摘要

本系统以 stm32f407vgt6 作为飞行控制核心，瑞萨 R5f100LEA 作为图像处理及导航核心设计并制作的一个四旋翼自主飞行器，实现了一键起飞，定高定点，简单的循迹，吸附重物等功能。系统采用运动处理模块 MPU6050 获得角速度和加速度数据；使用 OV7670 摄像头实时获取场地图像数据，R5f100LEA 对数据进行处理以检测出黑色点线，辅以超声波数据实现定点，循迹等功能；使用直流吸盘式电磁铁吸取和投放铁块。

关键字：stm32f407vgt6　R5f100LEA　图像识别

一：系统方案

1.1 飞行器动力系统的比较与选择

方案一：采用 X2212 KV980 无刷电机和电调作为飞行器的动力系统。无刷电机具有转速快，升力大，功率高，发热低，和有同等功率有刷电机相比体积更小的特点。是最适合做四旋翼飞行器的电机之一。但无刷电机转速快，在调试时需要很小心，容易伤到人。

方案二：采用空心杯电机和场效应管作为飞行器的动力系统。空心杯电机能量转换效率高，起动、制动迅速，响应极快，在高速运转状态下，可以方便地对转速进行灵敏的调节。适合做小型四旋翼飞行器的电机。但这样的系统升力不大，载重能力差。

虑到题目中的要求，需携带超声波，MPU6050，摄像头等传感器，需要有较强的携重能力，所以选择方案一。

1.2 飞行器循迹系统的比较与选择

方案一：采用激光传感器整列。利用激光对管组成传感器整列，激光发光管发出激光光线，光芒接收管接收地面的反射光。接收管接收的反射光强度随地面反射物的颜色不同而变化，传感器输出的电压值也随之变化。把每个激光传感器输出的信号都化成数字信号，根据激光管的排列，即可识别出黑线的位置。通过多个光电管的排列，经过算法处理，即可知道当前位置及黑线轨迹。该方案结论明显，外围电路简单。但需要多对激光管，重量大，实际有效距离较近，且飞行器在飞行过程中震动较大，高度变化明显，会对结果产生影响。

方案二：采用摄像头进行图像采集，将采集到的数据送与 MUC 进行图像处理，识别场地的黑点，黑线。从而进行黑线循迹，辅以超声波数据进行定点定高等功能。该方案算法处理复杂，难度较高，但可以得到较为理想的效果。更加适合用于飞行器上。

综上所述，采用方案二。

1.3 摄像头选择

方案一：采用 OV7620 的摄像头模块，OV7620 分辨率达到 640*480，传输速率可以达到 30 帧；数据格式包括 YUV、YCrCb、RGB 三种，能够满足一般图像采集系统的要求。尤其是 YUV 输出，对于白底黑线的场地来说是非常适合的。

方案二：采用 OV7620+FIFO 摄像头模块，相比于方案一，该方案由于增加了 FIFO，更能达到速率的匹配，同时当 MCU 执行任务是能保证接收到完整的帧，避免在执行中断时丢失摄像头数据。且对于该方案瑞萨提供解决方案，官方提供了相关的历程，但现在市面上很少有该摄像头模块，没有获得渠道。且由于时间较为紧迫，也基本排除了自己设计该模块的方案。

方案三：采用 OV7670+FIFO 摄像头模块。该摄像头模块是市面上最常见的摄像头模块之一。其分辨率为 640*480，传输速率 30 帧，其默认输出为 RGB 彩色图像，但同样可设定为 YUV 输出。

考虑到比赛时间的紧迫，且团队第一次使用瑞萨的产品，所以选择了方案二，但由于没有途径获得该模块，考虑到方案三与方案二虽有区别，但对于 MCU 直接操作 FIFO 的方法是相同的，可借鉴与官方例程，且可对 OV7670 进行超频，传输速率可达到 50 帧，所以选择方案三。

