浙江省第二届大学生智能机器人创意大赛（主题三轮式自主格斗）

作品设计说明

目录

第一章 引言

大赛简介：

智能机器人格斗大赛（Intelligent Robot Fighting Competition，简称IRFC），IRFC将中国武术、竞技运动与人工智能、机器人等技术结合，融技术性、对抗性、挑战性、观赏性于一体，参赛队伍进行一对一、多对多等不同项目的角逐。

为进一步推进学生创新意识和创造能力培养，强化学生动手能力和工程实践能力，激励广大学生踊跃参加课外科技活动，有效推动新工科人才培养，促进校际交流。经浙江省大学生智能机器人创意竞赛组委会研究，决定举行浙江省第二届大学生智能机器人创意竞赛

其他：

准备比赛过程中收获到的成果离不开学校的大力支持以及老师的悉心教导，在此对学校以及老师表示衷心的感谢。

第二章 整体设计

2.1 设计框架概述：

本车利用主控提供的 16 个 AD 通道，与数字量红外传感器和模拟量红外测距相连，进行敌方车辆测距、台上边缘检测、台下识别。通过输入 pwm 波对电机进行转速控制，达到直行、转弯等效果。

调试时在自带 LCD 屏幕上显示红外数值。

以下为系统框架设计：

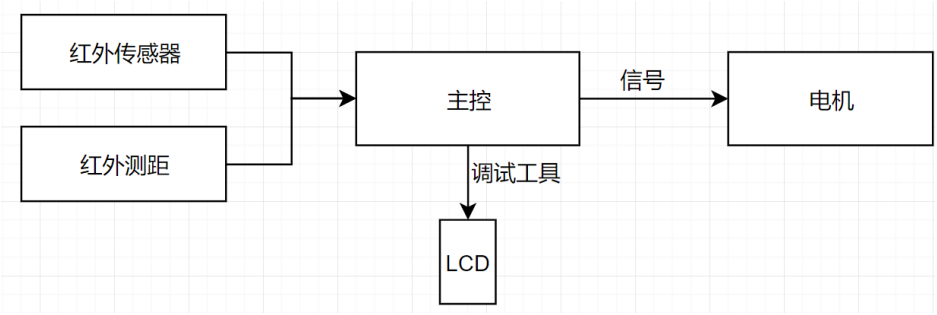

图 2.1 设计框架

2.2 车模整体布局

车模采用了 16 个传感器，八个红外传感器以及八个测距传感器。其中四个红外传感器用作边缘检测，让车在靠近台子边缘事能够及时停下，另外四个红外传感器在台上时检测敌方车辆，在台下时能够确定自身位置并进行调整。测距传感器在台上时检测敌方车辆，在台下能够确认自身状态早起台下，具体分布如图：

