //通过点的右边是否是白色来判断它是否是边界点，否则边界
点可能会在黑色区域，但是种子还要继续往右边走，却没有被存入了
        if (BinaryImage[left_seed.Y][left_seed.X + 1] ==
IMAGE_WHITE)
        {
            LeftLine[left_seed.Y] = left_seed.X;

CentreLine[left_seed.Y]=(left_seed.X+RightLine[left_seed.Y])/2;
        }
    }
}
//右种子生长
seed_queue.front = 0; seed_queue.rear = 7;//重置队头和队尾，但是数据域
进行改变即可
int right_seed_grow_startline=right_seed.Y;
for(;right_seed.Y>bias_endline-1 &&
right_seed.Y<right_seed_grow_startline+1 &&
(seed_queue.front+1)%8!=seed_queue.rear;)
{
    uint8 break_for_flag = 0;//用于判断是否找到黑色的区域从而不继续
```

标号查看生长

//判断种子是丢线还是已经到了前边的大片黑色区域

if (right_seed.X - 1 <= 0 && BinaryImage[right_seed.Y][right_seed.X] == IMAGE_BLACK)//右种子长到了左边，并且还是黑色说明这一段已经步入了全黑,赛道外了

{

RightLine[right_seed.Y] = MT9V03X_W-1;

right_seed.Y--;

}

else if (right_seed.X + 1 >= MT9V03X_W && (BinaryImage[right_seed.Y][right_seed.X] == IMAGE_WHITE || BinaryImage[right_seed.Y - 1][right_seed.X] == IMAGE_WHITE))//右种子一直在右边边界，并且是白色，说明是正常丢线

{

RightLine[right_seed.Y] = MT9V03X_W - 1;

right_seed.Y--;

}

else

{

for (; (seed_queue.front + 1) % 8 != seed_queue.rear; seed_queue.front++)

{

switch (r_data[seed_queue.front])

{

case 0:

if (BinaryImage[right_seed.Y][right_seed.X + 1] == IMAGE_BLACK)

{

```
                right_seed.X++;
                break_for_flag = 1;
            }
            break;
        case 1:
            if (BinaryImage[right_seed.Y - 1][right_seed.X + 1] ==
IMAGE_BLACK)
            {
                right_seed.X++; right_seed.Y--;
                break_for_flag = 1;
            }
            break;
        case 2:
            if (BinaryImage[right_seed.Y - 1][right_seed.X] ==
IMAGE_BLACK)
            {
                right_seed.Y--;
                break_for_flag = 1;
            }
            break;
        case 3:
            if (BinaryImage[right_seed.Y - 1][right_seed.X - 1] ==
IMAGE_BLACK)
            {
                right_seed.X--; right_seed.Y--;
                break_for_flag = 1;
```

```
                }
                break;
            case 4:
                if (BinaryImage[right_seed.Y][right_seed.X - 1] ==
IMAGE_BLACK)
                {
                    right_seed.X--;
                    break_for_flag = 1;
                }
                break;
            case 5:
                if (BinaryImage[right_seed.Y + 1][right_seed.X - 1] ==
IMAGE_BLACK)
                {
                    right_seed.X--; right_seed.Y++;
                    break_for_flag = 1;
                }
                break;
            case 6:
                if (BinaryImage[right_seed.Y + 1][right_seed.X] ==
IMAGE_BLACK)
                {
                    right_seed.Y++;
                    break_for_flag = 1;
                }
                break;
            case 7:
```

```
                if (BinaryImage[right_seed.Y + 1][right_seed.X + 1] ==
IMAGE_BLACK)
                {
                    right_seed.X++; right_seed.Y++;
                    break_for_flag = 1;
                }
                break;
            default:
                break;
            }
            //在这里判断一下前面的 switch 有没有找到黑色区域
            if (break_for_flag == 1)
            {
                char temp = seed_queue.front;
                if (temp - 2 < 0)//判断是否-2 会小于 0，会的话就下一
次从 0 号找
                {
                    seed_queue.front = 0;
                }
                else
                {
                    seed_queue.front = temp - 2;
                }
                seed_queue.rear = (seed_queue.front + 7) % 8;//重置队
头
                break;//跳出这次的 8 领域寻找，进入下一个种子 8 领
```

域搜索

