风车能量机关检测识别 (Windmill)

一、流程总览

对于风车能量机关的检测识别，目标装甲板是待击打状态扇叶上的装甲板（扇叶的状态分为两种，一种是呈现“锤子”形状的待击打状态，另一种是呈现“宝剑”形状的击打过状态，详见图1.2 图1.3），基本原理主要是先根据事先采集的扇叶模板，通过模板匹配筛选出待击打状态的扇叶，再根据 OpenCV 图像轮廓与其子轮廓的关系来确定待击打扇叶上的装甲板，最后再通过拟合圆法来对目标装甲板进行预测。

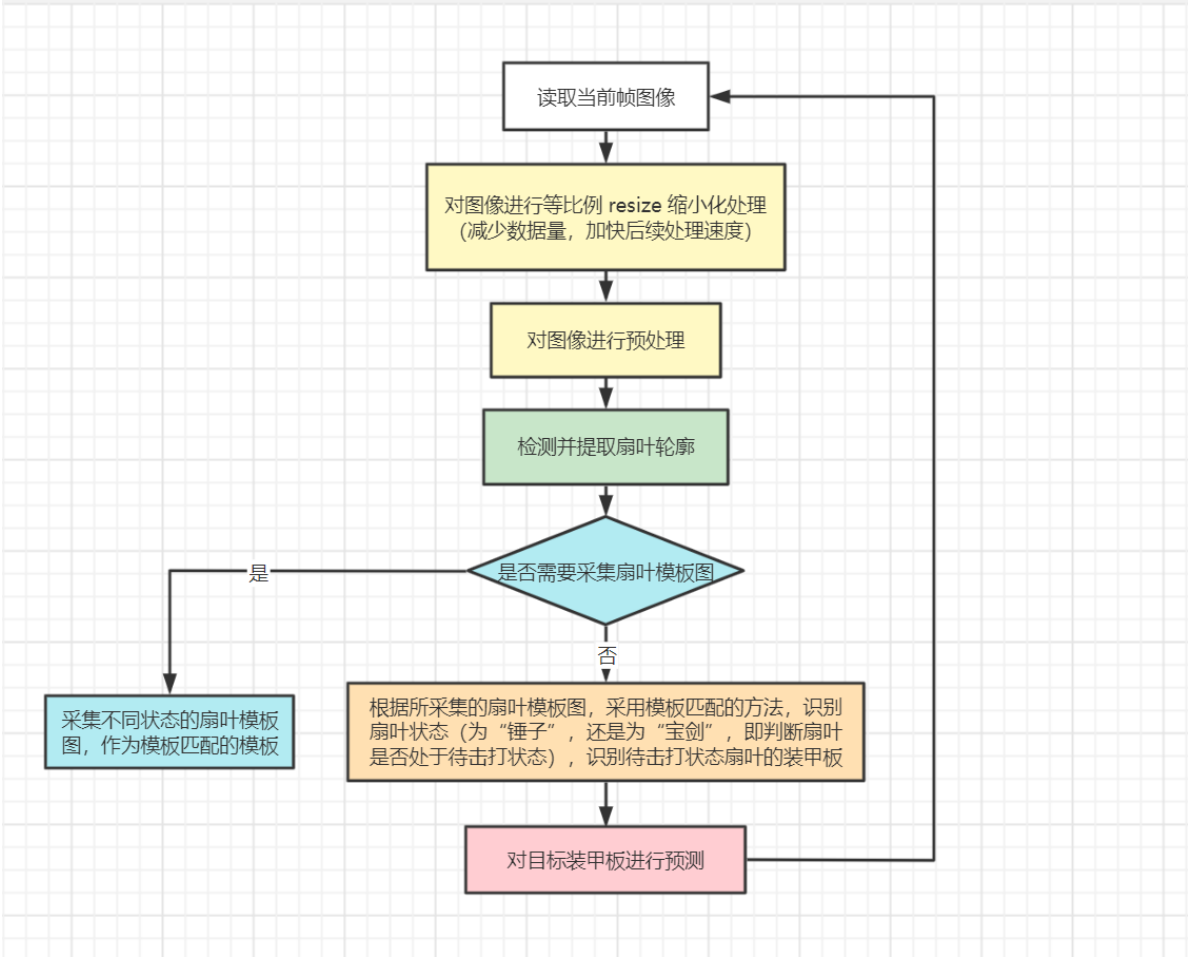

图1.1 风车能量机关识别流程图

图1. 2 待击打状态的扇叶（呈“锤子”状）

图1.3 击打过状态的扇叶（呈“宝剑”状）

二、图像预处理

与装甲板自瞄一样先根据风车灯条颜色，通道相减获取单通道灰度图，对其二值化，但由于我们后面是根据扇叶轮廓的子轮廓来确定装甲板的，因此这一步需要膨胀与开操作，来确保外层轮廓的连通（即使锤柄处的轮廓连通，避免对装甲板轮廓产生干扰），更好地去提取子轮廓。（关于图像轮廓，详见 三、扇叶状态的检测与目标装甲板的识别（模板匹配法））