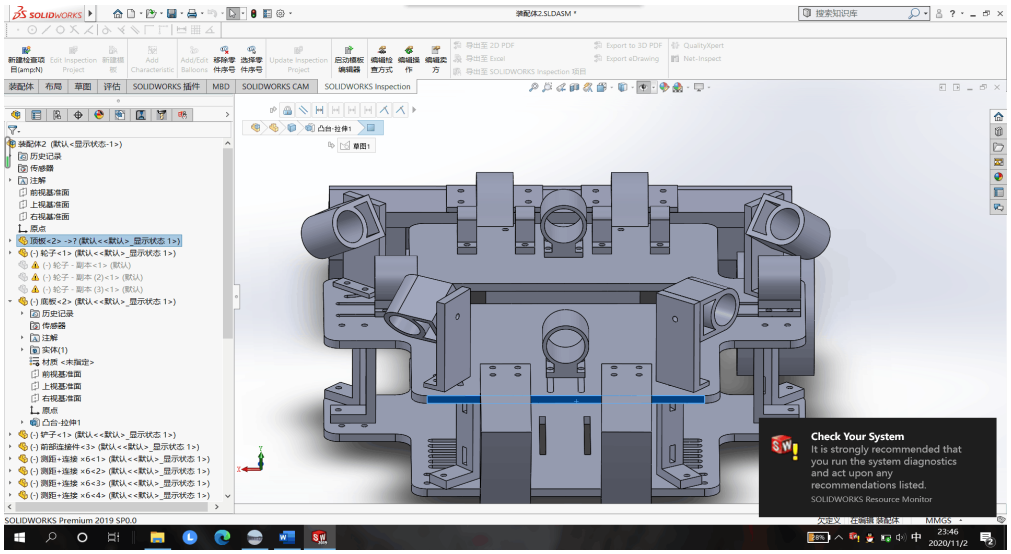

图 2.2 车模整体布局

2.3 主要模块介绍

车身模块主要分为：边缘检测模块；

台上台下检测模块；

寻敌攻击模块；

供电模块；

车身机械结构模块。

附实物图

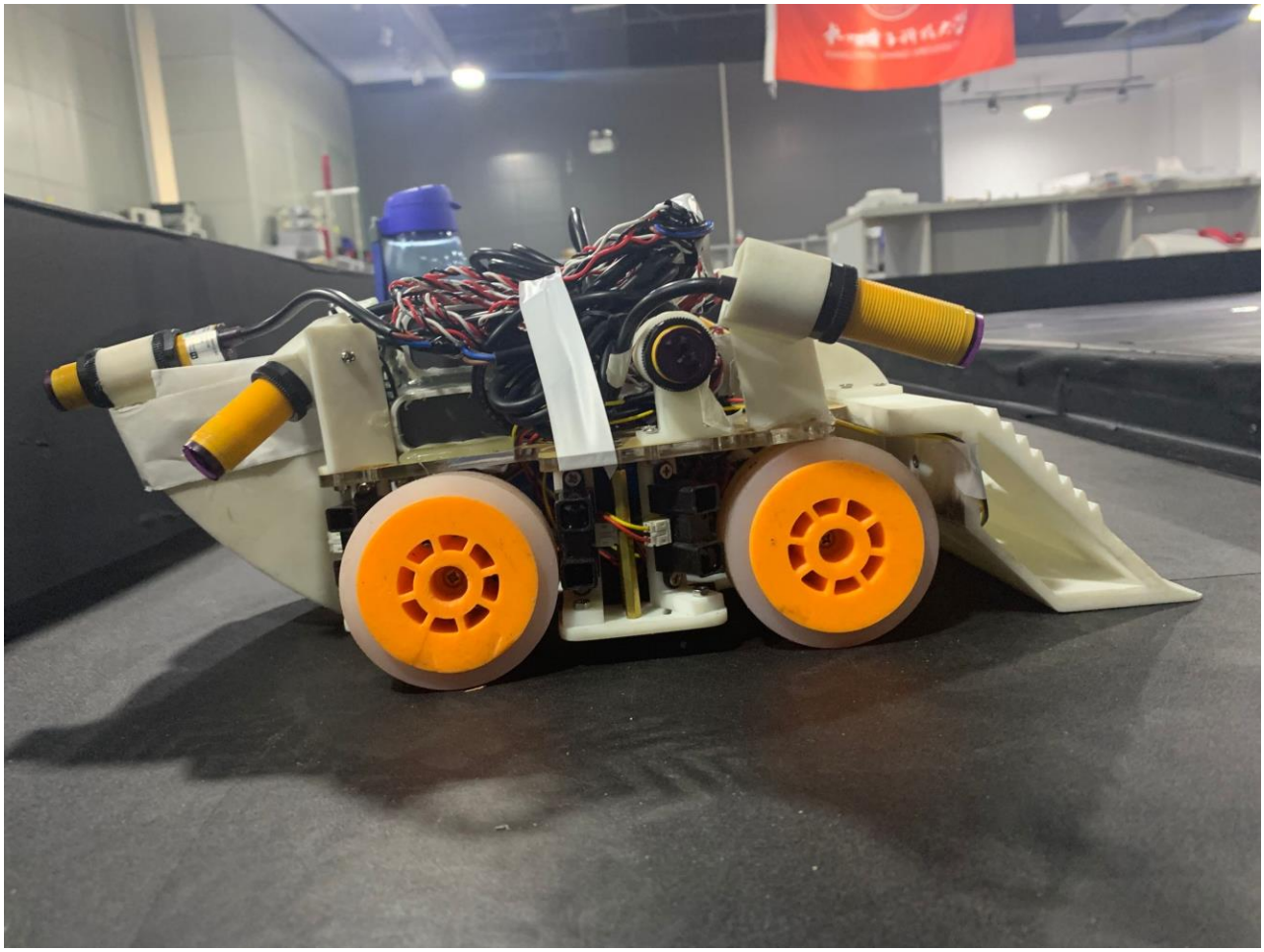

图 2.3 成品车模

第三章 软件设计

3．1 软件整体框架：

3.1.1 程序流程及简介

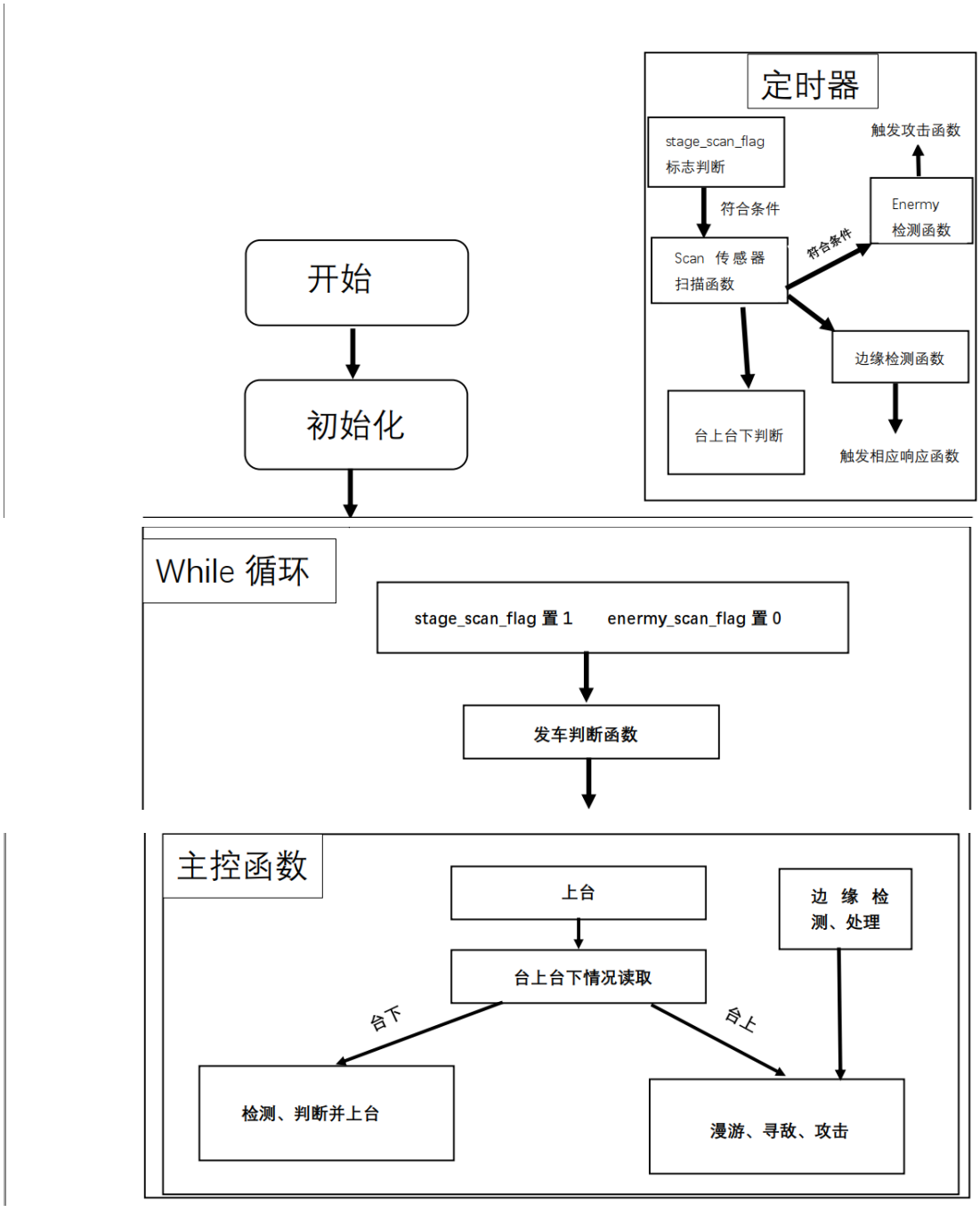

图 3.1 软件框架

基础应用标准组件的函数封装，例如控制车轮的 move 函数和 timer.c 文件中的定时器函数以及一些底层库文件。自己另设部分代码函数，例如发

车、边缘检测、寻觅敌方车辆、攻击、位置检测等函数。对整体框架做了一定的修改，将各个检测函数放入定时器内对车模进行实时监测，返回相应返回值用来触发检测或进入状态处理函数，对于每一个情况做出相应的判断和相应的行动。

3.1.2 软件改进：

将原有的 delay 算法改为实时监测的算法。运用函数返回值在定时器中控制各检测函数的触发与否。新增加刹车优化算法等等，使得车辆进一步的稳定。另外添加多种情况，添加卡尔曼滤波算法，加强稳定鲁棒性。

3.2 边缘检测算法：

我们采用在车的四角各安放一个红外光电传感器的策略，将传感器倾斜朝下检测，适当调整倾角及检测距离，使之能正确识别边缘。

```
void Edge()  //检测边缘
{
    edge_flag=0;

        //未检测到边缘
        if((AD9 < 1000)&&(AD10 < 1000)&&(AD11 < 1000)&&(AD12 < 1000))
        {
            edge_flag=0;
        }


        //左前检测到边缘
        else if((AD9 > 1000)&&(AD10 < 1000)&&(AD11 < 1000)&&(AD12 < 1000))
        {
            edge_flag=1;
        }

        //右前检测到边缘
        else if((AD9 < 1000)&&(AD10 > 1000)&&(AD11 < 1000)&&(AD12 < 1000))
        {
            edge_flag=2;
        }
```

```
        //右后检测到边缘
        else if((AD9 < 1000)&&(AD10 < 1000)&&(AD11 > 1000)&&(AD12 < 1000))
        {
            edge_flag=3;
        }

        //左后检测到边缘
        else if((AD9 < 1000)&&(AD10 < 1000)&&(AD11 < 1000)&&(AD12 > 1000))
        {
            edge_flag=4;
        }

        //前方两个检测到边缘
