RM 2015 summer camp 第八组算法方案技术报告

一、 任务分析

夏令营任务要实现自主飞行机器人的设计，在 8*8m 的场地上实现自主飞行和自主抓球和取球。因此可以提取地面二维码建立全场坐标或者局部定位来实现 M100 的定位，此外还需要识别取球区、圆盘和九宫格。我们使用 opencv 开发。

二、 定位方案设计

以下方案单目基于摄像头装在飞机底部，向下观察场地的基础上。

1） 利用二维码实现全场定位

利用开源库 acruo 或者 apriltag 均可以得出二维码相对于摄像头的位置信息。其中我们实验了 acruo 和 apriltag，发现 apriltag 稳定性和可靠性高于 acruo。因此利用 apriltag 可以实现局部定位。然后通过局部实时定位就可以得出全场定位。

具体方案设想为：利用一个起始 apriltag 标定原点，M100 起飞后提取出飞机下方的二维码坐标，通过原点到飞机坐标系和飞机坐标系到地面的转换，实时求出飞机视野中所有可见二维码的坐标，与地图里面的二维码比较后标记这些二维码，同时由已经标记的二维码可以求出飞机的坐标和未标记的二维码的坐标，从而实现全场定位。

此方案的优点是比较稳定可靠，apriltag 开源库对适应性做了全面的考虑和处理，可以利用它直接得出坐标。难点在于二维码的锁定标记管理、查表搜索和坐标系的转换。

2） 利用巡线与 guidance 辅助的局部定位

巡线分横竖，横线代表已行走的距离，竖线修正行走方向。

白色直线的提取方案如下：

（1） 利用 HSV 分量中的 S 分量滤除背景中高饱和度的干扰；

（2） 利用 V 分量阈值处理，提取出亮度比较高的白色区域;

（3） 由于 V 分量有时会将黑色区域当成白色区域，将处理后的 S 分量和 V 分量与运算；

（4） 腐蚀与膨胀，滤除噪声

（5） 累积概率霍夫线变换，提取直线（效率低，因此我们适度缩小了图像 160*120）

（6） 分类算法，将线变换后的直线分类，将同一直线处理后提取的许多条直线处理为一条直线（利用斜率和到中心的距离分类）

（7） 区分横线和竖线，提取出竖线，用来跟踪

（8） 跟踪算法，在同一图像中的多条竖线中选择一条去跟踪（使用历史值去搜索视野中的最小欧式距的线）

蓝边搜索算法

（1） 利用 H 分量提取出场地边蓝色区域

（2） 腐蚀膨胀，去除噪声（核设的比较大，用以滤除比较大的区块和连通蓝边）
（3） 轮廓提取，然后滤除图像边界的轮廓
（4） 在处理后的轮廓中提取直线
（5） 在已提取中的所有竖线中寻找与上一步提取的直线距离最小的一条，此直线即为蓝边
（6） 在竖线中寻找与蓝边最近的直线，作为蓝边旁边的线来跟踪

Guidance 辅助定位

Guidance 输出的光流速度在短时间内积分得到的距离还是可信的，所以在白线出现在图心附近时判断 Guidance 得到的距离与实际一个格子长度的误差是否在容忍范围内，以此修正纯粹的白横线得到的距离。