图2.1 二值化后的待击打扇叶的轮廓

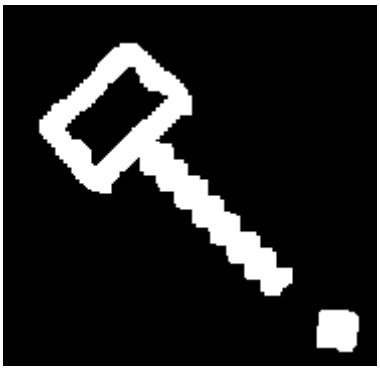

图2.2 经膨胀、开操作后处理的扇叶轮廓，可以看到锤柄处的箭头轮廓都连通到一起了

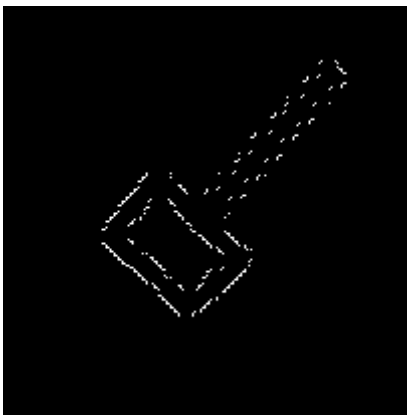

图2.3 待击打扇叶轮廓由两部分组成，一个外轮廓一个外轮廓的子轮廓

图2.4 已被击打过的扇叶轮廓，有多个轮廓

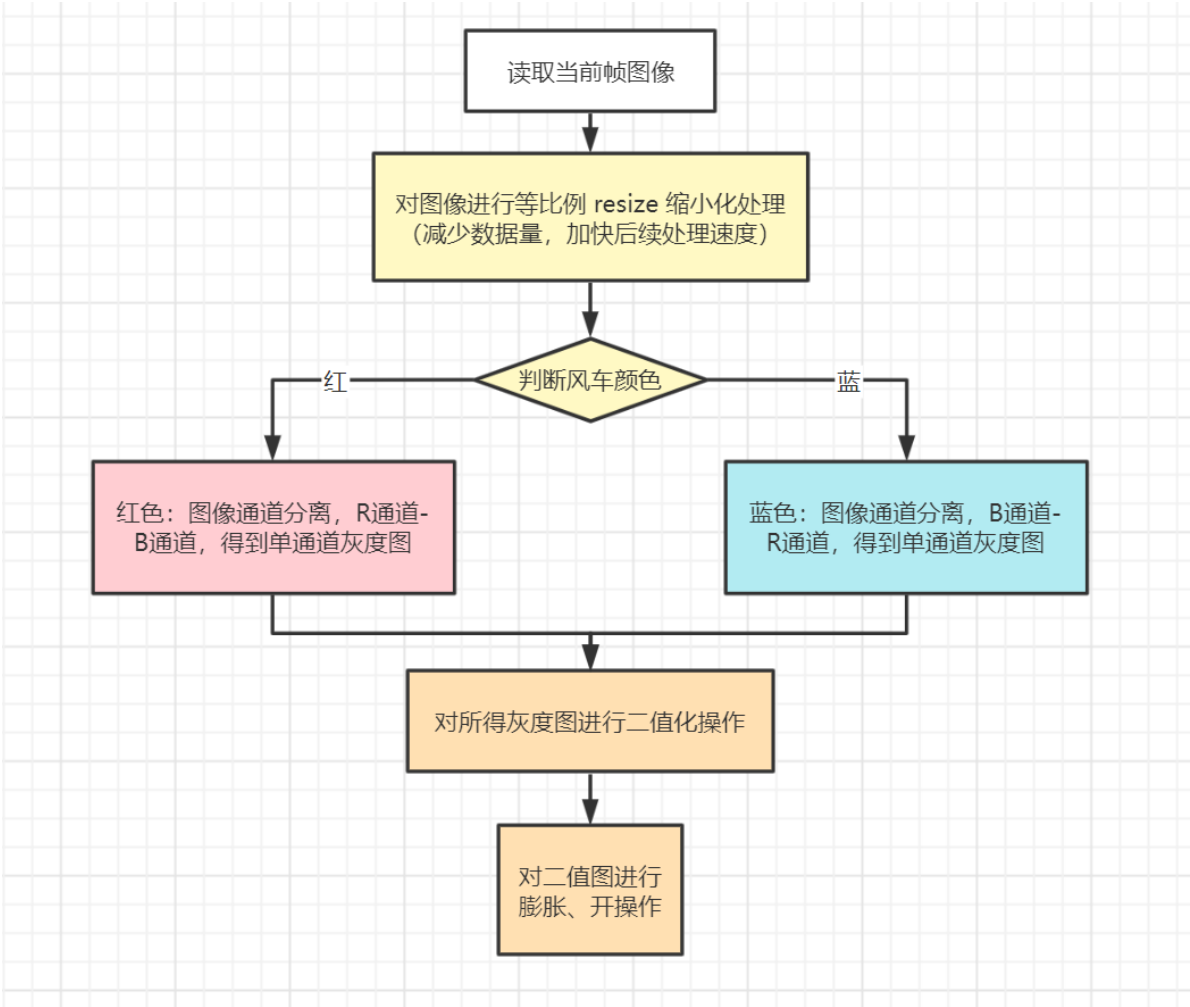

图2.3 图像预处理流程

核心代码：

```
Mat ImageProcess::Windmill_ImgProcess( int windmill_color ){

    //1.分离通道，转换为灰度图
    Mat grayImg;
    vector<Mat> channels;
    split( srcImg, channels );
    if( windmill_color == 0 ){
        grayImg = channels.at(2) - channels.at(0);
    }else{
        grayImg = channels.at(0) - channels.at(2);
    }

    //2.二值化灰度图
    Mat threImg;
    threshold( grayImg, threImg, Win_ThreSh_Value, 255, THRESH_BINARY );

    //3.膨胀
    Mat dilImg;
    int structElementSize=2;
    Mat element =
getStructuringElement(MORPH_RECT,Size(2*structElementSize+1,2*structElementSize+
1),Point(structElementSize,structElementSize));
    dilate( threImg, dilImg, element );

    //4.开运算，消除扇叶上可能存在的小洞
    structElementSize=3;
    element =
getStructuringElement(MORPH_RECT,Size(2*structElementSize+1,2*structElementSize+
1),Point(structElementSize,structElementSize));
    morphologyEx( dilImg, this->procImg, MORPH_CLOSE, element );

    return this->procImg;
}
```

