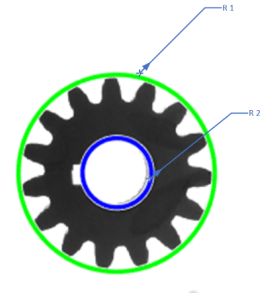
齿轮冲压件测量及图形化标注

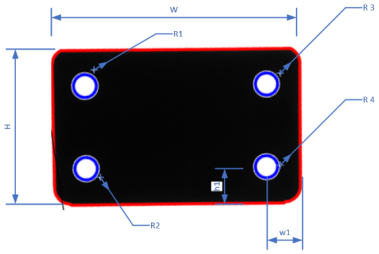
Edited by Alvis in 2024.10.14

一、任务

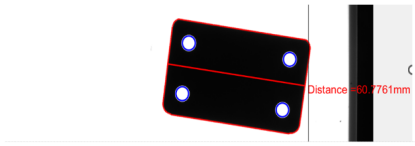
1 任务1和任务2，根据实验一采集的图片，进行标定，测出关键尺寸
2 任务3，实现鼠标点击，测出任意两点的距离
3 可以用maltab， python， c++任一工具实现
4 考核：分组到实验室一一演示，并现场给出相似性成绩，具体时间再通知



任务1：齿轮：
1 拟合孔和齿顶圆
2 自动计算孔、齿顶圆、齿根圆的圆心坐标和半径



任务2：冲压件：
1 拟合四边形和4个孔
2 自动计算长宽W/H
3 自动计算4个小圆的圆心坐标和半径
4 自动计算孔边距w1和h1



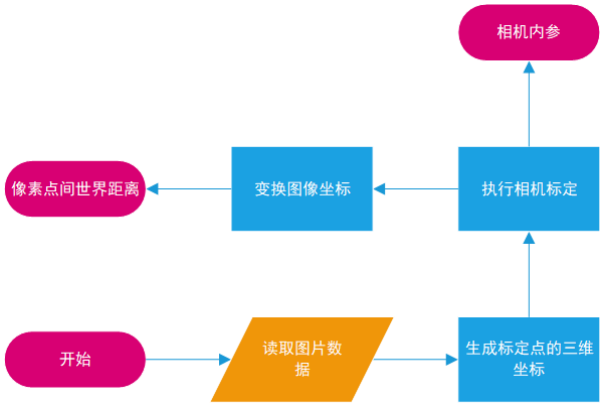
任务3：鼠标任意点两个点，自动计算并显示毫米距离

二、检测流程图