        else if((AD9 > 1000)&&(AD10 > 1000)&&(AD11 < 1000)&&(AD12 < 1000))
        {
            edge_flag=5;
        }

        //后方两个检测到边缘
        else if((AD9 < 1000)&&(AD10 < 1000)&&(AD11 > 1000)&&(AD12 > 1000))
        {
            edge_flag=6;
        }
```

```
        //左侧两个检测到边缘
        else if((AD9 > 1000)&&(AD10 < 1000)&&(AD11 < 1000)&&(AD12 > 1000))
        {
            edge_flag=7;
        }

        //右侧两个检测到边缘
        else if((AD9 < 1000)&&(AD10 > 1000)&&(AD11 > 1000)&&(AD12 < 1000))
        {
            edge_flag=8;
        }

        else if((AD9>1000)&&(AD10>1000)&&(AD11>1000)&&(AD12<1000))
        {
            edge_flag=9;
        }

        else if((AD10>1000)&&(AD11>1000)&&(AD12>1000)&&(AD9<1000))
        {
            edge_flag=10;
        }

        else if((AD11>1000)&&(AD12>1000)&&(AD9>1000)&&(AD10<1000))
        {
            edge_flag=11;
        }

        else if((AD12>1000)&&(AD9>1000)&&(AD10>1000)&&(AD11<1000))
        {
            edge_flag=12;
        }
}
```

图 3.2 边缘检测函数设计

设计思路：通过各种 if 语句，根据四角红外传感器的返回值，给出相应判断，判断车处于边缘的什么位置，车的哪个部位对着边缘，返回对应的 edge_flag 值，有主控函数来接受 edge_flag 的值做出相应判断控制车模的运动。

3.3 定时器核心算法：

我们采用将各检测函数放入定时器实时检测的方法，实时检测车模的状态，和触发相应的控制函数。

```
if (TIM_GetITStatus(TIM3, TIM_IT_Update) != RESET)
{
    if(stage_scan_flag)
    {
        scan();
    }

    nStage=Stage();
    Edge();
    if(!edge_flag&&enermy_scan_flag)
    {
        enermy();
    }


    TIM_ClearITPendingBit(TIM3, TIM_IT_Update);
```