代码2.1 图像预处理代码

三、扇叶状态的检测与目标装甲板的识别（模板匹配法）

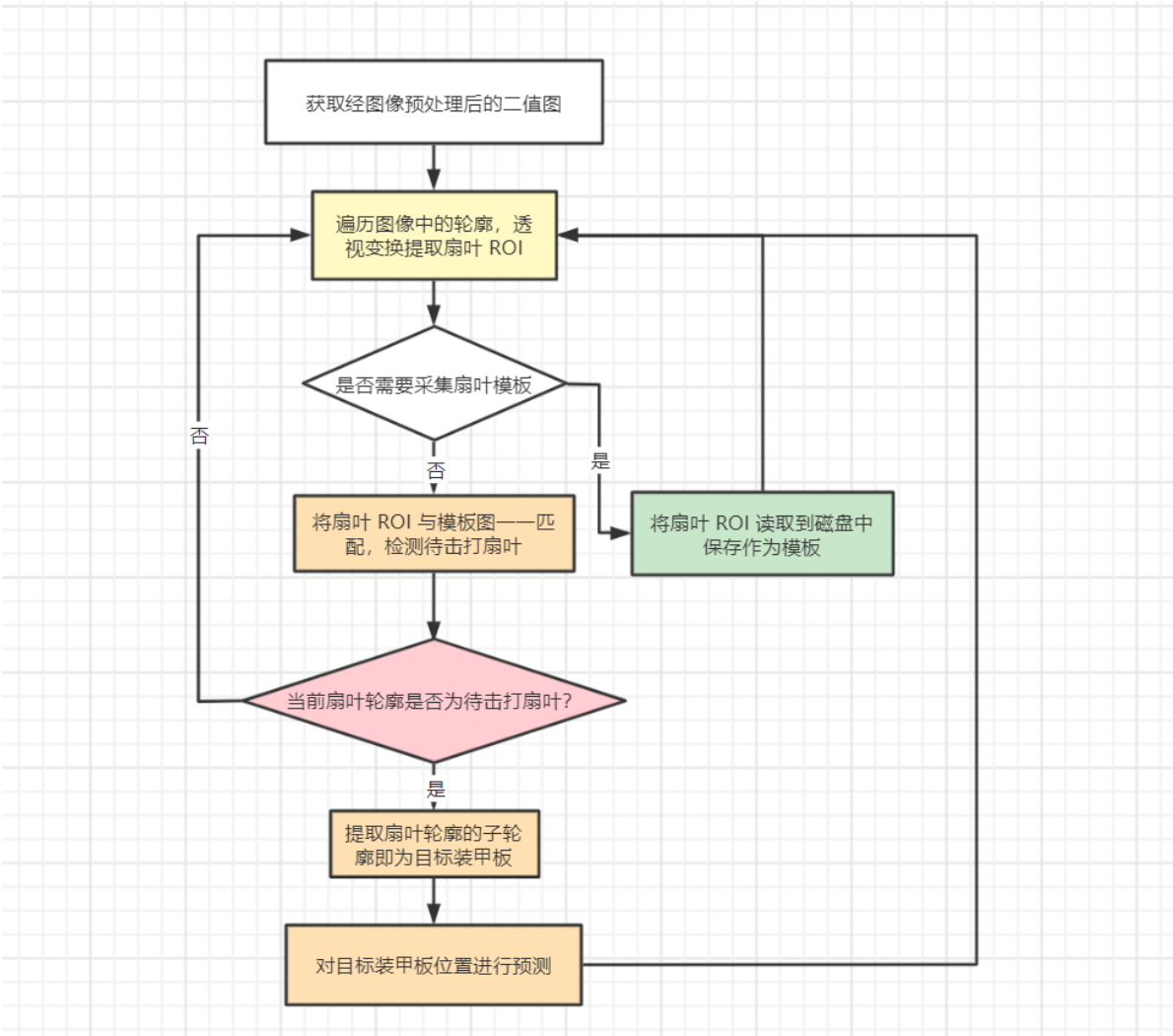

图3.1 流程总览

1. 扇叶轮廓的提取与 OpenCV 中图像轮廓的说明

1). 轮廓的遍历

这里同装甲板自瞄一样轮廓的提取也是通过 findContours 来获取的

```
vector< vector<Point> > contours; //存储轮廓点坐标
vector<Vec4i> hierarchy; //记录轮廓以及相关轮廓的编号
findContours( procImg, contours, hierarchy, CV_RETR_TREE, CHAIN_APPROX_SIMPLE );

if( hierarchy.size()>0 ){// 在查找轮廓前首先要判断轮廓是不是为空，之后再开始轮廓查找
    RotatedRect tmp_rect;
    for( int i = 0; i > -1; i = hierarchy[i][0] ){ //遍历图像中所有最外侧的轮廓
        int child = hierarchy[i][2]; //hierarchy[i][2]表示编号为i的轮廓的子轮廓
        if( child == -1 ) continue; //若child == -1表示当前轮廓i没有子轮廓，不是扇叶
        // 后续代码过多，此处省略，详见代码附录 ......
    }
}
```

代码3.1 图像中轮廓的遍历

2). OpenCV 中图像轮廓的说明

a. findContours 函数

对于 findContours 函数的一些理解可以参考此篇博客：
[opencv cv.findContours 函数详解*Ibelievesunshine*的博客-*CSDN*博客cv.findcontours]