```
                    }
                }
                //通过点的右边是否是白色来判断它是否是边界点，否则边界点可能会在黑色区域，但是种子还要继续往右边走，却没有被存入了
                if (BinaryImage[right_seed.Y][right_seed.X - 1] == IMAGE_WHITE)
                {
                    RightLine[right_seed.Y] = right_seed.X;

CentreLine[right_seed.Y]=(LeftLine[right_seed.Y]+right_seed.X)/2;
                }
            }
        }
        //如果找到了结束行停下来的那么更新上拐点的结束行便于循迹做偏差
    //      LcdDrawRow(left_seed.Y, PURPLE);
    //      lcd_showint32(TFT_X_MAX-50, 2, left_seed.Y, 3);
        if(left_seed.Y<*UpInflectionY) *UpInflectionY=left_seed.Y;
    }

    /*************************************************************
     ** 函数功能: 根据两点进行补线(直线)
     ** 参      数: int *LeftLine: 左线，int *CentreLine: 中线，int *RightLine: 右线 （变为全局变量了）
     **             char Choose: 选择补左线还是右线
     **             Point StarPoint:      起点
     **             Point EndPoint:       终点
```

```
**  返  回  值: 无
**  作       者: 刘骏帆
**  注       意：- StarPoint.Y>EndPoint.Y
**                - 把图像映射到第四象限进行 y=kx+b 的操作，y 先全取负运算之后，描黑的时候再负运算
**                - 2022/2/27 17:40 DeBuglog: K 应该为浮点型，否则精度损失为 0

*****************************************************************/
void FillingLine(char Choose, Point StarPoint,Point EndPoint)
{
    float K;//斜率为浮点型，否则 K<1 时，K=0
    int B,Y,X;

    /*特殊情况：当要补的线是一条垂线的时候*/
    if(EndPoint.X==StarPoint.X)
    {
        for(Y=StarPoint.Y;Y>EndPoint.Y;Y--)
        {
            switch(Choose)
            {
                case 'L':
                    LeftLine[Y]=StarPoint.X;
                    CentreLine[Y]=(StarPoint.X+RightLine[Y])/2;
                    break;
                case 'R':
```

```
                    RightLine[Y]=StarPoint.X;
                    CentreLine[Y]=(LeftLine[Y]+StarPoint.X)/2;// 在里面进行中线的修改，因为不会出现补两边的情况，正入十字就直接冲，斜入就补一边而已
                    break;
                default:break;
            }
        }
        return;
    }

    K=(float)(-EndPoint.Y+StarPoint.Y)/(EndPoint.X-StarPoint.X);//k=(y2-y1)/(x2-x1)，强制类型转化否则会损失精度仍然为 0
    B=-StarPoint.Y-K*StarPoint.X;//b=y-kx

    for(Y=StarPoint.Y;Y>EndPoint.Y;Y--)
    {
        X=(int)((-Y-B)/K);          //强制类型转化：指针索引的时候只能是整数

        //判断 X 会不会越界
        if(X<0)                     X=0;
        else if(X>MT9V03X_W-1) X=MT9V03X_W-1;

        switch(Choose)
        {
            case 'L':
```

```
                    LeftLine[Y]=X;
                    CentreLine[Y]=(X+RightLine[Y])/2;
                    break;
                case 'R':
                    RightLine[Y]=X;
                    CentreLine[Y]=(LeftLine[Y]+X)/2;//在里面进行中线的修改，因为不会出现补两边的情况，正入十字就直接冲，斜入就补一边而已
                    break;
                default:break;
            }
        }
    }

    /*
     ** 函数功能: 扫线提取左中右三线的坐标
     ** 参      数: *LeftLine：左线数组
     **             *CentreLine：中线数组
     **             *RightLine：右线数组
     **             path：默认扫线方向
     ** 返 回 值: 无
     ** 作      者: WBN
     */
    void GetImagBasic(int *LeftLine, int *CentreLine, int *RightLine ,char path)
    {
        uint8 row,cloum;                    //行,列
```