本任务主要分为三部分，相机内参标定、齿轮测量和冲压片测量。

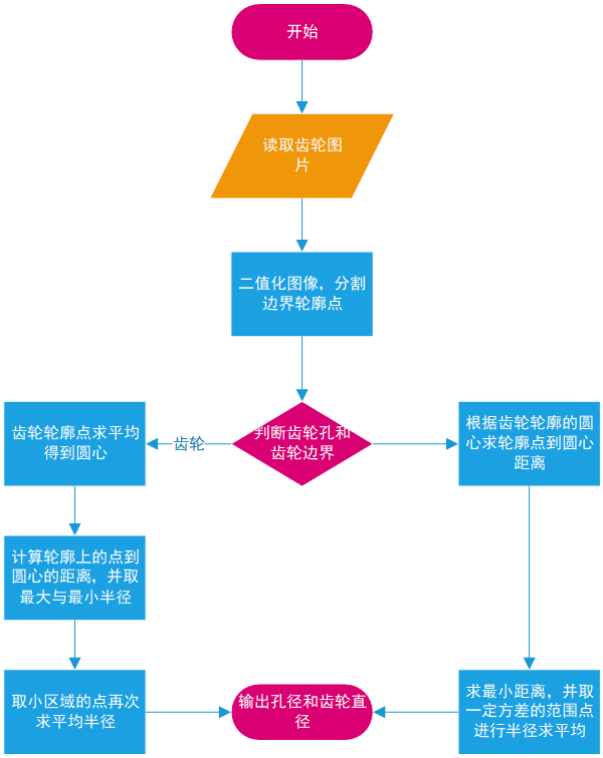
1. 相机内参标定

相机内参标定被封包到 cameraCheck 函数中，主要实现了标定板对内参的标定，输出相机内参和像素距离。



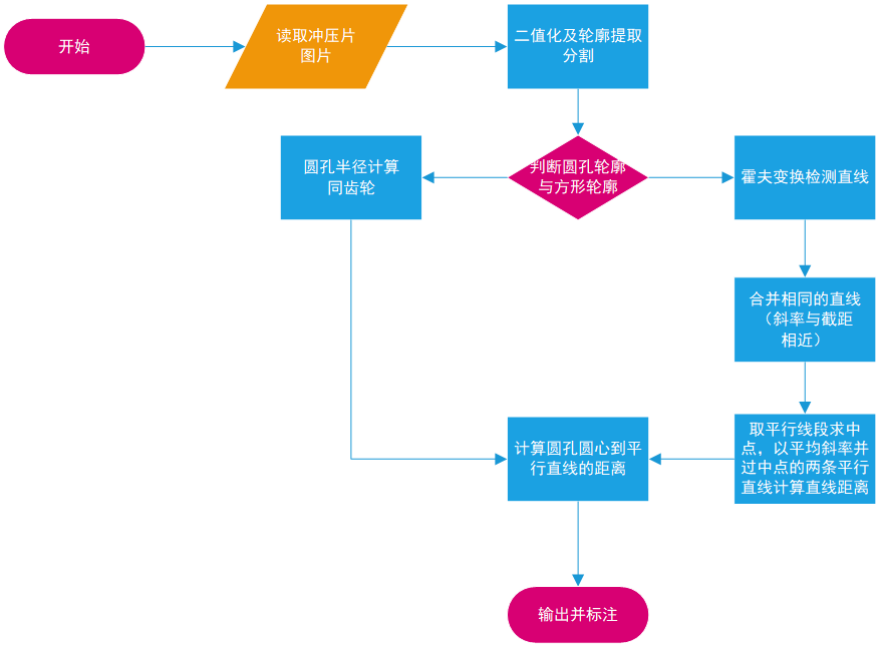
2. 齿轮测量

齿轮测量被封包到Gear_Measure函数中，函数主要实现了**齿轮孔、齿顶圆、齿根圆的视觉测量及其可视化标注**，主要使用**平均圆心、方差去除圆的突出部**的方法。



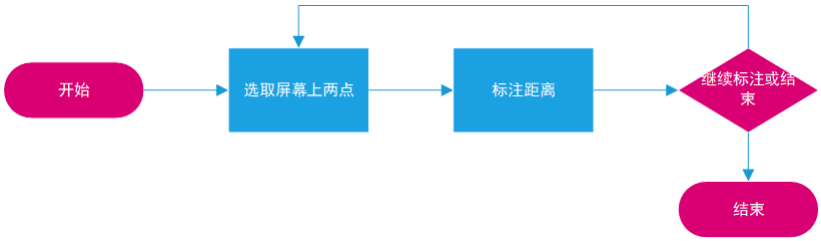
3. 冲压片测量

冲压片测量被封包到Square_Measure函数中，函数实现了**对冲压孔、冲压片长宽以及圆孔位置的测量，并实现可视化标注**，主要使用**霍夫变换检测直线**的方法。



4. 图形化标注

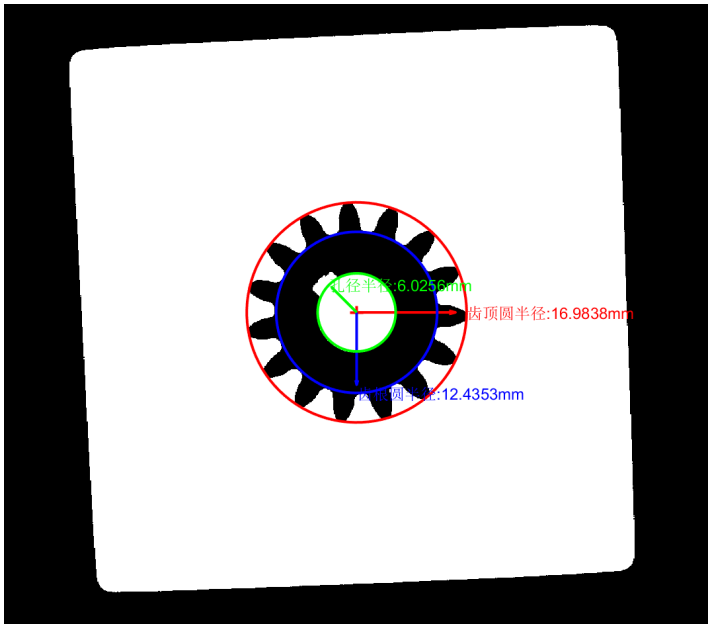
GUI测量被封包在GUI_Measure函数中，可选取图片进行图像两点选取可视化标注。



