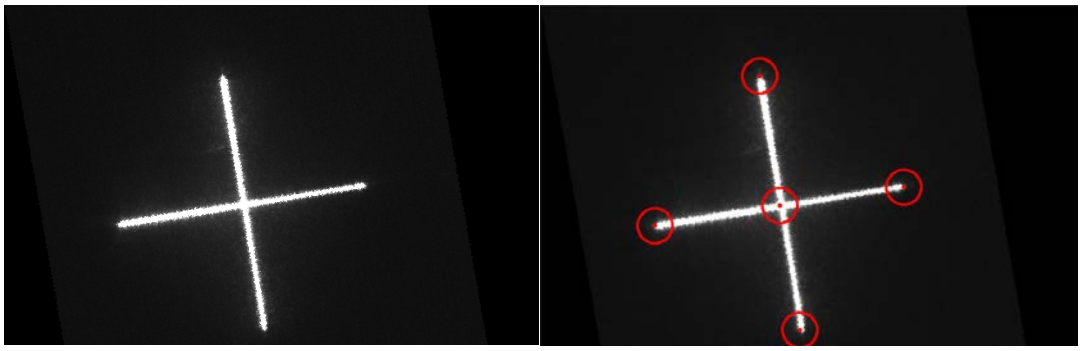
条码识别图像中心区域关键点识别算法

一. 算法背景和目的

对于 CM46 采集到的中心区域图像，需要采用准确而快速的算法来实现对关键点（光亮十字的四个端点和一个中心点）的探测，并输出其对应图像坐标。该算法应实现类似如下的效果：（左图为算法的输入数据，右图标识的五个关键点坐标为算法的输出）。算法应具有一定的准确度，并且程序对于单张图像的识别时间应小于 20ms。



二. 算法实现步骤

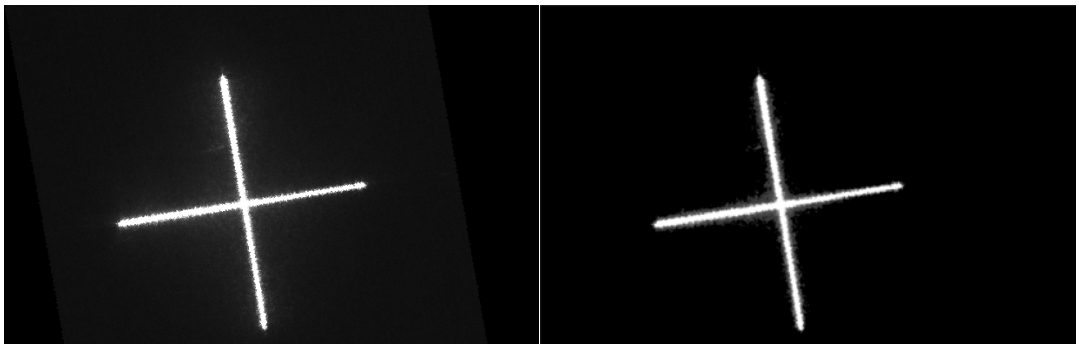
具体实现步骤如下。

1.灰度处理

对输入图像 src 进行灰度处理，转化为单通道的灰度值图像 src_gray。该步骤采用 OpenCV 库中成熟的灰度处理函数 cvtColor(src, src_gray, CV_RGB2GRAY);

2.阈值化处理

对 src_gray 图像进行阈值化处理，对于小于阈值的像素点，其灰度值将被设置为 0。该操作可消除噪音，得到去除大部分噪点后的图像 after_threshold。对于给予的 CM46 中心区域图像训练集，我们发现可分为噪点较多和噪点较小的原始图像，对于噪点图像的分类采用编写的 count_noise_point 函数实现。对噪点较多的原始图像，可使用较大的 threshold_value2，可去除更多的噪点，加快后续的图像遍历和处理。而对于噪点较小的原始图像，需采用较小的 threshold_value，以免灰度值不大的关键点被阈值化处理。实现效果如下。