```
uint8 flag_l=0,flag_r=0;       //记录是否丢线 flag，flag=0：丢线
uint8 num=0;                   //记录中线连续丢失的次数
LostNum_LeftLine=0;LostNum_RightLine=0; //把丢线的计数变量清零


//开始扫线(从下往上,从中间往两边),为了扫线的严谨性,我们做 BORDER_BIAS 的误差处理，即扫线范围会小于图像大小
for(row=MT9V03X_H-1;row>0;row--) //从下往上，遍历整幅图像
{
    //在赛道外，优先按 path 扫线方向寻找赛道
    if(BinaryImage[row][Mid]==IMAGE_BLACK)  //扫线中点是黑色的（中点在赛道外）
    {
        num++;      //中线连续丢失，+1
        if(path=='L')    //默认向左扫线
        {
            //先向左边扫线，寻找右边界点
            for(cloum=Mid;cloum-BORDER_BIAS>0;cloum--)      //向左扫
            {
                if(BinaryImage[row][cloum]==IMAGE_WHITE    && BinaryImage[row][cloum-BORDER_BIAS]==IMAGE_WHITE)  //找到白点（从赛道外扫到赛道内：黑-白）
                {
                    RightLine[row]=cloum;      //记录右边界点（向左找到的是右边界点）
                    flag_r=1;                  //flag 做无丢线标记
                    break;
```

```
            }
        }
        //根据上面扫线的结果判断丢失的赛道是在左边还是右边从而决定继续向哪边扫线
        if(flag_r==1)    //找到了右边界（丢失的赛道在左边）
        {
            //继续向左寻找左边界
            for(;cloum-BORDER_BIAS>0;cloum--)       //继续向左扫
            {
                if(BinaryImage[row][cloum]==IMAGE_BLACK && BinaryImage[row][cloum-BORDER_BIAS]==IMAGE_BLACK)       //找到黑点：（从赛道内扫到赛道外：白-黑）
                {
                    LeftLine[row]=cloum;    //记录左边界点
                    flag_l=1;                //flag 做无丢线标记
                    break;
                }
            }
        }
        else                //没有找到右边界（丢失的赛道在右边）
        {

for(cloum=Mid;cloum+BORDER_BIAS<MT9V03X_W-1;cloum++)      //向右扫
            {
```

```
                    if(BinaryImage[row][cloum]==IMAGE_WHITE
&& BinaryImage[row][cloum+BORDER_BIAS]==IMAGE_WHITE)
                    {
                        LeftLine[row]=cloum;      //记录左边界点
（向右找到的是左边界点）
                        flag_l=1;                 //flag 做无丢线标
记
                        break;
                    }
                }
                if(flag_l==1)    //找到了左边界（丢失的赛道在右边）
                {
                    for(;cloum+BORDER_BIAS<MT9V03X_W-
1;cloum++) //继续向右扫
                    {

if(BinaryImage[row][cloum]==IMAGE_BLACK                    &&
BinaryImage[row][cloum+BORDER_BIAS]==IMAGE_BLACK)
                        {
                            RightLine[row]=cloum;      //记录右边
界点
                            flag_r=1;                 //flag 做无丢
线标记
                            break;
                        }
                    }
                }
            }
```

```
        }
        else if(path=='R')   //默认向右扫线
        {
            //先向右边扫线，寻找左边界点
            for(cloum=Mid;cloum+BORDER_BIAS<MT9V03X_W-1;cloum++)      //向右扫
            {
                //判断左边界点
                if(BinaryImage[row][cloum]==IMAGE_WHITE   && BinaryImage[row][cloum+BORDER_BIAS]==IMAGE_WHITE)  //找到白点（从赛道外扫到赛道内：黑-白）
                {
                    LeftLine[row]=cloum;      //记录左边界点（向右找到的是左边界点）
                    flag_l=1;                 //flag 做无丢线标记
                    break;
                }
            }
            //根据上面扫线的结果判断丢失的赛道是在左边还是右边从而决定继续向哪边扫线
            if(flag_l==1)    //找到了左边界（丢失的赛道在右边）
            {
                //继续向右寻找右边界
                for(;cloum+BORDER_BIAS<MT9V03X_W-1;cloum++) //继续向右扫
                {
```