三、实验结果

1. 齿轮检测结果

可视化标注结果：

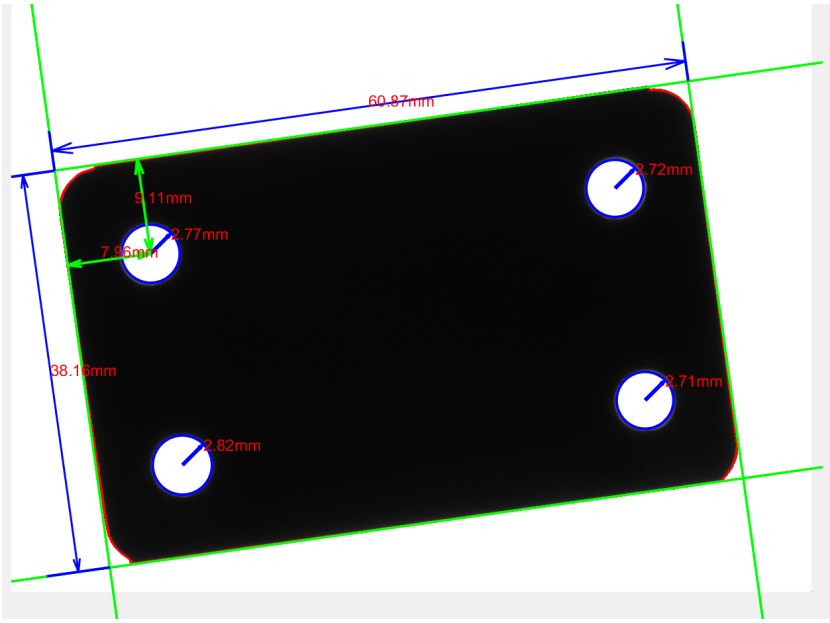


Matlab输出结果：

```
齿轮测量开始
齿轮孔径： 12.05 mm
齿顶圆直径： 33.97 mm
齿根圆直径： 24.87 mm
齿轮测量结束
```

2. 冲压片检测结果

可视化标注结果：

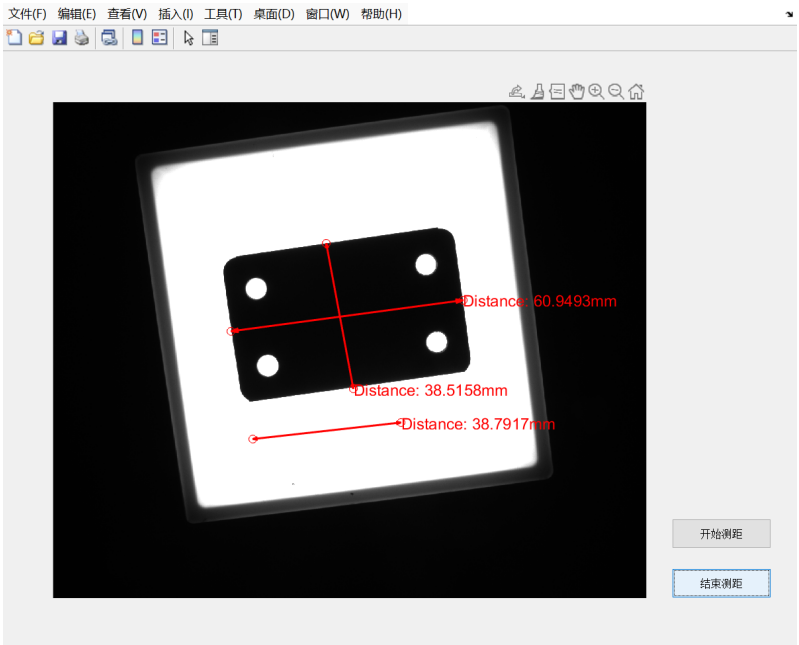


Matlab输出结果：

```
冲压件测量开始
孔 1 直径： 5.54 mm
孔 2 直径： 5.64 mm
孔 3 直径： 5.44 mm
孔 4 直径： 5.41 mm
平行边的距离：
边 1 和 边 3 之间的距离： 38.16 mm
边 2 和 边 4 之间的距离： 60.87 mm
圆心到直线的距离：
圆心到边 1 的距离： 9.11 mm
圆心到边 3 的距离： 7.96 mm
冲压件测量结束
```