1.4 摄像头镜头选择

方案一：选择普通摄像头镜头。可视角度约为 30°左右。

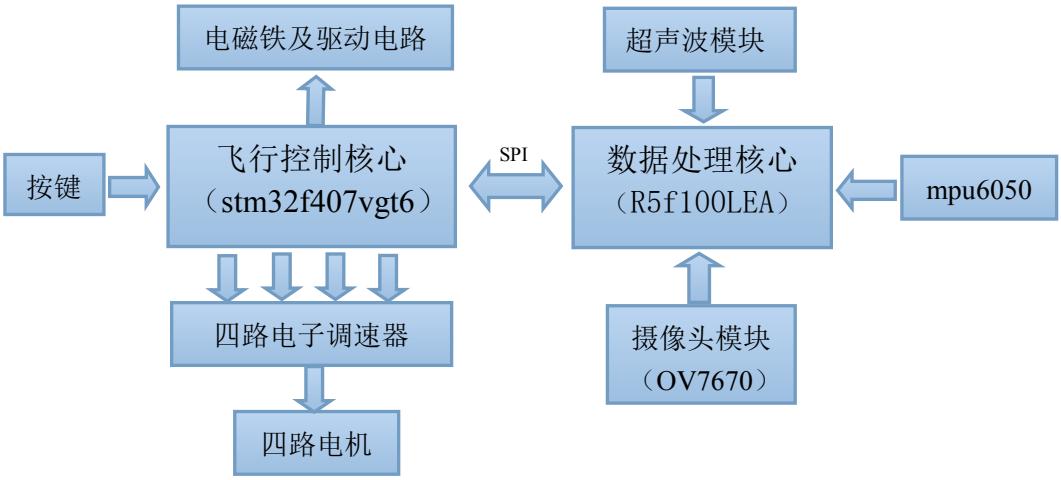
方案二：选择超广角镜头，可视角度可达 170°。

团队开始并没有考虑到镜头的选择，在调试过程中发现当飞行器于 A 点起飞时，由于镜头与 A 点距离太近，采集到的图像大部分为黑色，不利于飞行器的原点稳定起飞；且在寻线过程中，若飞行器稍稍偏差角度，摄像头便无法检测到黑线，使得系统的可靠性，稳定性低。而超广角镜头可视角度大，可以有效的避免这样的情况。

综上所述，选择方案二。

1.5 系统总体方案设计与论证

系统实现框图如图一：



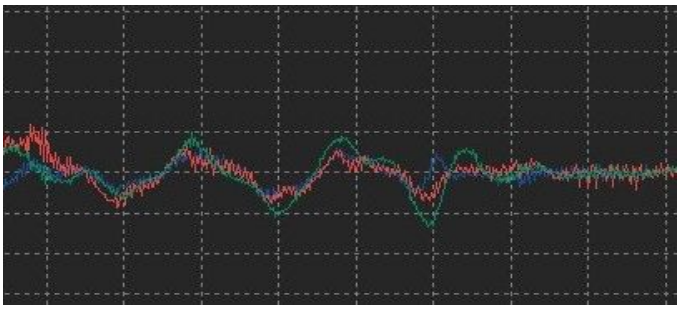
图一：系统实现框图

R5f100LEA 将读取 mpu6050 中的数据并进行滤波，姿态解算，将解算的欧拉角数据与期望值的差值送入 PID 控制器 1，读取超声波模块提供的飞行器距离地面的高度，将实时高度与期望值的差值送入 PID 控制器 2，读取摄像头数据，对其进行二值化后，送入 PID 控制器 3,；stm32f407vgt6 将由 R5f100LEA 获得的值，通过电调控制电机转速，改变升力。姿态控制与升力控制相结合，使飞行器能够完成三维空间的各种运动。

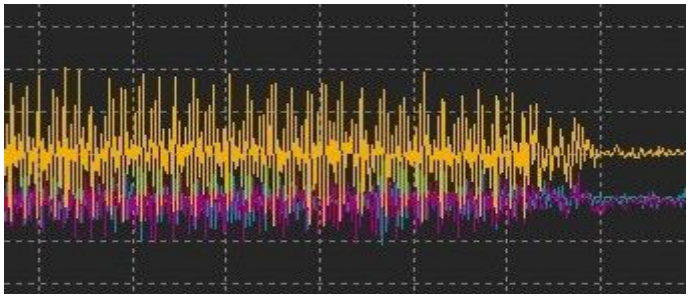
二：设计与论证

2.1 数据滤波算法

对于 MPU6050 的原始数据，我们用滑动窗口数据滤波进行了初次滤波。该滤波方法可以平滑图像，减小极限值的干扰。有速度快，算法简单等特点。



图二：滤波后的陀螺仪波形



图三：滤波后的加速度波形

2.2 姿态解算过程

2.2.1 初始姿态四元数

初始姿态四元数(q0,q1,q2,q3)=(1,0,0,0) 命名为 A（四元数初值）

2.2.2 四元数姿态更新

将经过滑动窗口数据滤波的陀螺仪数据进行以下运算得到四元数：

$$q_0 = q_0 + (-q_1\omega_{xB} - q_2\omega_{yB} - q_3\omega_{zB})\frac{\Delta t}{2}$$

$$q_1 = q_1 + (q_0\omega_{xB} + q_2\omega_{zB} - q_3\omega_{yB})\frac{\Delta t}{2}$$

$$q_2 = q_2 + (q_0\omega_{yB} - q_1\omega_{zB} + q_3\omega_{xB})\frac{\Delta t}{2}$$

$$q_3 = q_3 + (q_0\omega_{zB} - q_1\omega_{yB} + q_2\omega_{xB})\frac{\Delta t}{2}$$