图 3.3 定时器算法设计

Scan 函数作用：扫描，用来获取车上 16 个传感器的 AD 值。

Stage_scan_flag 作用：在扫描函数前加以判断，发车时开启扫描函数，上台时关闭扫描函数，防止其传入的值触发台上台下检测函数或边缘检测函数对上台造成干扰。

Stage 函数：通过返回值表明车模是在台上还是台下。

Edge：边缘检测函数。

Edge_flag 及 enermy_scan_flag：给寻敌函数加一限制，使车模在台上且没有到达边缘（即漫游）时才进行寻敌，防止发生误判。

3.4 主函数 main()

```
int main()
{
    /*初始化系统*/
    UP_System_Init();

  while(1)
    {
        stage_scan_flag=1;
        enermy_scan_flag=0;
        if(AD14<1000&&AD16<1000&&AD14!=0&&AD16!=0)
        {
            break;
        }
        UP_delay_ms(10);
    }
    main_control();

}
```

图 3.4 主函数设计

功能：发车检测

发车时 stage_scan_flag 置 1，开启 scan 扫描函数，使得发车检测能够进行。

3.5 主控函数 main_control()

3.5.1 上台函数

```
///////上台////////
void shangtai()
{
    stage_scan_flag=0;
    move(0,0);
    UP_delay_ms(100);

    move(-1023,-1023);//对准擂台
    UP_delay_ms(1200);

    move(0,0);
    UP_delay_ms(100);

    wheel_45_L();

    stage_scan_flag=1;
}
```

图 3.5.1 上台函数设计

stage_scan_flag 置 0，关闭 scan 扫描函数，先使车停下，防止上一状态是转弯影响上台。对准擂台后，四轮倒转速度拉满，猛冲 1200ms 后停下，使车模得以稳定。最后车模左转 45°，stage_scan_flag 置 1，开启 scan 扫描函数，使车在台上顺利进行漫游、寻敌、边缘检测等操作。

3.5.2 漫游函数

```
//漫游
void wander()
{
    //int count0=0;
    while((!edge_flag)&&(!fight_flag))
    {
//      count0++;

        move(500,500);
        UP_delay_ms(50);

//      if(count0>65535)
//      {
//          break;
//      }
    }
```

图 3.5.2 漫游函数设计

While 循环判断，判断条件：!edge_flag&&!fight_flag，即未检测到边缘也未检测到敌人时，使车以一定速度前进。

心得：利用少延时多触发循环的方法，有效避免了单片机处理延时函数无法对其他情况做出快速反应的问题。

其他：count 作用，防止程序死循环，但在实际测试中发现可行性不大。

3.5.3 边缘处理

```
if(edge_flag&&(!fight_flag))
{
        if(edge_flag==1)
        {

            move(-500,-500);
            UP_delay_ms(300);
            wheel_45_R();
        }
        if(edge_flag==2)
        {

            move(-500,-500);
            UP_delay_ms(250);
            wheel_45_L();
        }
        if(edge_flag==3)
        {
            move(500,500);
            UP_delay_ms(300);
            wheel_45_R();
        }
```

```
        if(edge_flag==4)
        {
            move(500,500);
            UP_delay_ms(300);
            wheel_45_L();
        }
        if(edge_flag==5)
        {
            move(-700,-700);
            UP_delay_ms(400);
            wheel_90_R();
        }
        if(edge_flag==6)
        {
            move(700,700);
            UP_delay_ms(400);
            wheel_90_R();
        }
        if(edge_flag==7)
        {
            wheel_90_R();
            move(600,600);
            UP_delay_ms(300);

        }
```

```
        if(edge_flag==8)
        {
            wheel_90_L();
            move(600,600);
            UP_delay_ms(250);
        }
        if(edge_flag==9)
        {
            move(-700,-700);
            UP_delay_ms(100);
            wheel_90_L();
            move(300,300);
            UP_delay_ms(100);
        }
        if(edge_flag==10)
        {
            move(700,700);
            UP_delay_ms(100);
            wheel_90_L();
            move(300,300);
            UP_delay_ms(100);
        }

        if(edge_flag==11)
        {
            move(700,700);
            UP_delay_ms(100);
            wheel_90_R();
            move(300,300);
            UP_delay_ms(100);
        }
        if(edge_flag==12)
        {
            move(-700,-700);
            UP_delay_ms(100);
            wheel_90_R();
            move(300,300);
            UP_delay_ms(100);
        }
}
```

图 3.5.3.1 边缘处理

通过边缘处理函数返回的 edge_flag 的值，进入各个边缘处理函数，对车的边缘情况进行处理。

其他：边缘处理和攻击间干扰的解决方案

```
if(edge_flag&&fight_flag)
{
    move(-600,-600);
    UP_delay_ms(50);
}
```

图 3.5.3.2 解决方案

攻击时触发边缘，优先处理边缘，使车后退，防掉台，再进行其他操作。

3.5.4 攻击处理

```
if(fight_flag&&(!edge_flag))
{
    if(fight_flag==1)
    {
        move(0,0);
        move(-900,-900);
        UP_delay_ms(150);
        move(0,0);
        wheel_45_L();
        fight();
    }
    if(fight_flag==2)
    {
        move(0,0);
        move(-900,-900);
        UP_delay_ms(150);
        move(0,0);
        wheel_45_R();
        fight();
    }
    if(fight_flag==3)
    {
        fight();
    }
```