3. 图形化标注

可视化标注结果：



四、误差分析

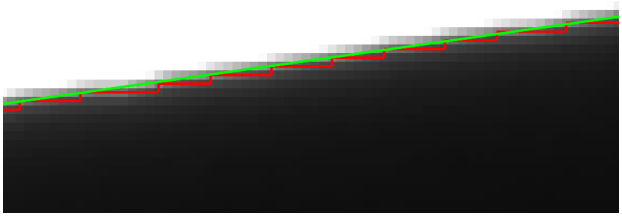
1. 齿轮误差分析

齿轮**误差主要在于齿轮的轮廓是否规律且平整**，这关系到求圆心。因为本算法的依赖于齿轮轮廓点求平均得到圆心坐标，后续的测量均以该圆心作为原点。

除此之外，齿轮孔并不是一个标准的圆，因此**内孔圆心也是依赖于外齿轮的圆心**，计算得到轮廓距离，有一个方差的误差范围以内的轮廓点才会被选入，来计算平均半径，来尽可能减小偏差。

2.冲压片误差分析

冲压片的**误差主要在于冲压片的霍夫变换直线检测**，很显然图片的**边界并非是一条直线而且不平整**，与分辨率、二值化处理和测量件本身有关，**边界其实是一条条锯齿状的直线组成的**，如同下图：



这些直线是水平的，但是在不同的高度上，拟合成了一条直线，因此在霍夫变换的直线检测可能会由于冲压片边缘的不平整而将一条直线检测成多条直线，因此代码里还添加了相同直线合并的相关程序。

为了进一步缩减误差，本代码采用线段中点与斜率平均的方式来减小测量偏差。

五、心得体会

其实也是上完这门课的心得感受吧，几乎应用了所有学习到的知识。

1. 在标定的时候，使用了相机内参标定函数，并且对图像进行去畸变处理；
2. 进一步的对于图像进行二值化处理，并且提取轮廓方便进一步处理，有时也会使用腐蚀、膨胀对于图像进行处理，霍夫变换为了使得直线检测效果更好，进行了一次膨胀腐蚀操作；
3. 学习使用了一般的测量方法，并且应用霍夫变换进行直线检测，也进一步了解到直线计算的误差以及技巧。

六、核心代码

1. Gear_Measure函数，只保留核心部分

```
%%% ---------------
% 2024/9/4 lxy
%%% ---------------
function Gear_Measure(gear_image, distance)
    % gear_image = medfilt2(gear_image);
    % 二值化齿轮图像
    bw = imbinarize(gear_image);
    boundaries = bwboundaries(bw);
    gear_eg = boundaries{3};
    size(gear_eg);
    gear_cir = boundaries{2};
    size(gear_cir);
    % 提取出齿轮圆心
    gear_center = [mean(gear_eg(:,1)), mean(gear_eg(:,2))];
    % gear_center2 = [mean(gear_eg(:,1)), mean(gear_eg(:,2))]
```

```
    % 计算孔径半径
    circle_eg = gear_cir - gear_center;
    min_cir = min(sqrt(circle_eg(:,1).^2 + circle_eg(:,2).^2));
    max_cir = max(sqrt(circle_eg(:,1).^2 + circle_eg(:,2).^2));
    delta = max_cir - min_cir;
    circle_min = circle_eg(sqrt(circle_eg(:,1).^2 + circle_eg(:,2).^2) < min_cir +
0.1*delta, :);
    circle_rd = mean(sqrt(circle_min(:,1).^2 + circle_min(:,2).^2));
    % 计算齿顶圆
    % 提取出齿顶圆的点
    gear_cir = gear_eg - gear_center;
    max_r = max(sqrt(gear_cir(:,1).^2 + gear_cir(:,2).^2));
    min_r = min(sqrt(gear_cir(:,1).^2 + gear_cir(:,2).^2));
    delta = max_r - min_r;
    gear_top = gear_cir(sqrt(gear_cir(:,1).^2 + gear_cir(:,2).^2) > max_r - 0.03*delta,
:);
    gear_min = gear_cir(sqrt(gear_cir(:,1).^2 + gear_cir(:,2).^2) < min_r + 0.03*delta,
:);
    % 计算齿顶圆半径
    top_rd = mean(sqrt(gear_top(:,1).^2 + gear_top(:,2).^2));
    min_rd = mean(sqrt(gear_min(:,1).^2 + gear_min(:,2).^2));
end
```