2.2.3 互补滤波器数据融合

由陀螺得到的四元数只能保证短期的精确度，加速度计受运动加速度影响大，特别是受飞行器机架的振动。而陀螺仪则受外部影响弱，稳定性好，但输出量为角速度，需要积分才能得到姿态，会有积分误差。所以将加速度误差构造成纠正旋转，叠加到陀螺测出的角增量上，实现高效的数据融合。融合运算如下：

$$\varphi = \arctan(\frac{2(q_1q_2 + q_0q_3)}{q_0^2 + q_1^2 - q_2^2 - q_3^2})$$

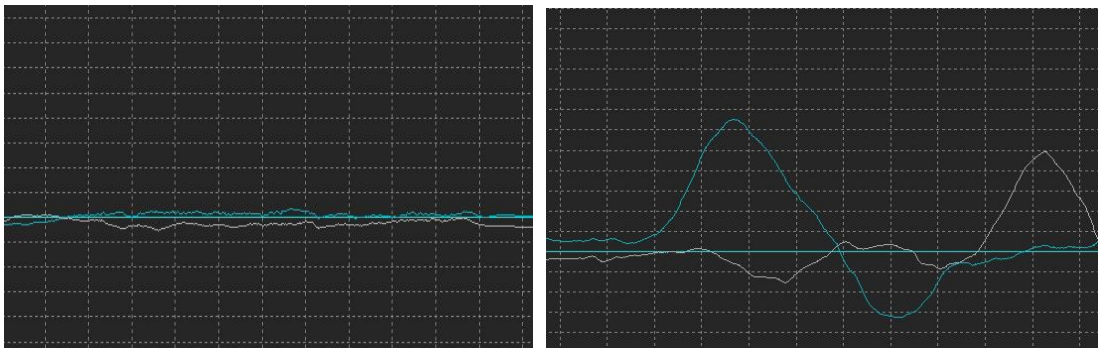
$$\theta = \arcsin(2(q_0q_2 - q_1q_3))$$

$$\psi = \arctan(\frac{2(q_0q_1 + q_2q_3)}{q_0^2 - q_1^2 - q_2^2 + q_3^2})$$

$$\theta_e = \theta_g + k(\theta_\alpha - \theta_g)$$

$$\varphi_e = \phi_g + k(\phi_\alpha - \phi_g)$$

$$\psi_e = \theta_g + k(\psi_\alpha - \psi_g)$$



图四：互补滤波后的波形

2.3 摄像头数据处理

1.配置 ov7670:配置 OV7670 为 YUV 4:2:2 输出。在获取到数据后，舍弃 UV 值，剩下的 Y 值即为灰度值。

2.取每帧图像灰度值的中值作为阈值将图像进行二值化。

3.在二值化后的图像中找原点，根据超声波测得的高度数据配置圆点的半径，寻找匹配的大小的圆。

4.在二值化后的图像中寻找直线，对找到的直线进行分类：

- 直角
- 单直线
- 圆点加直线
- 直线十字交叉

根据摄像头处理后的结果结合路线图进行导航。

2.4 PID

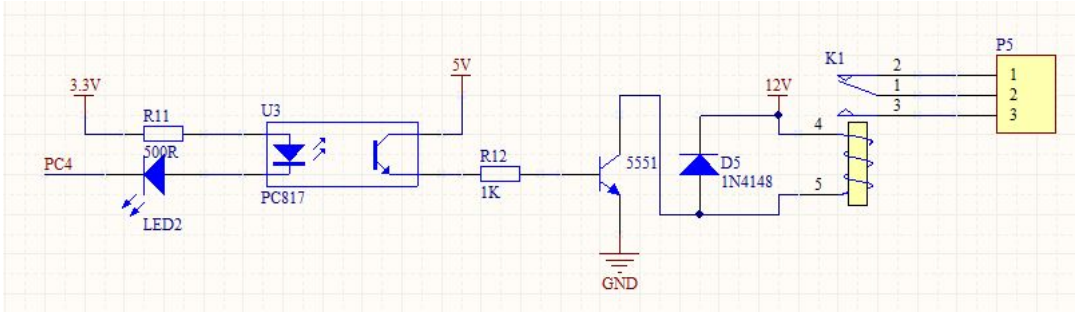
将上述过程中的俯仰，横滚，航向，超声波数据，摄像头二值化后的数据 5 个控制量分别输入 5 个独立的 PID 控制器，我们可以得到三个 PID 输出：pid_roll，pid_pitch，pid_yaw，pid_high，将这 5 个输出量做简单的线性运算输出给电机，每个电机的对应的控制量都是三个控制器输出的叠加，叠加量的正负与电机位置相关姿态控制与升力控制相结合，使飞行器能够完成三维空间的各种运动。