图 3.5.4 攻击函数

通过寻敌扫描返回的 fight_flag 值，进入各个情况，触发相应函数，对前、后、左、右、左前、右前等方向出现的敌人进行各种不同的反应。

3.5.5 寻敌函数

```
//检测敌人
void enermy()
{
    if(AD1>1100)//左前有敌人
    {
        fight_flag=1;
    }
    else if(AD4>1100)//右前有敌人
    {
        fight_flag=2;
    }
    else if((AD7+AD8)>2300||AD13<1000)//前有敌人
    {
        fight_flag=3;
    }
    else if(AD2>1200||AD16<1000)//左中有敌人
     {
        fight_flag=4;
     }
     else if(AD5>1200||AD14<1000)//右中有敌人
     {
        fight_flag=5;
     }
```

图 3.5.5 寻敌函数

通过前后左右红外光电传感器，左前左中右前右中测距传感器传入的 AD 值，对四周敌人的方位进行有效的判断，返回相应的 fight_flag 值，在攻击函数中触发相应操作。

3.5.6 台上台下检测

```
//检测是否在台上
int Stage()
{
        if( ( ((AD1>800)||(AD2>800)) && ((AD4>800)||(AD5>800)) ) || ( ((AD7>800)||(AD8>800)) && ((AD3>800)||(AD6>800
        {
            stage_flag=0;
            return 0;  //在台下
        }
        else
        {
            stage_flag=1;
            return 1;   //在台上
        }

}
```

图 3.5.6 台上台下检测函数

通过车四周靠下的测距传感器判断是否在台下，若右两侧同时检测到较近物体，则说明车在台下，反之则车在台上。并返回相应判断标志值。

3.5.7 软件算法改进

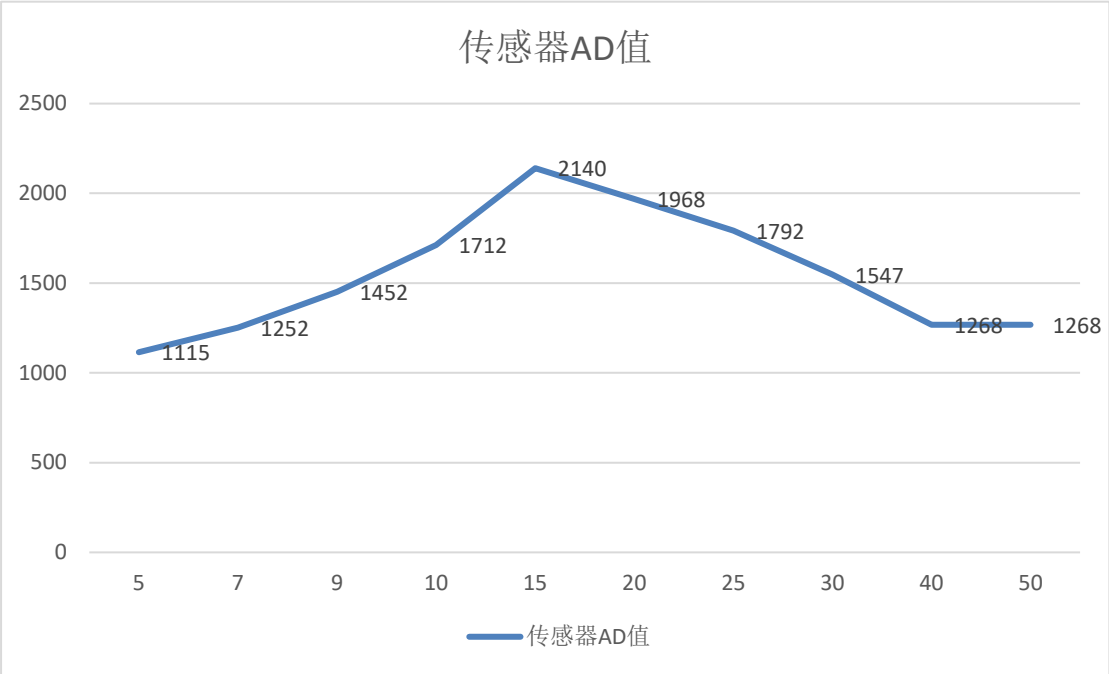

图 3.5.7.1 传感器数值图

实测发现，车模上安装的测距传感器数值在较近距离检测到物体时会急剧减小，在离传感器 15cm 左右得到的 AD 值最大，给我们的调试带来了很大的困难，多次实验，最后选用卡尔曼滤波算法，有效降低了噪声值，增加了平滑度。

```
#define KALMAN_Q 0.08   //过程噪声0.01  0.01
#define KALMAN_R 0.32   //测量噪声0.5   0.3

float KalmanFilter(const float ResrcData,float ProcessNiose_Q,float MeasureNoise_R)
{

    float R = MeasureNoise_R;
    float Q = ProcessNiose_Q;

    static float x_last;
    float x_mid = x_last;
    float x_now;

    static float p_last;
    float p_mid ;
    float p_now;

    float kg;

    x_mid=x_last;                       //x_last=x(k-1|k-1),x_mid=x(k|k-1)
    p_mid=p_last+Q;                     //p_mid=p(k|k-1),p_last=p(k-1|k-1),Q=噪声

    /*
     *  卡尔曼滤波的五个重要公式
     */
    kg=p_mid/(p_mid+R);                 //kg为kalman filter, R 为噪声
    x_now= (1-kg) * x_mid+kg * ResrcData;   //估计出的最优值
    p_now= (1-kg) * p_mid;                  //最优值对应的covariance
    p_last = p_now;                     //更新covariance 值
    x_last = x_now;                     //更新系统状态值

    return x_now;

}
```