对于findContours()函数，我们重点要看的是此函数中，在参数mode == CV_RETR_TREE时，参数hierarchy的一些概念：

- CV_RETR_TREE，检测所有轮廓，所有轮廓建立一个等级树结构; 外层轮廓包含内层轮廓，内层轮廓还可以继续包含内嵌轮廓
- mode == CV_RETR_TREE 表示寻找轮廓的方式是等级树结构 (也就是说将图像中的轮廓以树结构存储起来，树的每个节点存储的信息是轮廓，越外侧的轮廓在树中所处的位值置越"高")
- hierarchy包含了轮廓的拓扑结构，`hierarchy[i][0]~hierarchy[i][3]` 中，0代表与当前轮廓平级的后一个轮廓的的索引编号、1代表与当前轮廓平级的前一个轮廓的索引编号、2代表当前轮廓的子轮廓的索引编号、3代表当前轮廓的父轮廓的索引编号。
- 所以遍历轮廓的语句会写成 `for(int i=0;i>=0;i=hierarchy2[i][0])` hierarchy 的大小和 contours 的大小一样，所以若其大小为零说明没有轮廓也就不能遍历了，遍历会报错。

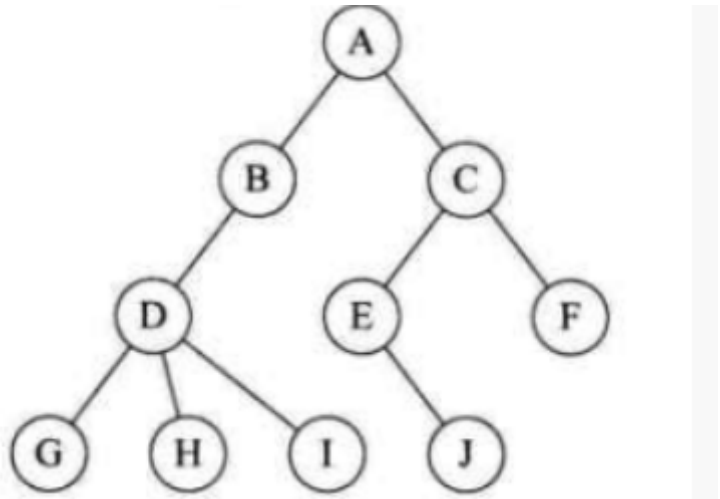

图3.2 树级结构

- 以上图中D节点为例：

  `hierarchy[D][0]` 表示 next : 与当前轮廓处于同一层级的下一条轮廓，最靠近当前轮廓节点的右兄弟节点存储的轮廓，图中的 E 节点存储的轮廓

  `hierarchy[D][1]` 表示previous：与当前轮廓处于同一层级的上一条轮廓，最靠近当前轮廓节点的左兄弟节点存储的轮廓，由于 D 就是最左边了所以没有，`hierarchy[D][1] == -1`，如果以图中的E节点为例的话，`hierarchy[E][1]` （ E 的previous）为图中的D点