三：电路与程序设计

3.1 硬件电路设计

3.1.1 stm32f407 飞控原理图见附录 1

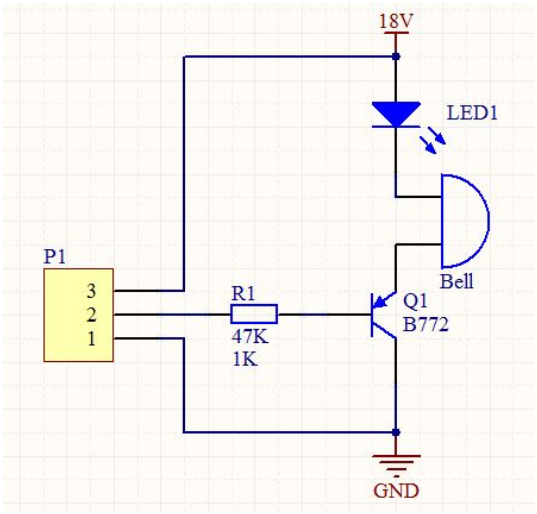
3.1.2 直流吸盘式电磁铁驱动电路



图五：直流吸盘式电磁铁驱动电路

本团队选用 ELE-P20/15 型电磁铁，12V 供电，吸力 2.5kg。用继电器进行驱动。并使用光耦隔离，当 PC4 脚为低是，光耦导通，继电器吸附，电磁铁开始工作，反之停止工作。以小驱大。