具体实现方法如下：

```
bool ColorImg::findLine(cv::Mat image)
{
    cv::resize(image, imgResize, cv::Size(160, 120),0,0,cv::INTER_LINEAR);
    line_exist = false;
    //初始化图像
    imgVthreshold = Mat::zeros(imgResize.rows,imgResize.cols,0);
    imgSthreshold  = Mat::zeros(imgResize.rows,imgResize.cols,0);
    imgAbstract = Mat::zeros(imgResize.rows,imgResize.cols,0);

    //[3]RGB->HSV
    imgHSV  = Mat::zeros(imgResize.rows,imgResize.cols,imgResize.type());
    cvtColor( imgResize, imgHSV, CV_BGR2HSV );
    //[4]分离HSV三个通道
    cv::split(imgHSV,channels);
//        imgH = channels.at(0);
    imgS = channels.at(1);
    imgV = channels.at(2);

    Mat imgHthreshold = (imgH>70)&(imgH<147);
    imgSmask = imgS > 173;
    imgVthreshold = imgV > 200;
    imgSthreshold = imgS < 50;
    imgAbstract = imgVthreshold&imgSthreshold&(~imgHthreshold);


    //[7]腐蚀
    erode_ele = getStructuringElement(0, Size(3, 3));
    imgAbstractErode  = Mat::zeros(imgAbstract.rows,imgAbstract.cols,imgAbstract.type());
    erode(imgAbstract, imgAbstractErode, erode_ele);

    //[8]膨胀
    dilate_ele = getStructuringElement(0, Size(3, 3));
```

```
    dilate(imgAbstractErode,imgAbstractDilate,dilate_ele);

    //[9]进行霍夫线变换
    vector<Vec4i> lines;//定义一个矢量结构lines用于存放得到的线段矢量集合
    HoughLinesP(imgAbstractDilate, lines, 1, CV_PI/90, 75, 25, 25 );

    //[10]分离图像中的横线和竖线
//    Mat imgLinesV,imgLinesH;//竖线和横线
//    imgLinesV  = Mat::zeros(imgResize.rows,imgResize.cols,imgResize.type());
//    imgLinesH  = Mat::zeros(imgResize.rows,imgResize.cols,imgResize.type());
    Mat imgLines;//所有线
    imgLines  = Mat::zeros(imgResize.rows,imgResize.cols,imgResize.type());
    vector<Vec4f> describeLine;//线段的描述,
                               //[0]代表斜率,[1]代表截距,
                               //[3]代表图形中心与直线之间的距离,[4]数目
    for( size_t i = 0; i < lines.size(); i++ )
    {
      Vec4i l = lines[i];
      Vec4f lineData;

      lineData[0] = (l[3]-l[1])/(l[2]-l[0]+0.0);//斜率
      lineData[1] = ((l[2]*l[1])-(l[3]*l[0]+0.0))/(l[2]-l[0]);//截距
      double k = lineData[0];//斜率
      double b = lineData[1];//截距
      //斜率无穷大,则k和b都是无穷大,此时lineData[0]任然存入无穷大,lineData[1]存入x的值
      if(((*(int64*)&k==0x7FF0000000000000LL))||((*(int64*)&k==0xFFF0000000000000LL) ))
      {
          lineData[0] = k;
          lineData[1] = l[0];
          lineData[2] = imgLines.cols/2-l[0];//保存中心到直线的距离
      }
      else
      {
          lineData[2] = (k*imgLines.cols/2 - imgLines.rows/2 + b)
                        /sqrt(1+k*k);//中心到线的距离
          if(k<0)
             lineData[2] = -lineData[2];
      }

      describeLine.push_back(lineData);  //插入新的元素

    }

    //[11]根据分离出的线段,将其分类整合,一簇的合并成一条线,存入classfiedLine
```

```
    vector<Vec4f> classfiedLine;//分类后的直线
    for(size_t i = 0; i < describeLine.size(); i++)//从线段的描述中分类
    {
        Vec4f lineData = describeLine[i];//一条直线的信息
        bool isExist = false;//有相似直线

        if(classfiedLine.size() != 0)
        {
            //遍历已经存在的每一条直线数据,找出最相似的
            for(size_t j = 0;j < classfiedLine.size();j++)
            {
                Vec4f templateLine = classfiedLine[j];//已经存在的模板直线

                //角度差小于20,图像中心到线的距离差小于30,这两个值可以调整
                //这里之所以不用截距是因为当直线斜率很大时,图像与坐标轴的交点在远处
                //一个很小的斜率差就会导致聚到的截距差,而无法把两条相近的直线分到一类
                if( ( abs((atan(lineData[0]) - atan(templateLine[0]))/CV_PI*180) < 20 )&&
                    abs(lineData[2] - templateLine[2])<25 )
                {
                    //判断为同一条直线,求平均(斜率和截距)
                    templateLine[0] = (templateLine[0]*lineData[3] +
lineData[0])/(lineData[3]+1);