图 3.5.7.2 卡尔曼滤波算法

第四章 车模机械结构设计

4.1 后部零件

通过对比赛台的测量以及对车整体机械结构的计算，设计出的车上台零件如图所示，45°的弧度设置使车辆能够更为轻松的上台

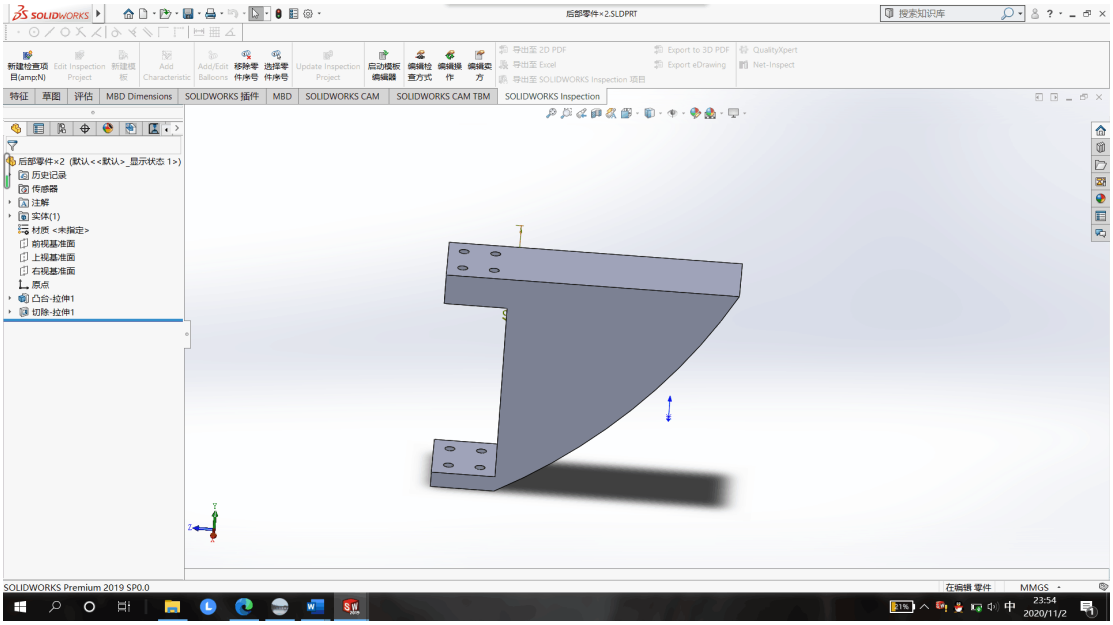

图 4.1 后部零件

4.2 顶部边缘检测红外

考虑到在调试车辆时顶部边缘检测红外测量器的位置角度需要反复改变，所以设计出的边缘检测红外有两个零件组成，可以 180° 自由调整前后探测位置，并且底部开两孔，配合顶板上的一孔一圆弧可以在 90° 自由调整左右探测位置的同时保证边缘检测红外的稳固。

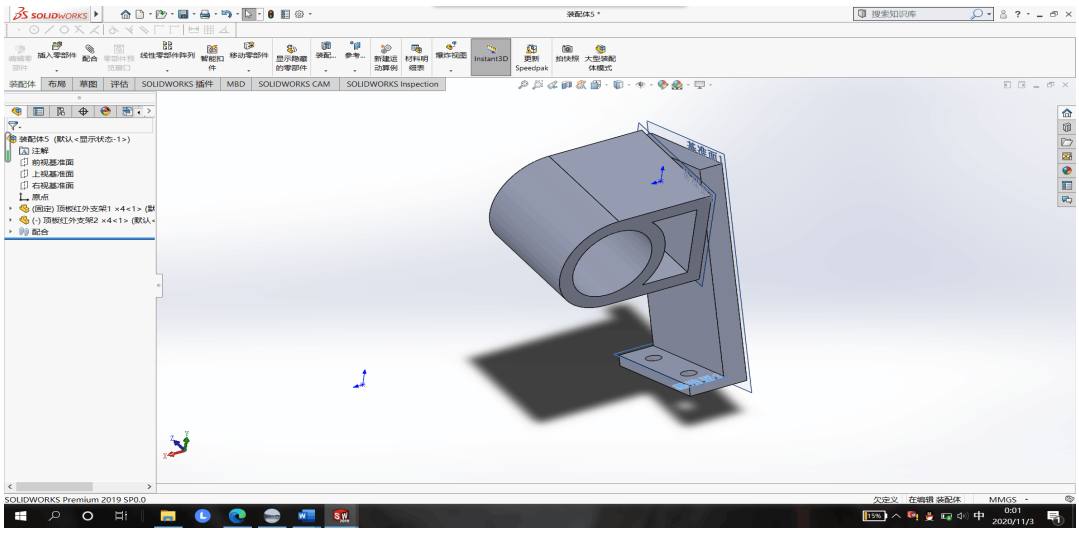

图 4.2 顶部边缘检测红外零件

4.3 铲子

铲子通过对车高度以及轮子直径精确的测量，做到了完美贴地，铲子上的凹凸能让敌方车辆的轮子上到我方的铲子时难以摆脱，铲子上的两个测距用以识别敌人并且判断在比赛场上的位置。铲子中空的设计使得测距能够安装在铲子中，并且起到减重的效果，使得重心后移，让上台更为轻松。