  `hierarchy[D][2]` 表示 first child : 当前轮廓的第一个子轮廓，当前轮廓节点的孩子节点中最左侧的节点存储的轮廓，即图中的 G

  `hierarchy[D][3]` 表示 parent: 当前轮廓的父轮廓，当前轮廓节点的父节点存储的轮廓，即图中的B

  当没有轮廓时，序号就变为-1

b. 轮廓的遍历顺序

根据轮廓的中心坐标：从下到上，从左到右

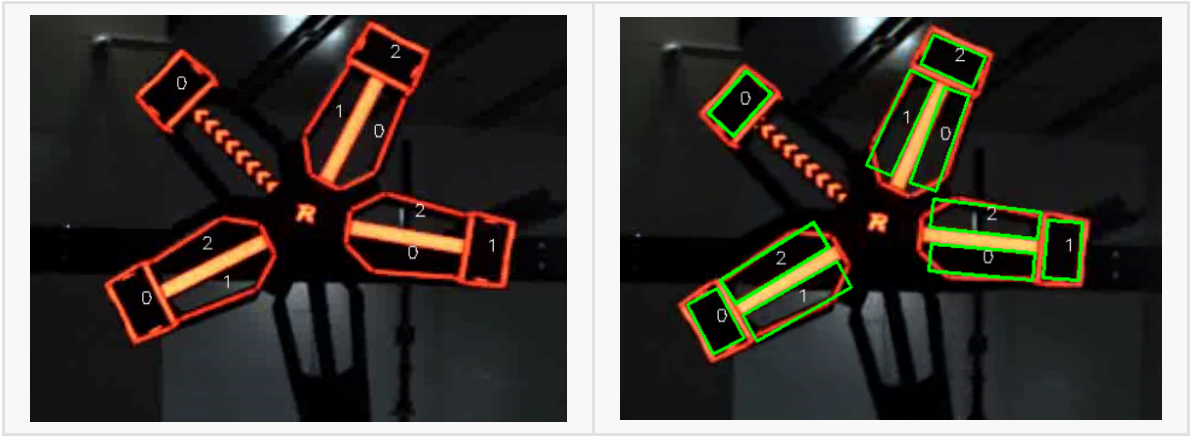

图3.3 扇叶树级轮廓遍历顺序演示

```
if( hierarchy.size()>0 ){
    for( int tmp_child = hierarchy[i][2], t = 0; tmp_child != -1; tmp_child =
hierarchy[tmp_child][0], t++ ){
        tmp_rect = minAreaRect( contours[tmp_child] );
        Point2f tmp_points[4];
        tmp_rect.points(tmp_points);
        drawArea( dstImg, tmp_points, gcolor );
        string c = to_string(t);
        putText( dstImg, c, tmp_rect.center, FONT_HERSHEY_SIMPLEX, 0.5,
Scalar(255, 255, 255) );
    }
}
```

代码3.2 图像轮廓遍历顺序测试

2. 模板匹配所需扇叶模板图的获取

采集的扇叶模板就是通过透视变换矫正提取扇叶 ROI 来获取的（这步无论是否需要采集模板都需要进行），对于模板匹配来说原图像与模板图像的大小越接近，匹配效果就越好，因此我们需要统一设定好模板的大小与所提取的扇叶 ROI 的大小，让二者大小一致（大小不宜过大，处理会更耗时些）。

```
//透视变换：将img图像上位于srcRect[4]坐标上的图像，透视变换到dstRect[4]坐标处
Mat PerspectiveTransform( Mat img, Point2f srcRect[4], Point2f dstRect[4] ){
    //得到透视变换矩阵
    Mat transform = getPerspectiveTransform( srcRect, dstRect );
    //进行透视变换，perspectMat是透视变换后的图像
    Mat perspectMat;
    warpPerspective( img, perspectMat, transform, img.size() );
    //imshow( "perspectMat", perspectMat );
    return perspectMat;
}
```

代码3.3 透视变换

```
tmp_rect = minAreaRect( contours[i] );
Point2f P[4];
tmp_rect.points(P); //将矩形的四个点保存在P中
//为透视变换做准备
Point2f srcRect[4]; //透视变换前的轮廓外接矩的四个顶点坐标
```

```
Point2f dstRect[4]; //透视变换后的轮廓外接矩的四个顶点坐标
double width  = getDistance( P[0], P[1] );
double height = getDistance( P[1], P[2] );

//矫正提取的叶片的宽高，这一步对每个轮廓的宽高进行处理使宽大于高，也就是在透视变换后将是一个不扭曲的长方形，如果不进行这一步可能会得到图像很胖的长方形。
if(width>height){
    srcRect[0]=P[0]; srcRect[1]=P[1];
    srcRect[2]=P[2]; srcRect[3]=P[3];
}else{
    swap(width,height);
    srcRect[0]=P[1]; srcRect[1]=P[2];
    srcRect[2]=P[3]; srcRect[3]=P[0];
}
```

代码3.4 透视变换的前期准备

```
//通过面积筛选
double area=height*width;
if( area>5000 ){ // 过滤掉面积过小的轮廓

    dstRect[0]=Point2f(0,0);        dstRect[1]=Point2f(width,0);
    dstRect[3]=Point2f(0, height);  dstRect[2]=Point2f(width,height);
    // 透视变换提取到扇叶 ROI testim
    Mat perspectiveMat = PerspectiveTransform( procImg, srcRect, dstRect );
    Mat testim = perspectiveMat(Rect(0,0,width,height));
    //imshow( "testim", testim );
    if( testim.empty() ){
        cout<<"filed open"<<endl;
        continue;
    }

    //存储图像：如果已采集过模板图，那么此段可以注释掉
    Mat storImg;
    resize( testim, storImg, Size(280, 150) ); //调整存储的模板匹配图像的的大小一致
    imshow("storImg", storImg);

    //存储锤子模板图
    string s = "hammerLeaf"+to_string(cnnt)+".jpg";
    imwrite("/home/xuelei/RM/上位机_视
觉/include_VisualGroup/Windmill2021_xuelei/templateLeaves/hammerLeaves/"+s,
storImg);

    //存储宝剑模板图
    string s = "swordLea"+to_string(cnnt)+",jpg";

 imwrite("/home/xuelei/rm_ws/src/hkvs/src/include_VisualGroup/Windmill2021_xuele
i/templateLeaves/swordLeaves"+s, storImg);
    cnnt++;

    /*
      后续模板匹配操作，此处省略...
     */
}
```