                    templateLine[1] = (templateLine[1]*lineData[3] +
lineData[1])/(lineData[3]+1);
                    templateLine[2] = (templateLine[2]*lineData[3] +
lineData[2])/(lineData[3]+1);
                    lineData[3]++;
                    isExist = true;//找到相似直线,置标志位,跳出循环
                    break;
                }
                else
                {
                    //没有相似直线,清标志位
                    isExist = false;
                }
            }

            //如果没有找到相似直线,插入新的直线
            if(!isExist)
            {
                lineData[3] = 1;
                classfiedLine.push_back(lineData);  //插入新的元素
            }
        }
```

```
        else//还没有元素,直接插入新的元素
        {
            lineData[3] = 1;
            classfiedLine.push_back(lineData);  //插入新的元素
        }

    }

        if(classfiedLine.size()==0)
        {
            std::cout<<"there's no lines:  "<<std::endl;
            file_record<<"there's no lines "<<std::endl;
            return false;
        }

    //[12]画出分类的直线,通过命令行输出直线数据
    vector<Vec4f> VLines, HLines;//分类后的直线
    for( size_t i = 0; i < classfiedLine.size(); i++ )
    {
      Vec4f l = classfiedLine[i];
      double k = l[0];//斜率

      if((k<-
1)||(k>1)||((*(int64*)&k==0x7FF0000000000000LL))||((*(int64*)&k==0xFFF0000000000000LL)))//竖
线的斜率和中心点与它的距离
      {
          VLines.push_back(l);
      }
      else
      {
          HLines.push_back(l);

      }
    }//end for

////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////

/*****************************************************************************************
*******************************/

////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////
```

```
//////////////////////////////

        if(HLines.size()!=0)
        {
            for( size_t i = 0; i < HLines.size(); i++ )
            {

                if(abs(HLines[i][2])< 10 )
                {
                    // first step
                    if(start_walk_forward==false)
                    {
                        start_walk_forward = true;
                        hor_line_cnt += start_walk_forward?1:(-1);
std::cout<<"first forward:\t"<<control_x<<", \t"<<hor_line_cnt<<std::endl;
file_record<<"first forward:\t"<<walk_x<<",\t"<<control_x<<"\t"<<g_vo.vx<<"\t"<<vx_body_level
<<",\t"<<hor_line_cnt<<",\t"<<is_forward_back<<",\t"<<HLines[i][2]<<",\t"<<HLines.size()<<"\t
"<<walk_y<<"\t"<<control_y<<std::endl;
                        walk_x = 0;
                        control_x = 0;
                        break;
                    }

                    //change direction   drift of guidance(1m/10min)
                    if(abs(control_x)<500)
//                     if(abs(walk_x)<700)
                    {
//                        if(walk_x==0) continue;
                        if(control_x==0) continue;
                        else
                        {
                        //limit about change direction  (when low speed crossing the
horizontal lines
                        //1, speed direction(g_vo.vx)and is_forward_back  2, walk_x and
is_forward_back
//                            if(( g_vo.vx * (is_forward_back?1:(-1)) )<0)//
( (walk_x*(is_forward_back?1:(-1)))>0 ) //1. g_vo.vx&is_fr_bk 2. walk&is_fr_bk
                            if((vx_body_level* (is_forward_back?1:(-1)) )<0)
                            {
                                is_forward_back = ! is_forward_back;
                                hor_line_cnt += is_forward_back?1:(-1);
std::cout<<"change direction : "<<control_x<<", \t"<<hor_line_cnt<<std::endl;
file_record<<"change