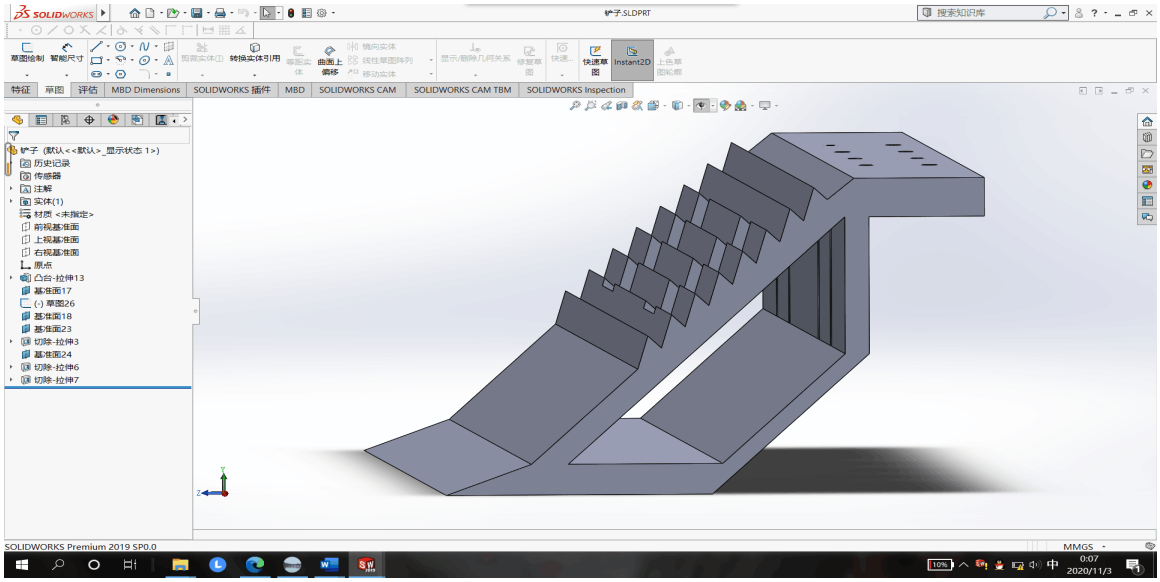

图 4.3.1 铲子

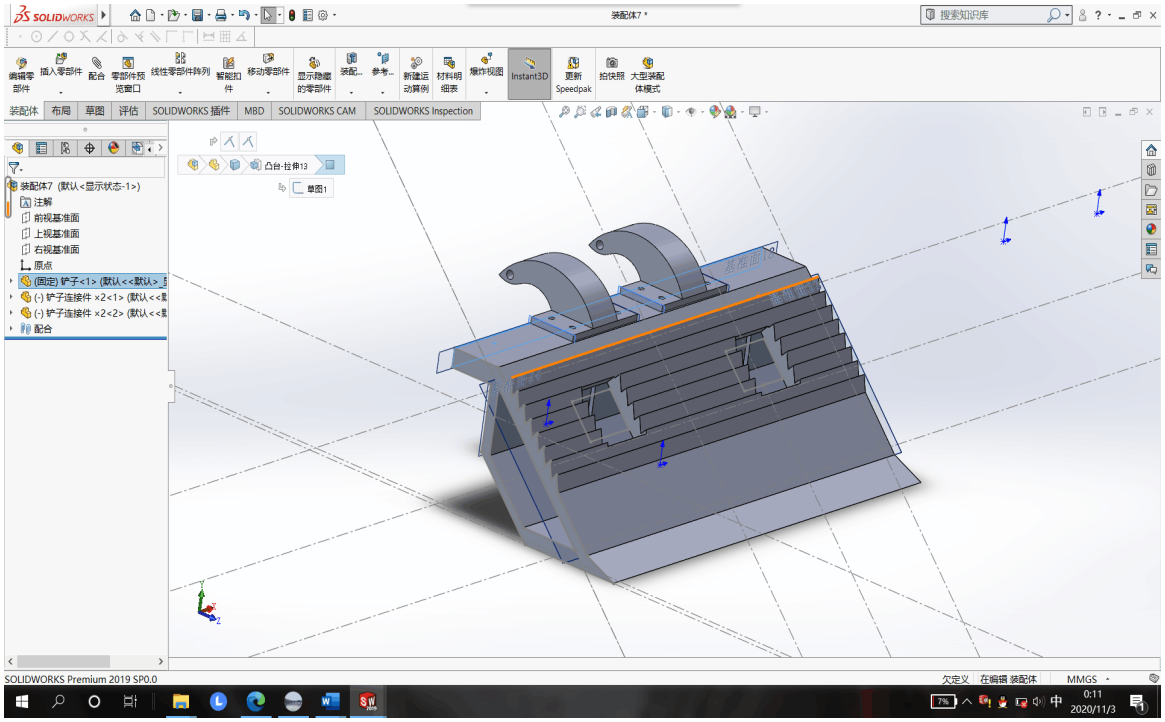

图 4.3.2 铲子

第五章 问题及解决方案

1 车无法上坡问题

问题原因 1:车在上坡过程中卡在台阶上,经多次尝试发现卡住过程中车前轮，即在台下的轮子离地悬空，后轮也处于半空中，所以轮子无法给予车动力，由此车卡住了。

解决方案:将前轮前移，使后轮悬空时，前轮仍着地，给予车动力，问题解决。

问题原因 2：车在上车过程中车冲的时间远远短于程序设计时间，冲到一半就断电。

解决方案：反复实验发现传感器的线离电池开关太近，上台时车身倾斜导致传感器的线碰到开关使得开关断开。将传感器的线固定在离开关较远位置，问题解决。

问题原因 3：车上台过程中猛冲后仍掉下台翻车。

解决方案：实验发现车身重心靠前，导致车上台后车尾不够重，重心不在台上导致上不了台。将车重心后移，车尾增重，问题解决。

2 进入攻击状态时边缘检测失效

问题原因：发现在执行延时函数时，无法从中断中读取红外值。而攻击函数大部分时间都在实现延时，所以在攻击时边缘检测失效

解决方案：减小攻击延时，反复检测，少延时多执行，反复进入判断函数。并在攻击函数中手动添加红外扫描与检测边缘后退程序。

3 车到达擂台拐角边缘出现打转现象导致下台

问题原因：边缘处理函数中，转弯延时过长，导致拐弯过大，使其他边缘检测红外传感器检测到边缘，触发另外的边缘处理函数，反复打转。

解决方案：在软件中减少转弯延时，将边缘检测处理函数化繁为简，并适当改变边缘检测函数倾角，反复实验，问题解决。

第六章 开发及调试工具

6.1 Keil

Keil MDK-ARM 是美国 Keil 软件公司（现已被 ARM 公司收购）出品的支持 ARM 微控制器的一款 IDE（集成开发环境）。MDK-ARM 包含了工业标准的 Keil C 编译器、宏汇编器、调试器、实时内核等组件。具有 ARM C/C++编译工具链，完美支持 Cortex-M、Cortex-R4、ARM7 和 ARM9 系列器件，包含世界上品牌的芯片。比如：ST、Atmel、Freescale、NXP、TI 等众多大公司微控制器芯片。