代码3.5 透视变换获取扇叶 ROI 并采集作为模板图

图3.4 采集到的待击打状态的扇叶的模板图

图3.5 采集到的已击打状态的扇叶的模板图

3. 扇叶状态的区分与识别

我们所要识别的是未被击打过的扇叶即“锤子”状的扇叶，通过前文透视变换提取到的扇叶 ROI 与采集的扇叶模板图进行模板匹配，根据匹配的结果来检测未被击打过的扇叶。

是锤子扇叶的条件：

1. 首先保证图像是个锤子即 hammer_value[hammer_maxv] > sword_value[sword_maxv] 锤子的匹配结果值的最大值要比宝剑匹配结果的最大值要大
2. 保证锤子模板匹配最大值要 > 0.6即 hammer_value[hammer_maxv] > 0.6

```
//对传入的图像与先前导入的模板图进行模板匹配，并返回匹配值
double Windmill::WinMatchTemplate( Mat img, Mat tmpl_leaf, Point& matchPosi, int
method ){

    double value = 0;
    int cols = img.cols - tmpl_leaf.cols + 1;
    int rows = img.rows - tmpl_leaf.rows + 1;
    Mat result = Mat( cols, rows, CV_32FC1 );
    matchTemplate( img, tmpl_leaf, result, method );

    double minVal, maxVal;
    Point minLoc, maxLoc;
    minMaxLoc( result, &minVal, &maxVal, &minLoc, &maxLoc, Mat() );

    switch(method){

        case CV_TM_SQDIFF:
        case CV_TM_SQDIFF_NORMED:
            matchPosi = minLoc;
            value = minVal;

        default:
```

```
            matchPosi = maxLoc;
            value = maxVal;

    }
    return value;
}
```

代码3.6 单个模板图与提取的扇叶 ROI 进行模板匹配

将采集到的模板图依次与提取到的扇叶 ROI 进行模板匹配，最后根据匹配值来判断扇叶的状态。

```
//判断扇叶是否为锤子扇叶（即待打击扇叶）
bool Windmill::isHammerLeaf( Mat img ){

    Point matchPosi;
    double value;

    // tmpl_hammerLeaves[i]，tmpl_swordLeaves[i] 为预先读取的扇叶模板图
    for( int i = 0; i < tmpHamNum ; ++i ){
        value = WinMatchTemplate( img, tmpl_hammerLeaves[i], matchPosi,
CV_TM_CCOEFF_NORMED);
        this->hammer_value.push_back(value);
    }

    for( int i = 0; i < tmpSwoNum; ++i ){
        value = WinMatchTemplate( img, tmpl_swordLeaves[i], matchPosi,
CV_TM_CCOEFF_NORMED);
        this->sword_value.push_back(value);
    }

    //与锤子扇叶模板匹配的最大值编号
    int hammer_maxv = 0;
    for( int t1 = 1; t1 < tmpHamNum; t1++ ){
        if( this->hammer_value[t1] > this->hammer_value[hammer_maxv]){
            hammer_maxv=t1;
        }
    }

    //与宝剑扇叶模板匹配的最大值编号
    int sword_maxv = 0;
    for( int t2 = 1; t2 < tmpSwoNum; t2++ ){
        if( this->sword_value[t2] > this->sword_value[sword_maxv] ){
            sword_maxv=t2;
        }
    }

    cout<<"hammer: "<<this->hammer_value[hammer_maxv]<<endl;
    cout<<"sword: "<<this->sword_value[sword_maxv]<<endl;

    /*
        是锤子扇叶的条件：
            1.首先保证图像是个锤子即 hammer_value[hammer_maxv] >
sword_value[sword_maxv]
            2.保证锤子模板匹配最大值要>0.6即 hammer_value[hammer_maxv]>0.6
    */
```

```
    bool result = ( ( this->hammer_value[hammer_maxv] > this-
>sword_value[sword_maxv] ) && this->hammer_value[hammer_maxv] > 0.8 );

    //一定要清空向量 hammer_value sword_value 内的值，防止下次调用此函数时，先前存储的数据
造成干扰
    clearHamSwdValue();

    return result;

}
```