direction:\t"<<walk_x<<",\t"<<control_x<<"\t"<<g_vo.vx<<"\t"<<vx_body_level
```

```
<<",\t"<<hor_line_cnt<<",\t"<<is_forward_back<<",\t"<<HLines[i][2]<<",\t"<<HLines.size()<<"\t
"<<walk_y<<"\t"<<control_y<<std::endl;
                                walk_x = 0;
                                control_x = 0;
                                break;
                            }
                        }
                    }
                    else if( abs(control_x%1000)<=350 )
                    {
                        hor_line_cnt +=  (int)( (control_x-control_x%1000) / 1000 ) ;
std::cout<<"walk over: "<<control_x<<", \t"<<hor_line_cnt<<std::endl;
file_record<<"walk over:\t"<<walk_x<<",\t"<<control_x<<"\t"<<g_vo.vx<<"\t"<<vx_body_level
<<",\t"<<hor_line_cnt<<",\t"<<is_forward_back<<",\t"<<HLines[i][2]<<",\t"<<HLines.size()<<"\t
"<<walk_y<<"\t"<<control_y<<std::endl;
                        walk_x = 0;
                        control_x = 0;
                        break;
                    }
                    else if( abs(control_x%1000)>=650 )
                    {
                        hor_line_cnt +=  (int)( (control_x-control_x%1000)/1000 +
(control_x>0)?1:(-1));
std::cout<<"walk lack: "<<control_x<<", \t"<<hor_line_cnt<<std::endl;
file_record<<"walk
lack:\t"<<walk_x<<",\t"<<control_x<<"\t"<<g_vo.vx<<"\t"<<vx_body_level<<",\t"<<hor_line_cnt<<
",\t"<<is_forward_back<<",\t"<<HLines[i][2]<<",\t"<<HLines.size()<<"\t"<<walk_y<<"\t"<<contro
l_y<<std::endl;
                        walk_x = 0;
                        control_x = 0;
                        break;
                    }
                    else
                    {
std::cout<<"g nearing: "<<control_x<<", \t"<<hor_line_cnt<<std::endl;
file_record<<"g nearing:\t"<<walk_x<<",\t"<<control_x<<"\t"<<g_vo.vx<<"\t"<<vx_body_level
<<",\t"<<hor_line_cnt<<",\t"<<is_forward_back<<",\t"<<HLines[i][2]<<",\t"<<HLines.size()<<"\t
"<<walk_y<<"\t"<<control_y<<std::endl;
                    }
                }
                else
                {
std::cout<<"color nearing: "<<control_x<<", \t"<<hor_line_cnt<<std::endl;
file_record<<"color nearing:\t"<<walk_x<<",\t"<<control_x<<"\t"<<g_vo.vx<<"\t"<<vx_body_level
```

```
<<",\t"<<hor_line_cnt<<",\t"<<is_forward_back<<",\t"<<HLines[i][2]<<",\t"<<HLines.size()<<"\t
"<<walk_y<<"\t"<<control_y<<std::endl;
                }
            }//end for( size_t i = 0; i < HLines.size(); i++ )
        }
        else
        {
std::cout<<"no hlines: "<<control_x<<", \t"<<hor_line_cnt<<std::endl;
file_record<<"no hlines:\t"<<walk_x<<",\t"<<control_x<<"\t"<<g_vo.vx<<"\t"<<vx_body_level
<<",\t"<<hor_line_cnt<<",\t"<<is_forward_back<<",\t"<<"null"<<",\t"<<"null"<<"\t"<<walk_y<<"\
t"<<control_y<<std::endl;
        }//end if(HLines.size()!=0)

line_sum = hor_line_cnt;

////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////

/*************************************************************************************
*******************************/

////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////
//////
//////////////////////////////////////////////////////////////////search the nernest line to
blue area
///        //[2]调整大小
//        resize(imgIn,imgResize,Size(160,120),0,0,INTER_LINEAR);

        //[3]RGB->HSV
        imgHSV  = Mat::zeros(imgResize.rows,imgResize.cols,imgResize.type());
        cvtColor( imgResize, imgHSV, CV_BGR2HSV );
        //分离HSV三个通道
        cv::split(imgHSV,channels);
        imgH = channels.at(0);
        imgS = channels.at(1);
        imgV = channels.at(2);

        imgHthreshold1 = (imgH>96)&(imgH<120);