```
                    //判断右边界点
                    if(BinaryImage[row][cloum]==IMAGE_BLACK
&& BinaryImage[row][cloum+BORDER_BIAS]==IMAGE_BLACK)      //找到黑
点（从赛道内扫到赛道外：白-黑）
                    {
                        RightLine[row]=cloum;    //记录左边界点
                        flag_r=1;                //flag 做无丢线标
记
                        break;
                    }
                }
            }
            else              //没有找到左边界（丢失的赛道在左边）
            {
                for(cloum=Mid;cloum-BORDER_BIAS>0;cloum--)
//向左扫
                {
                    //判断右边界点
                    if(BinaryImage[row][cloum]==IMAGE_WHITE
&& BinaryImage[row][cloum-BORDER_BIAS]==IMAGE_WHITE)
                    {
                        RightLine[row]=cloum;    //记录右边界点
（向左找到的是右边界点）
                        flag_r=1;                //flag 做无丢线标
记
                        break;
                    }
                }
```

```
                    if(flag_r==1)    //找到了右边界（丢失的赛道在左边）
                    {
                        //继续向左寻找左边界
                        for(;cloum-BORDER_BIAS>0;cloum--)    //继续向左扫
                        {
                            //判断左边界点
if(BinaryImage[row][cloum]==IMAGE_BLACK && BinaryImage[row][cloum-BORDER_BIAS]==IMAGE_BLACK)  //判断右边界点
                                {
                                    LeftLine[row]=cloum;       //记录左边界点
                                    flag_l=1;                  //flag 做无丢线标记
                                    break;
                                }
                        }
                    }
                }
            }
        }
        //在赛道中，正常扫线
        else
        {
            num=0;  //打断中线连续丢失，=0
```

```
                //左边扫线
                for(cloum=Mid;cloum-BORDER_BIAS>0;cloum--)
//向左扫
                {
                    if(BinaryImage[row][cloum]==IMAGE_BLACK &&
BinaryImage[row][cloum-BORDER_BIAS]==IMAGE_BLACK)  //判断左边界点
（BORDER_BIAS 防偶然因素）
                    {
                        LeftLine[row]=cloum;      //记录左边界点
                        flag_l=1;                 //flag 做无丢线标记
                        break;
                    }
                }
                //右边扫线
                for(cloum=Mid;cloum+BORDER_BIAS<MT9V03X_W-
1;cloum++)     //向右扫
                {
                    if(BinaryImage[row][cloum]==IMAGE_BLACK &&
BinaryImage[row][cloum+BORDER_BIAS]==IMAGE_BLACK)  //判断右边界点
（BORDER_BIAS 防偶然因素）
                    {
                        RightLine[row]=cloum;     //记录右边界点
                        flag_r=1;                 //flag 做无丢线标记
                        break;
                    }
                }
            }
```

```
            //1.29 晚上重新写的数据处理
            if(flag_l==0)     //左边界丢线
            {
                LeftLine[row]=0;                    //左边界点等于图像的左边界
                LostNum_LeftLine++;                 //左丢线数+1
            }
            if(flag_r==0)     //右边界丢线
            {
                RightLine[row]=MT9V03X_W-1; //右边界点等于图像的右边界
                LostNum_RightLine++;                //右丢线数+1
            }
            CentreLine[row]=(LeftLine[row]+RightLine[row])/2;      //计算中线
点
            //为下一次扫线做准备
            Mid=CentreLine[row];       //以上一次的中线值为下一次扫线的中
间点
            flag_l=0;                   //左边界丢线 flag 置 0
            flag_r=0;                   //右边界丢线 flag 置 0

        //          //LCD 绘制图像
        //          lcd_drawpoint(LeftLine[row],row,GREEN);
        //          lcd_drawpoint(CentreLine[row],row,RED);
        //          lcd_drawpoint(RightLine[row],row,BLUE);
        //          systick_delay_ms(STM0,50);
        }
    }
```