代码3.7 待击打状态扇叶的检测

4. 目标装甲板的识别

结合前文提到的"待击打扇叶的子轮廓即为目标装甲板"，我们便可以在检测到待击打扇叶后，提取其子轮廓并进行标记记录来获取目标装甲板。

```
//判断是否为锤子（即判断是否为待击打的扇叶）
if( wind_mill.isHammerLeaf( comImg ) ){

    //在之前已经有对是否有子轮廓的判断，能到这里就能保证当前轮廓一定是有子轮廓的
    RotatedRect child_rect = minAreaRect( contours[child] );
    Point2f child_p[4];
    child_rect.points( child_p );

    float width  = child_rect.size.width;
    float height = child_rect.size.height;

    if( height > width ) swap(height,width);
    winArmor.modifyArmorParam(height, width);

    //进行条件筛选
    //if( winArmor.isArmor() ){
    Point2f armor_center = child_rect.center;
    circle( dstImg, armor_center, height/2, gcolor, 2 );
    drawArea( dstImg, child_p, gcolor );
    float angle = 0;

    posi.InputImagePoints_rotObj2( dstImg, armor_center, child_p[0], child_p[1],
child_p[2], child_p[3], child_rect );
    Point2f armor_next_posi = FittingCircle( dstImg, theta, height/2,
armorCenters, armor_center );

    //}

}
```

代码3.8 目标装甲板的获取

四、对目标装甲板的预测（拟合圆法）

参考此篇博客实现：[最小二乘法拟合圆Ivan 的专栏-CSDN博客最小二乘法拟合圆]

```
/* 拟合圆部分：
 * 通过最小二乘法来拟合圆的信息
 * pts: 所有点坐标
 * center: 得到的圆心坐标
 * radius: 圆的半径
 */
bool CircleInfo(std::vector<cv::Point2f>& pts, cv::Point2f& center, float&
radius){
    center = cv::Point2d(0, 0);
    radius = 0.0;
    if (pts.size() < 3) return false;;

    double sumX = 0.0;
    double sumY = 0.0;
    double sumX2 = 0.0;
    double sumY2 = 0.0;
    double sumX3 = 0.0;
    double sumY3 = 0.0;
    double sumXY = 0.0;
    double sumX1Y2 = 0.0;
    double sumX2Y1 = 0.0;
    const double N = (double)pts.size();
    for (int i = 0; i < pts.size(); ++i)
    {
        double x = pts.at(i).x;
        double y = pts.at(i).y;
        double x2 = x * x;
        double y2 = y * y;
        double x3 = x2 *x;
        double y3 = y2 *y;
        double xy = x * y;
        double x1y2 = x * y2;
        double x2y1 = x2 * y;

        sumX += x;
        sumY += y;
        sumX2 += x2;
        sumY2 += y2;
        sumX3 += x3;
        sumY3 += y3;
        sumXY += xy;
        sumX1Y2 += x1y2;
        sumX2Y1 += x2y1;
    }
    double C = N * sumX2 - sumX * sumX;
    double D = N * sumXY - sumX * sumY;
    double E = N * sumX3 + N * sumX1Y2 - (sumX2 + sumY2) * sumX;
    double G = N * sumY2 - sumY * sumY;
    double H = N * sumX2Y1 + N * sumY3 - (sumX2 + sumY2) * sumY;

    double denominator = C * G - D * D;
    if (std::abs(denominator) < DBL_EPSILON) return false;
    double a = (H * D - E * G) / (denominator);
    denominator = D * D - G * C;
    if (std::abs(denominator) < DBL_EPSILON) return false;
```

```
    double b = (H * C - E * D) / (denominator);
    double c = -(a * sumX + b * sumY + sumX2 + sumY2) / N;

    center.x = a / (-2);
    center.y = b / (-2);
    radius = std::sqrt(a * a + b * b - 4 * c) / 2;
    return true;
}
```

代码4.1 拟合圆

```
//返回预测得到的装甲板出现的下一个位置的坐标
Point2f FittingCircle( Mat& img, double theta, int r, vector<Point2f>& circleV,
Point2f center ){

    Point2f resPoint = (Point2f)(-1, -1); //预测得到的装甲板中心运动的下一个位置坐标
    Point2f cc; //计算得到的拟合圆圆心

    //在得到装甲板中心点后将其放入缓存队列中
    //用于拟合圆，用30个点拟合圆
    if( circleV.size() < 30 ){
        circleV.push_back( center );
    }else{
        float R;
        //得到拟合的圆心
        CircleInfo( circleV, cc, R );
        circle( img, cc, R, gcolor, 2 );
        //circleV.clear();
    }
    //将打击点围绕圆心旋转某一角度得到预测的打击点
    if( cc.x != 0 && cc.y != 0 ){
        //得到旋转一定角度（这里是30度）后点的位置
        Mat rot_mat = getRotationMatrix2D(cc,theta,1);
        float sinA=rot_mat.at<double>(0,1);//sin(theta);
        float cosA=rot_mat.at<double>(0,0);//cos(theta);
        float xx=-(cc.x-center.x);
        float yy=-(cc.y-center.y);
        resPoint = Point2f( cc.x+cosA*xx-sinA*yy, cc.y+sinA*xx+cosA*yy );
        circle( img, resPoint, r, rcolor, 3 );
    }

    return resPoint;

}
```