        imgHmask = imgHthreshold1 > 90;
```

```
        /*
        //[4]H分量阈值调整
        cv::threshold(imgH,imgHthreshold1,96,255,cv::THRESH_TOZERO);
        cv::threshold(imgHthreshold1,imgHthreshold2,120,255,cv::THRESH_TOZERO_INV);

        imgHmask = imgHthreshold2 > 90;
*/
        //[5]V分量阈值调整

        //[6]求Mask
        imgMask = imgHmask;

        //[7]腐蚀
        erode_ele = getStructuringElement(0, Size(11, 11));
        erode(imgMask,imgMaskErode,erode_ele);

        //[8]膨胀
        dilate_ele = getStructuringElement(0, Size(20, 20));
        dilate(imgMaskErode,imgMaskDilate,dilate_ele);

        //[9]找轮廓
        Mat imgtmp =  Mat::zeros(imgResize.rows,imgResize.cols,0);
        vector<vector<Point> > contours;
        vector<Vec4i> hierarchy;
        findContours( imgMaskDilate, contours, hierarchy,RETR_CCOMP,CHAIN_APPROX_SIMPLE );
        //[10]求凸包画凸包
        vector<vector<Point> > hull(contours.size());
        for(uint i = 0;i<contours.size();i ++)
        {
            convexHull(Mat(contours[i]),hull[i],true);
        }
        Point2f center;
        float radius;
        for(uchar i = 0;i < contours.size();i ++)
        {
            Scalar color(255,255,255);
            //drawContours(imgResize,contours,i,color,1,8,vector<Vec4i>(),0,Point());
            drawContours(imgtmp,hull,i,color,2,8,vector<Vec4i>(),0,Point());

            //用圆将其圈住,求圆心
            minEnclosingCircle(contours[i],center,radius);
            circle(imgtmp,center,1,CV_RGB(255,100,100),1);
        }
```

```
        //[11]进行霍夫线变换
        vector<Vec4i> lines1;//定义一个矢量结构lines用于存放得到的线段矢量集合
        HoughLinesP(imgtmp, lines1, 2, CV_PI/90, 50, 20, 0 );

        //[12]将有效线段(去除靠近边界的线后的线)信息保存下来
        vector<Vec4f> describeLine1;//线段的描述,
                                    //[0]代表斜率,[1]代表截距,
                                    //[3]代表图形中心与直线之间的距离
        for( size_t i = 0; i < lines1.size(); i++ )
        {
          Vec4i l = lines1[i];

          //去掉图像上靠近边界的线
          if( ((l[0]>=imgMaskDilate.cols-10)&&((l[2]>=imgMaskDilate.cols-10)))
            ||((l[1]>=imgMaskDilate.rows-10)&&((l[3]>=imgMaskDilate.rows-10)))
            ||((l[0]<=10)&&((l[2]<=10)))
            ||((l[1]<=10)&&((l[3]<=10))))
          {}
          else
          {
              //依次在图中绘制出每条线段
              line( imgResize, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(0,0,255), 1,
CV_AA);

              ///保存直线信息
              Vec4i l = lines1[i];
              Vec4f lineData;
              lineData[0] = (l[3]-l[1])/(l[2]-l[0]+0.0);//斜率
              lineData[1] = ((l[2]*l[1])-(l[3]*l[0]+0.0))/(l[2]-l[0]);//截距
              double k = lineData[0];//斜率
              double b = lineData[1];//截距
              //斜率无穷大,则k和b都是无穷大,此时lineData[0]任然存入无穷大,lineData[1]存入x的
值
              if(l[2]==l[0])
              {
                  lineData[0] = k;
                  lineData[1] = l[0];
                  lineData[2] = imgResize.cols/2 - l[0];//保存中心到直线的距离
              }
              else
              {
                  lineData[2] = (k*imgResize.cols/2 - imgResize.rows/2 + b)
                                /sqrt(1+k*k);//中心到线的距离
```

```
                if(k<0)
                    lineData[2] = -lineData[2];
            }
            describeLine1.push_back(lineData);  //插入新的元素
        }
    }

    //[13]根据分离出的线段,将其分类整合,一簇的合并成一条线,存入classfiedLine
    vector<Vec4f> classfiedLine1;//分类后的直线
    for(size_t i = 0; i < describeLine1.size(); i++)//从线段的描述中分类
    {
        Vec4f lineData = describeLine1[i];//一条直线的信息
        bool isExist = false;//有相似直线

        if(classfiedLine1.size() != 0)