2. Square_Measure函数，只保留核心部分

```
%%% ---------------
% 2024/10/6 lxy
%%% ---------------
function Square_Measure(square_image,distance)
    % 边缘检测
    edges = edge(grayImg, 'Canny');
    % 霍夫变换提取直线
    [H, T, R] = hough(cleanedEdges);
    % plot(H);
    peaks = houghpeaks(H, 4, 'threshold',ceil(0.15*max(H(:)))); % 获取前4个峰值
    lines = houghlines(cleanedEdges, T, R, peaks);
    % 提取直线参数
    lineParams = zeros(length(lines), 4);
    for k = 1:length(lines)
        % 计算直线方程参数
        x1 = lines(k).point1(1);
        y1 = lines(k).point1(2);
        x2 = lines(k).point2(1);
        y2 = lines(k).point2(2);
        % 线的斜率和截距
        if x1 ~= x2
            slope = (y2 - y1) / (x2 - x1);
            intercept = y1 - slope * x1;
            lineParams(k, :) = [slope, intercept, 1, k]; % 线的参数 (斜率, 截距, 类型, 索
引)
        else
            % 垂直线
            lineParams(k, :) = [Inf, x1, 0, k]; % x = const
        end
    end
    deleteIndex = [];
    % 判断是否有斜率截距相近的直线，将其合并
    for i = 1:length(lines)
```

```
        for j = i+1:length(lines)
            % 计算两条直线的斜率和截距之差
            diff = abs(lineParams(i, 1) - lineParams(j, 1)) + abs(lineParams(i, 2) -
lineParams(j, 2));
            if diff < 10
                % 合并两条直线，取平均值
                lineParams(i, 1) = (lineParams(i, 1) + lineParams(j, 1)) / 2;
                lineParams(i, 2) = (lineParams(i, 2) + lineParams(j, 2)) / 2;
                deleteIndex = [deleteIndex, j];
            end
        end
    end
    % 删除合并的直线
    lineParams(deleteIndex, :) = [];
    lines(deleteIndex) = [];
    for i = 1:length(lineParams)
        lineParams(i, 4) = i;
    end
    % 计算平行边的距离
    distances = zeros(length(lineParams)/2, 1);
    % 将直线按斜率排序
    lineParams = sortrows(lineParams, 1);
    % 计算两两直线之间的距离
    for i = 1:2
        slope1 = lineParams(i*2-1, 1);
        slope2 = lineParams(i*2, 1);
        % 根据线的类型计算距离
        if slope1 ~= Inf && slope2 ~= Inf
            % 两条非垂直直线
            % d = abs(intercept2 - intercept1) / sqrt(1 + slope1^2);
            x1 = (lines(lineParams(i*2-1,4)).point1(1)+lines(lineParams(i*2-
1,4)).point2(1))/2;
            y1 = (lines(lineParams(i*2-1,4)).point1(2)+lines(lineParams(i*2-
1,4)).point2(2))/2;
            x2 =
(lines(lineParams(i*2,4)).point1(1)+lines(lineParams(i*2,4)).point2(1))/2;
            y2 =
(lines(lineParams(i*2,4)).point1(2)+lines(lineParams(i*2,4)).point2(2))/2;
            k = (slope1+slope2)/2;
            A1 = (slope1+slope2)/2;
            B1 = -1;
            C1 = -k*x1+y1;
            C2 = -k*x2+y2;
            d=abs(C2-C1)/sqrt(A1^2+B1^2);
        else
            % 垂直线之间的距离
            d = abs(lineParams(i, 2) - lineParams(i+2, 2));
        end

        distances(i) = d;
    end
    % 计算相邻边的4个交点
    crosspoints = zeros(4, 2);
    point_lin = [[1,3];[1,4];[2,4];[2,3]];
    for i = 1:4
        index1 = point_lin(i,1);
        index2 = point_lin(i,2);
        slope1 = lineParams(index1, 1);
        intercept1 = lineParams(index1, 2);

        slope2 = lineParams(index2, 1);
```

```
        intercept2 = lineParams(index2, 2);

        % 计算交点
        if slope1 ~= Inf && slope2 ~= Inf
            % 两条非垂直直线
            x = (intercept2 - intercept1) / (slope1 - slope2);
            y = slope1 * x + intercept1;
        else
            % 一条垂直线
            x = intercept1;
            y = slope2 * x + intercept2;
        end
        crosspoints(i, :) = [x, y];
    end
end
```