代码4.2 预测目标坐标

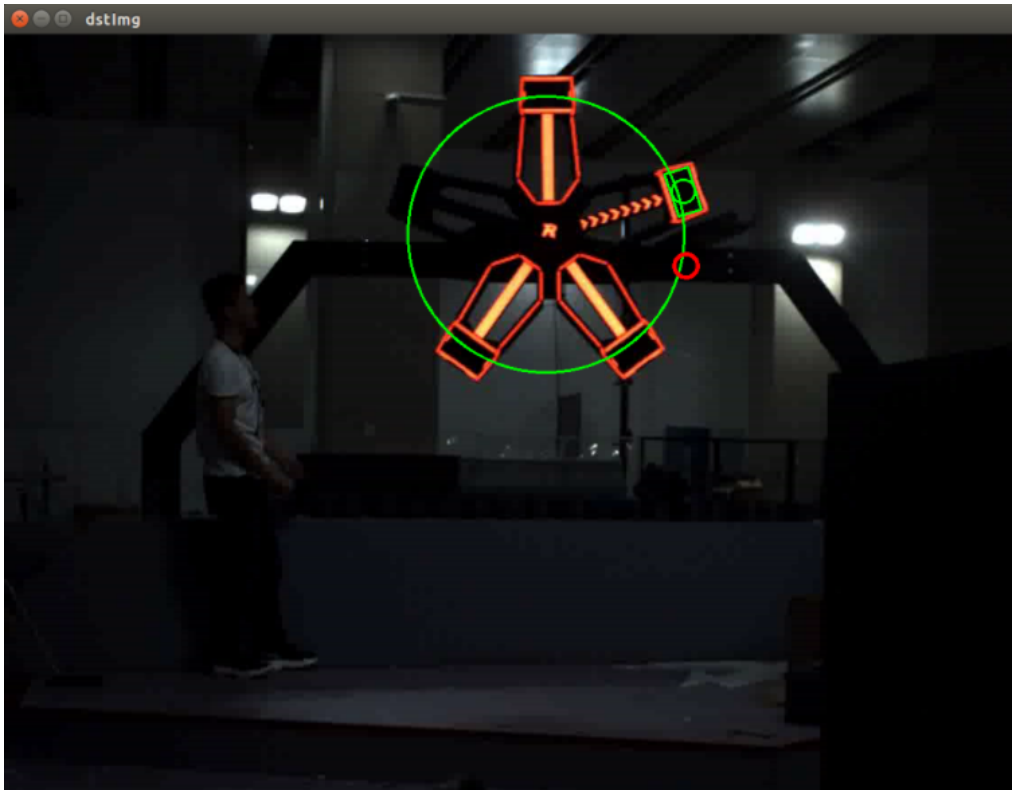

图4.1 识别效果1

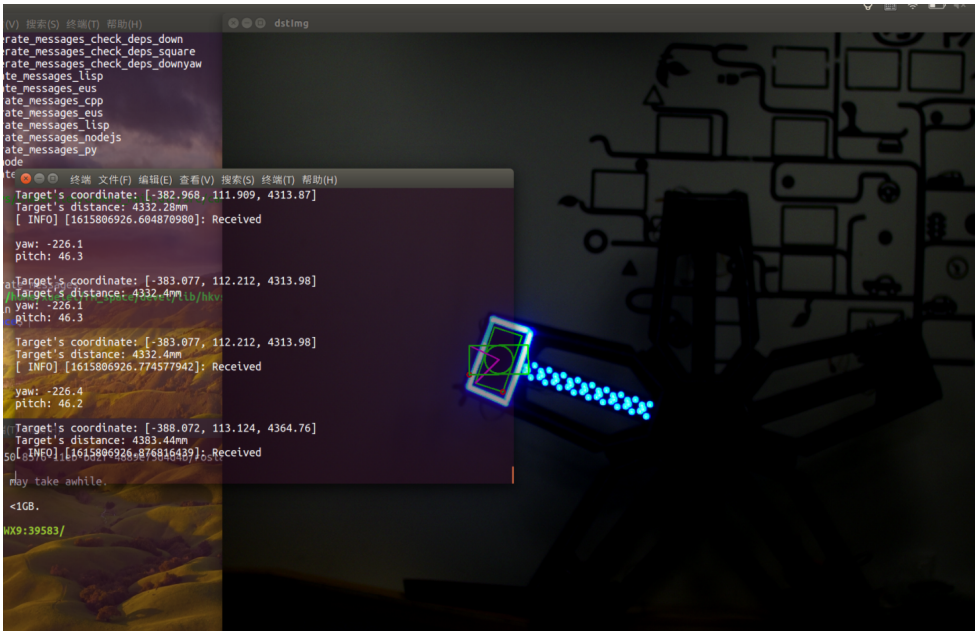

图4.2 识别效果2

五、参考资料

RoboMaster视觉教程（9）风车能量机关识别2江达小记-*CSDN*博客能量机关

opencv cv.findContours 函数详解*Ibelievesunshine*的博客-*CSDN*博客cv.findcontours

Java OpenCV findContours函数RETR_TREE轮廓顺序取名为猫的狗的博客-*CSDN*博客findcontours java

最小二乘法拟合圆*Ivan* 的专栏-*CSDN*博客最小二乘法拟合圆

风车能量机关 | ONES Wiki