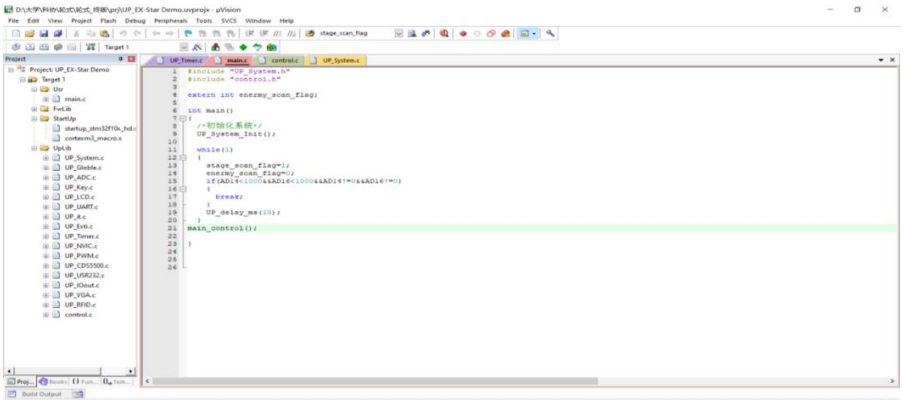

图 6.1 Keil 界面

6.2 SolidWorks

SolidWorks 是达索系统（Dassault Systemes ）下的子公司，专门负责研发与销售机械设计软件的视窗产品，公司总部位于美国马萨诸塞州。达索公司是负责系统性的软件供应，并为制造厂商提供具有 Internet 整合能力的支援服务。该集团提供涵盖整个产品生命周期的系统，包括设计、工程、制造和产品数据管理等各个领域中的最佳软件系统，著名的 CATIAV5 就出自该公司之手，目前达索的 CAD 产品市场占有率居世界前列

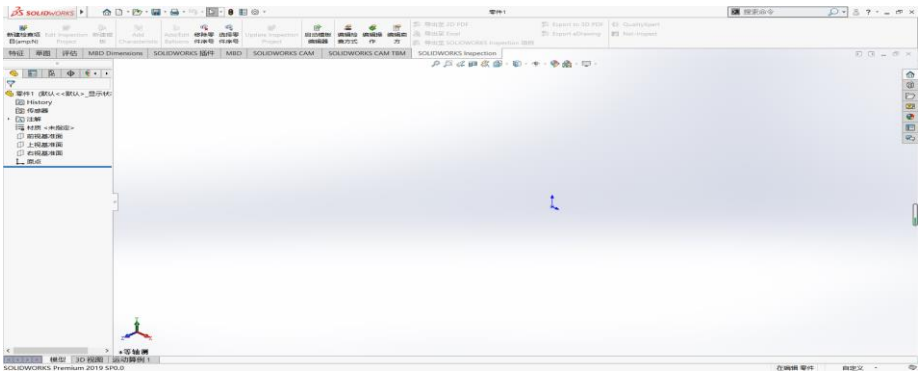

图 6.2SolidWorks 界面

第七章 主要参数说明

项目	参数
车功能用途描述	该车经调试，能够满足定位敌车（障碍物），边缘检测等功能。
车几何尺寸(长 X 宽 X 高)/cm	27.5*27.5
车模重量(带有电池)	3.38kg
传感器种类及个数	红外传感器 8 个 红外测距 8 个
电机个数	4
赛道信息检测频率	72HZ

第八章 总结

从组队参加轮式比赛，到软件代码构建与硬件车模构建和组装，再到调试轮式车与准备比赛，其中遇到过许许多多濒临崩溃或令人骄傲的瞬间，报告之中，区区百字，难以一言蔽之。但是，我们仍然希望这份承载着我们初心与努力的报告，能给予他人启发与帮助。

我们在调车过程中，发现了测距的不稳定性，大约 15cm 以内的测距属于失效状态，并不会随着距离增加而减小数值，这给调车带来了很大的难度。于是我们采用了卡尔曼滤波算法增加平滑度，再用红外传感与测距配合的方法，近距离测量尽量使用红外，尽最大努力减速测距的影响。我们相信轮式的车模构造和代码编写还有许许多多改善之处，也希望能给予我们宝贵的改正意见。士不可以不弘毅，任重而道远。几个月的竞赛经历，让我们明白了远大理想和坚持不懈的重要性，未来的路还很长，竞赛经历将成为我们人生中宝贵的财富。