3.1.3 示高线驱动及报警电路

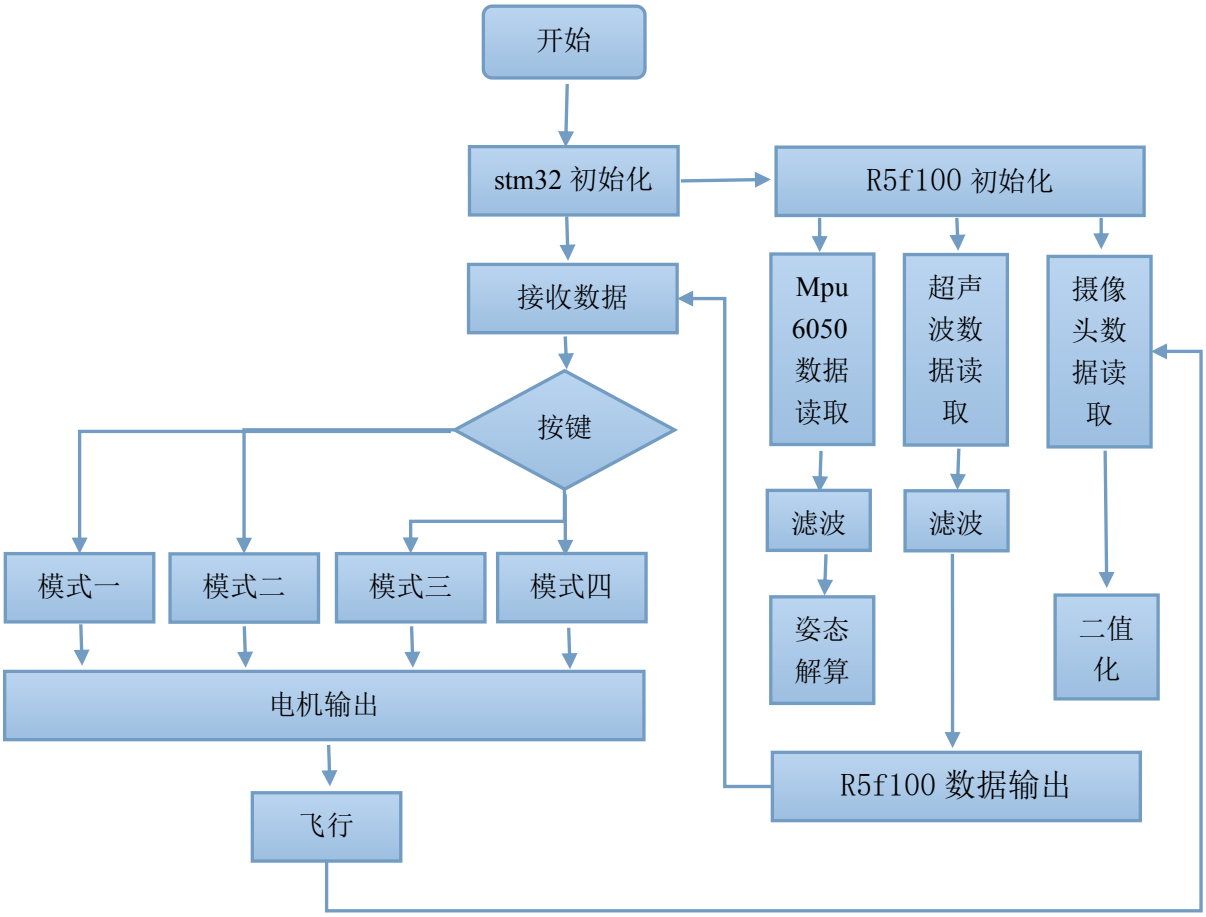


图六：示高线驱动及报警电路

采用 PEX-10FR 激光对射光电开关作为产生示高线设备，其有效距离为 10m。输出形式为 NPN 型。供电采用两节 9V 电池串联得到。当 P1 中 2 脚为低（激光未对齐）是，蜂鸣器响，灯亮。反之不响，灯灭。

3.2 程序流程设计（主函数，图像处理函数见附录 2）

程序流程图如下：



图七：程序流程图

四：测试方案与测试结果

4.1 测试仪器

1.白色 KT 板+黑色电工胶带模拟比赛场地；
2.联想笔记本电脑，串口无线收发模块一对；
3.匿名科创飞行器上位机；
4.DS1062E 示波器；
5.UT840 万用表。

4.2 测试方法

将一块串口无线与 stm32f407vgt6 飞控相连，另一块与笔记本电脑相连。两张建立连接后即可完成飞控与电脑上位机之间的无线通信。这样让调试过程更加的方便，也能够将飞行中的状态实时的发送到电脑上，可以更加直观的发现问题的所在。

4.3 测试结果

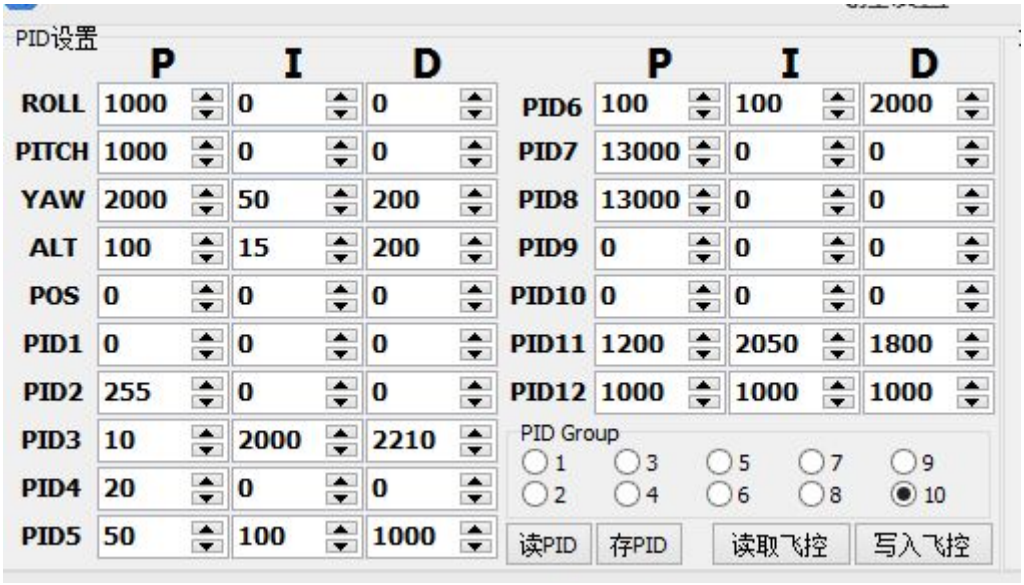
4.3.1 姿态角测试

利用串口无线模块将解算出的姿态角数据发回上位机，左右晃动并上下摇动机体，观察上位机数据曲线。

图八：波形输出图

图 3-1 中曲线为经过控制后的数据曲线，分布代表高度，P 值，I 值，D 值，合输出值，数值反应快，噪声小，且与改变飞行器方位状态一致，足以满足控制要求。

4.3.2 PID 值



图九：PID 最终值

4.3.3 模拟试飞调试

表一：A 到 B 飞行航拍测试

次数	飞行高度	飞行时间	航拍效果	是否落在 B 区	是否在 B 区中心
1	50cm	5s	拍摄抖动厉害	是	否
2	50cm	7s	轻微抖动	否	否
3	50cm	3s	轻微抖动	时	碰触到