        {
            //遍历已经存在的每一条直线数据,找出最相似的
            for(size_t j = 0;j < classfiedLine1.size();j++)
            {
                Vec4f templateLine = classfiedLine1[j];//已经存在的模板直线

                //角度差小于20,图像中心到线的距离差小于30,这两个值可以调整
                //这里之所以不用截距是因为当直线斜率很大时,图像与坐标轴的交点在远处
                //一个很小的斜率差就会导致聚到的截距差,而无法把两条相近的直线分到一类
                if( ( abs((atan(lineData[0]) - atan(templateLine[0]))/CV_PI*180) < 20 )&&
                    abs(lineData[2] - templateLine[2])<40 )
                {
                    //判断为同一条直线,求平均(斜率和截距)
                    templateLine[0] = (templateLine[0]*lineData[3] +
lineData[0])/(lineData[3]+1);
                    templateLine[1] = (templateLine[1]*lineData[3] +
lineData[1])/(lineData[3]+1);
                    templateLine[2] = (templateLine[2]*lineData[3] +
lineData[2])/(lineData[3]+1);
                    lineData[3]++;
                    isExist = true;//找到相似直线,置标志位,跳出循环
                    break;
                }
                else
                {
                    //没有相似直线,清标志位
                    isExist = false;
                }
            }
```

```
            //如果没有找到相似直线,插入新的直线
            if(!isExist)
            {
                lineData[3] = 1;
                classfiedLine1.push_back(lineData);  //插入新的元素
            }
        }
        else//还没有元素,直接插入新的元素
        {
            lineData[3] = 1;
            classfiedLine1.push_back(lineData);  //插入新的元素
        }

    }

    //[12]画出分类的直线,通过命令行输出直线数据
    for( size_t i = 0; i < classfiedLine1.size(); i++ )
    {
      Vec4f l = classfiedLine1[i];
      double k = l[0];//斜率
      double b = l[1];//截距
      //图像中心点到直线的距离
//      double distance = l[2];//中心点到直线的距离

      Point p1(0,b),p2(-b/k,0);//画的两个点
      //斜率无穷大
      if(((*(int64*)&k==0x7FF0000000000000LL))||((*(int64*)&k==0xFFF0000000000000LL) ))
      {
          p1 = Point(l[1],0);
          p2 = Point(l[1],imgResize.rows);
      }
      else
      {
          //画到图像外部去了
          if(b<0)
          {
              p1.x = (imgResize.rows - b) / k;
              p1.y = imgResize.rows;
          }

          if(-b/k<0)
          {
              p2.x = imgResize.cols;
```

```
                p2.y = k*imgResize.cols + b;
            }

        }

        line( imgResize, p1, p2, Scalar(0,255,255), 3, CV_AA);

    //      //画中心到直线的垂线
    //      double theta = atan(-1/k);
    //      if(theta<0)//角度处理
    //          theta = theta + CV_PI;
    //      if(k<0)
    //          distance = -distance;
    //      Point
p3(imgResize.cols/2+distance*cos(theta),imgResize.rows/2+distance*sin(theta));

//line( imgResize,Point(imgResize.cols/2,imgResize.rows/2),p3,Scalar(255,0,0),2,CV_AA);
        }

        //[14]寻找线数最多的一簇线
        double maxnum = 0,num = 0;
        for(uint i = 0;i < classfiedLine1.size();i ++)
        {
            Vec4f lineData = classfiedLine1[i];

            //寻找线数最多的线
            if(lineData[3] > maxnum)
            {
                maxnum = lineData[3];
                num = i;
            }
        }

        bool isBlueLineFound = false;
        int index = 0;
        if(classfiedLine1.size()>0)
        {
            isBlueLineFound = true;

            Vec4f l = classfiedLine1[num];
            double k = l[0];//斜率
            double b = l[1];//截距
            //斜率无穷大
            Point p1((imgResize.rows - b) / k,imgResize.rows),p2(-b/k,0);//画的两个点
```

```
            if(((*(int64*)&k==0x7FF0000000000000LL))||((*(int64*)&k==0xFFF0000000000000LL) ))
            {
                p1 = Point(l[1],0);
                p2 = Point(l[1],imgResize.rows);