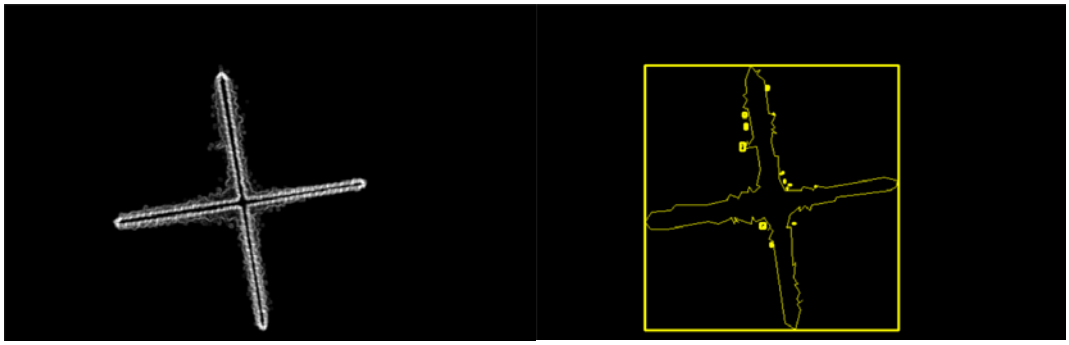


3.梯度计算

利用 sobel_derivatives 函数，对图像 after_threshold 中梯度变化较大的图像边界值进行提取，得到轮廓（contour）明显的图像 after_sobel。实现效果如下。

4. Creating Bounding boxes and for contours

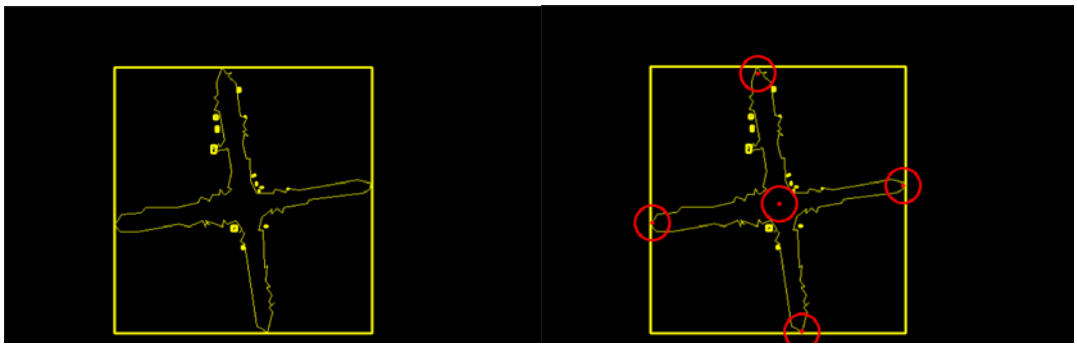
对于图像 after_sobel，采用内置的 boundingRect 函数，可对图像的主要轮廓进行检测，并自动寻找最小可包围主轮廓的矩形，得到图像 drawing。故该函数在官方文档中称为 Creating Bounding boxes and for contours。实现效果如下：