4	50cm	3s	轻微抖动	是	否
5	50cm	4s	轻微抖动	是	碰触到

第一次测试后航拍图像抖动厉害，进过增加减震片等措施，得到改善。循迹效果不是太过理想，很到停到中心点。

表二：矩形循迹飞行测试

次数	飞行高度	飞行时间	矩形飞行是否成功
1	50cm	3	否
2	50cm	9	否
3	50cm	7	否
4	50cm	5	否

无法完成矩形的循迹，问题主要在于转角处的识别及飞行器具有的间歇性偏角的问题。

表三：穿过示高线投块返航测试（200g）

次数	h1	h2	是否从线间飞过	M1 投放位置	降落位置	时间
1	30cm	120cm	是	B 区内	A 区外	7s
2	40cm	110cm	是	碰触 B 区中心	A 区内	6s
3	50cm	100cm	是	B 区内	A 区内	7s
4	50cm	90cm	否			
5	50cm	90cm	否			

最小能够从间距 50cm 的示高线差间飞过。

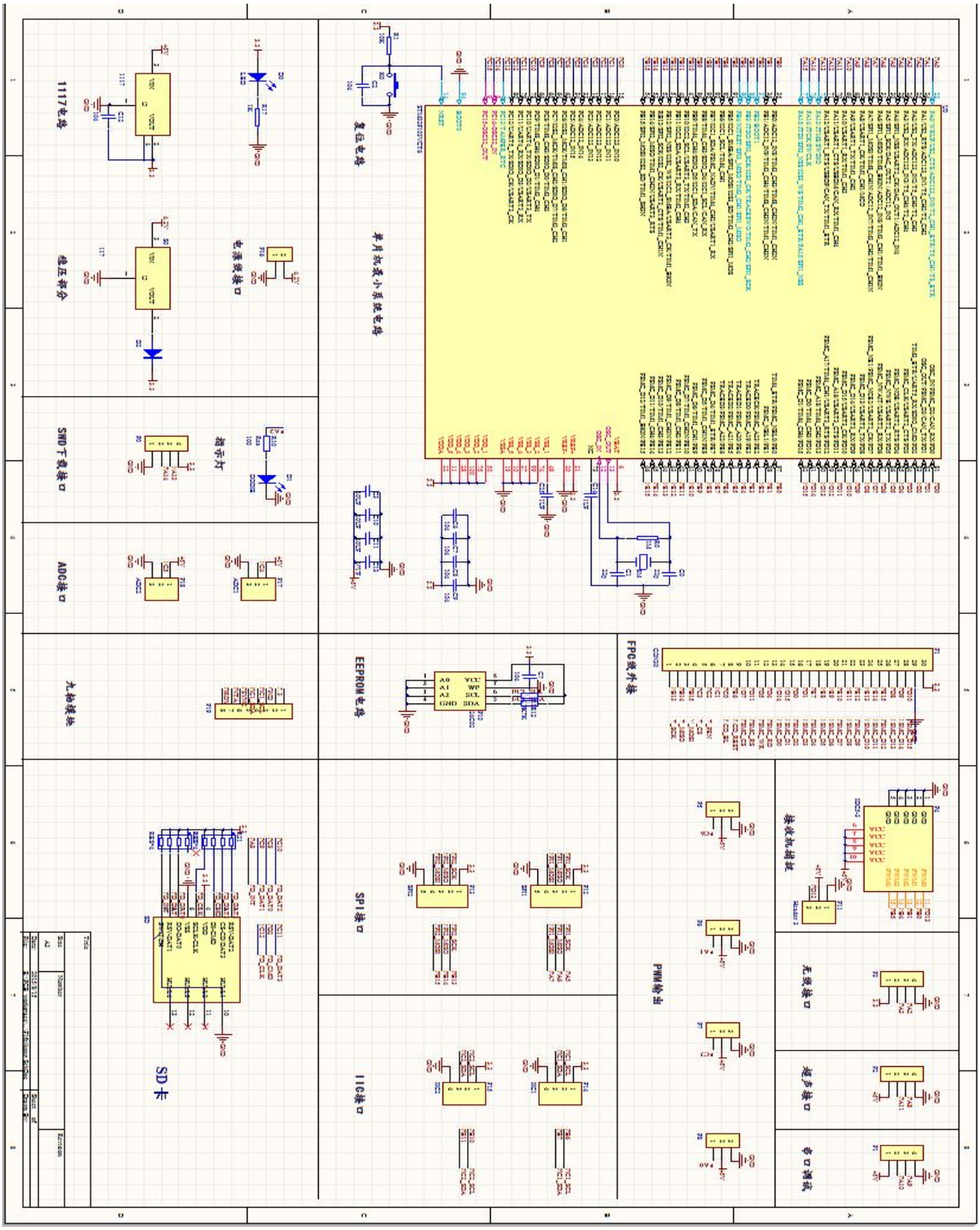
对于发挥部分二，由于时间问题无法对 B 区中的铁块进行精确识别并精准的降落在其上方，所以并未进行测试。