            }

            double distanceline = 1000;

            for(int i = 0;i < classfiedLine.size();i++)
            {
                double tmp;
                Vec4f l = classfiedLine[i];
                double k = l[0];//斜率
                double b = l[1];//截距
                if((k<-1)||(k>1))
                {
                    Point p3(- b / k,0),p4((imgResize.rows - b)/k,imgResize.rows);//画的两个
点
                    //斜率无穷大

if(((*(int64*)&k==0x7FF0000000000000LL))||((*(int64*)&k==0xFFF0000000000000LL) ))
                    {
                        p3 = Point(l[1],0);
                        p4 = Point(l[1],imgResize.rows);
                    }

                    if((p3.x+p4.x)/2<80)
                    {
                        p3.x=p3.x;
                    }

                    tmp = fabs((p3.x + p4.x)/2.0 - (p1.x+p2.x)/2.0);
                    if((tmp < distanceline)&&(tmp > 50))
                    {
                        distanceline = tmp;
                        index = i;
                    }
                }
            }

            {
                Vec4f l = classfiedLine[index];
                double k = l[0];//斜率
                double b = l[1];//截距
```

```
                //图像中心点到直线的距离
          //      double distance = l[2];//中心点到直线的距离

                Point p1(0,b),p2(-b/k,0);//画的两个点
                //斜率无穷大

if(((*(int64*)&k==0x7FF0000000000000LL))||((*(int64*)&k==0xFFF0000000000000LL) ))
                {
                    p1 = Point(l[1],0);
                    p2 = Point(l[1],imgResize.rows);
                }
                else
                {
                    //画到图像外部去了
                    if(b<0)
                    {
                        p1.x = (imgResize.rows - b) / k;
                        p1.y = imgResize.rows;
                    }

                    if(-b/k<0)
                    {
                        p2.x = imgResize.cols;
                        p2.y = k*imgResize.cols + b;
                    }
                }
                line( imgResize, p1, p2, Scalar(255,255,0), 3, CV_AA);
            }

        }
////////////////////////////////////////////////////////////////search the nernest line to
blue area
//////////////////////////////////////////////////////////////////////////////////////////
//////
        if(isBlueLineFound)//isBlueLineFound
        {
            pixel_center_dis = classfiedLine1[index][2];
            line_slope = atan( classfiedLine1[index] [0])*180.0/PI;

        }
        else
        {
            if(system_location == line_left)
                pixel_center_dis = -20;
```

```
            if(system_location == line_right)
                pixel_center_dis = 20;
            line_slope = 90;
        }


    cv::imshow("findLine", imgResize);
    m_deal_writer<<imgResize;
    return line_exist;

}
```

三 系统控制架构

由于早期方案的不确定性，在系统架构上选择了状态切换的模式来做。由于通过所在位置和状态，可以确定需要的信息源和要执行的运动控制。然后在视觉处理时通过 switch case 确定需要做的图像处理，在运动控制时也是这样确定需要执行的运动控制方式。

```
//action
enum status
{
    take_off,
    landing,
    follow_line,
    fix_circle,
    fix_sudoku,
    fix_start_square,
    fix_end_square,
    turn_left,
    turn_right,
    get_ball,
    drop_ball,
    default_status
};
//location
enum location
{
    line_right,
    line_down,
    line_left,
    line_up,
```

```
        start_square,
        end_square,
        disk,
        sudoku,
        defualt_location
    };
```

四 结论与不足

上述算法中由于时间原因，选择了对光线比较环镜比较敏感但实现比较简单的巡线与 Guidance 组合定位方式。这种方法在摄像头曝光和白平衡已经场地光线变化比较大的时候就需要重新调整预处理的参数，这对 V 分量影响比较大，因此比赛中我们白线方面的处理方法存在缺陷，适应性还有待提高。