4.3.4 测试结果分析

综上所述，本团对完成了基本要求和发挥部分的部分要求，主要问题在于无法精确的沿着直线飞行，以及摄像头对图像，尤其是直角处的识别算法和飞行器转向的运动算法上还有很大的不足。

附录

附录 1：整机详细电路图



附录 2：主要程序源代码

////////////////////////////////图像识别////////////////////////////////////

```
#include "EasyTracer.h"
//从腐蚀中心向外腐蚀，得到新的腐蚀中心
static int Point_Corrode(unsigned int oldx, unsigned int oldy, const TARGET_CONDI *Condition, RESULT *Resu)
{
    unsigned int Xmin, Xmax, Ymin, Ymax, i, FailCount = 0;
    for (i = oldx; i > Condition->IMG_X; i--)
    {
        if (!Point_ColorMatch(i, oldy))
            FailCount++;
        if (FailCount > (((Condition->WIDTH_MIN + Condition->WIDTH_MAX) >> 2) >> ALLOW_FAIL_PER))
            break;
    }
    Xmin = i;
    FailCount = 0;

    for (i = oldx; i < Condition->IMG_X + Condition->IMG_W; i++)
    {
        if (!Point_ColorMatch(i, oldy))
            FailCount++;
        if (FailCount > (((Condition->WIDTH_MIN + Condition->WIDTH_MAX) >> 2) >> ALLOW_FAIL_PER))
            break;
    }
    Xmax = i;
    FailCount = 0;

    for (i = oldy; i > Condition->IMG_Y; i--)
    {
        if (!Point_ColorMatch(oldx, i))
            FailCount++;
        if (FailCount > (((Condition->HIGHT_MIN + Condition->HIGHT_MAX) >> 2) >> ALLOW_FAIL_PER))
            break;
    }
    Ymin = i;
    FailCount = 0;
```

```
    for (i = oldy; i < Condition->IMG_Y + Condition->IMG_H; i++)
    {
        if (!Point_ColorMatch(oldx, i))
            FailCount++;
        if (FailCount > (((Condition->HIGHT_MIN + Condition->HIGHT_MAX) >> 2) >>
ALLOW_FAIL_PER))
            break;
    }
    Ymax = i;
    FailCount = 0;

    Resu->x = (Xmin + Xmax) / 2;
    Resu->y = (Ymin + Ymax) / 2;
    Resu->w = Xmax - Xmin;
    Resu->h = Ymax - Ymin;

    //如果 左右.上下尺寸 符合查找标准 返回 (1)
    if (((Xmax - Xmin) > (Condition->WIDTH_MIN)) && ((Ymax - Ymin) >
(Condition->HIGHT_MIN)) && \
            ((Xmax - Xmin) < (Condition->WIDTH_MAX)) && ((Ymax - Ymin) <
(Condition->HIGHT_MAX)) )
        return 1;
    else
        return 0;
}
//搜索腐蚀中心
static int Point_SearchCentre(unsigned int *x, unsigned int *y, const TARGET_CONDI
*Condition, const SEARCH_AREA *Area)
{
        RESULT Result;
    unsigned int SpaceX, SpaceY, i, j, k, FailCount = 0;

    SpaceX = Condition->WIDTH_MIN / 2;//x 步长
    SpaceY = Condition->HIGHT_MIN / 2;//y 步长

    for (i = Area->Y_Start; i < Area->Y_End; i += SpaceY)//从 Y 起始位置到 终点位子
以 y 步长差
    {
        for (j = Area->X_Start; j < Area->X_End; j += SpaceX)//从 x 起始位置到 终点位
子 以 x 步长差
        {
            FailCount = 0;
            for (k = 0; k < SpaceX + SpaceY; k++)
```

```
                {
                    //以十字搜索
                    if (k < SpaceX)
                    {
                        if (!Point_ColorMatch(j + k, i + SpaceY / 2)) //如果匹配
                            FailCount++;//匹配次数
                    }
                    else
                    {
                        if (!Point_ColorMatch(j + SpaceX / 2, i + (k - SpaceX))) //如果不匹配
                            FailCount++;//不匹配次数
                    }
                    if (FailCount > ((SpaceX + SpaceY) >> ALLOW_FAIL_PER))//容错率
                        break;
                }
                if (k == SpaceX + SpaceY)
                {
                            if(!Point_Corrode(j + SpaceX / 2, i + SpaceY / 2, Condition,
&Result))
                                continue;
                    *x = j + SpaceX / 2;
                    *y = i + SpaceY / 2;
                    return 1;
                }
            }
        }
        return 0;
}

static SEARCH_AREA Area;// = {Condition->IMG_X, Condition->IMG_X +
Condition->IMG_W, Condition->IMG_Y, Condition->IMG_Y + Condition->IMG_H};

void Point_Trace_Init(const TARGET_CONDI *Condition)
{
                Area.X_Start = Condition->IMG_X ;
                Area.X_End   = Condition->IMG_X + Condition->IMG_W;
                Area.Y_Start = Condition->IMG_Y ;
                Area.Y_End   = Condition->IMG_Y + Condition->IMG_H;
}
int Point_Trace(const TARGET_CONDI *Condition, RESULT *Resu)
{
        unsigned int i;
        static unsigned int x0, y0, flag = 0;
```

```
//查找范围
RESULT Result;
if (flag == 0)//首次进入
{
    //从上次迭代地点查起
    if (Point_SearchCentre(&x0, &y0, Condition, &Area))//寻找腐蚀中心
        flag = 9;//已找到
    else//没找着
    {
        //改变查找范围 从头找起
        Area.X_Start = Condition->IMG_X ;
        Area.X_End   = Condition->IMG_X + Condition->IMG_W;
        Area.Y_Start = Condition->IMG_Y ;
        Area.Y_End   = Condition->IMG_Y + Condition->IMG_H;

        if (Point_SearchCentre(&x0, &y0, Condition, &Area))//再次寻找
        {
            flag = 0;
            return 0;//退出
        }
    }
}
    else if (flag == 1)
{
    if (Area.Y_Start > Area.Y_End)
    {
        flag = 0; return 0;
    }
                //Area.X_Start    =    Area.X_Start+Condition->WIDTH_MIN    /
2;;//Area.X_Start+((Result.w) >> 1);
    Area.Y_Start = Area.Y_Start+Condition->HIGHT_MIN / 2;
    Area.X_End   = Condition->IMG_X + Condition->IMG_W;
    Area.Y_End   = Condition->IMG_Y + Condition->IMG_H;
    if (Point_SearchCentre(&x0, &y0, Condition, &Area))//寻找腐蚀中心
        flag=9;//已找到
    else
            {
        flag = 0;
                    return 0;
                    }
}
Result.x = x0;
Result.y = y0;
```

```
    //多次迭代
    for (i = 0; i < ITERATE_NUM; i++)
        Point_Corrode(Result.x, Result.y, Condition, &Result);//从腐蚀中心向外腐蚀，得到新的腐蚀中心

    //从腐蚀中心向外腐蚀，得到新的腐蚀中心
    if (Point_Corrode(Result.x, Result.y, Condition, &Result))
    {
        x0 = Result.x;
        y0 = Result.y;
        Resu->x = Result.x;
        Resu->y = Result.y;
        Resu->w = Result.w;
        Resu->h = Result.h;
        flag = 9;

        Area.X_Start = Result.x - ((Result.w));
        Area.X_End   = Result.x + ((Result.w));
        Area.Y_Start = Result.y - ((Result.h));
        Area.Y_End   = Result.y + ((Result.h));
        return 1;
    }
    else
    {
        flag=1;
        return 0;
    }
}
//输入 RBG 返回灰度值
int Gray_Threshold_H;
int Gray_Threshold_L;
u16 RGB_To_Gray(u16 C16)
{
    u16 Gray;
    COLOR_RGB Rgb;
    Rgb.red     =       (unsigned char)((C16 & 0xf800) >> 8);
    Rgb.green =       (unsigned char)((C16 & 0x07e0) >> 3);
    Rgb.blue   =       (unsigned char)((C16 & 0x001f) << 3);
    Gray = (Rgb.red * 2449 + Rgb.green * 4809 + Rgb.blue * 934) >> 13;
    return Gray;
}
```