5.查找关键端点并绘制中心点

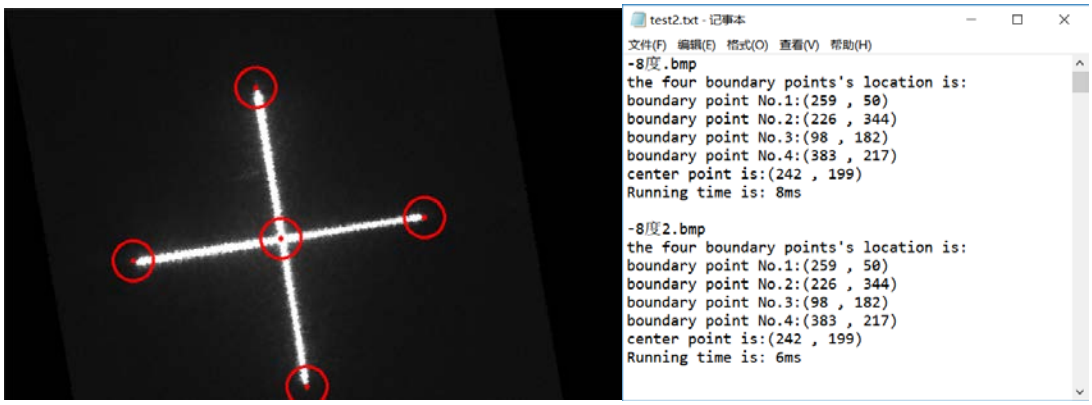
对于上述图像 drawing,程序将记录矩形四边的位置。对图像 after_threshold 进行像素点遍历，对于其中像素值大于 thresh_detect_image 的点，寻找最靠近矩形四边的四个十字端点。

得到四个十字端点后，基于该四点绘制两条交叉直线，他们的交点即被设置为探测到的中心点。实现效果如下：



6.结果输出

记录五个关键点坐标，并在 src 图像上绘制出来。并在 txt 文件中输出图像的文件名、五个关键点坐标、程序运行时间。实现效果如下。



三. 算法实现关键点

1.对于噪点较大和较小的图像分类

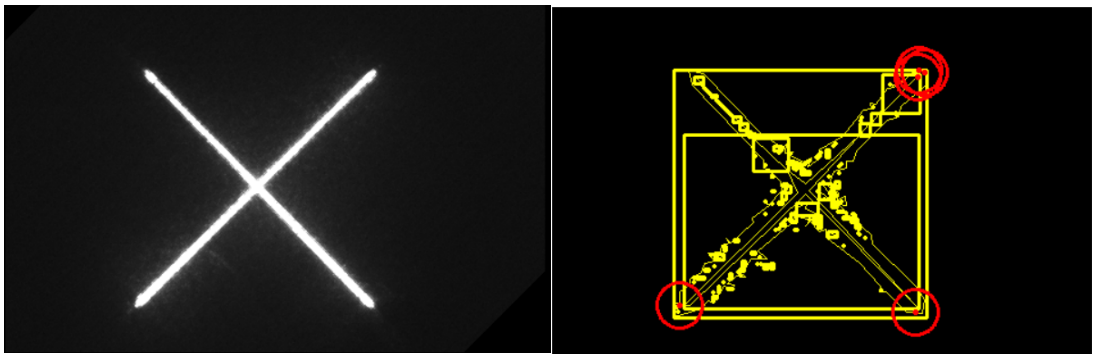
在本算法实现的最初阶段，全部图像都采用同一个较小的 threshold_value 进行阈值化处理。在调试中发现对于较大噪声的训练图像，图像在该步骤后仍然存在较多的噪声，导致在后期的图像处理中程序运行速度缓慢，并导致 sobel 函数和 boundingRect 函数的成像效果差，大大减小了关键点的准确度。因此在灰度处理之后，将灰度图像立即输入到噪声图像分类函数 count_noise_point 中。

在该函数中，遍历图像所有的像素点，对其中大于 threshold_for_count 值的像素点进行计数（count 值），随后进行判断：若 count 大于（src_gray.rows*src_gray.cols*rate），则判断输入图像为噪点较大图像，在随后的阈值化处理中采用较大的 threshold_value2。否则采用 threshold_value。其中 threshold_for_count 和 rate 值为可调试参数。为了兼顾准确性和快速

性，分别设置为 160 和 0.028。

2. 对于 45 度图像的重检测

在实现步骤 5 中，我们发现程序对于 45 度图像的识别将出现错误。错误的识别图像如下：



识别原因是当图像为 45 度时，将可能出现十字的某端上同时距离矩形两边距离最小的情况。在该情况下算法需对图像进行重检测。

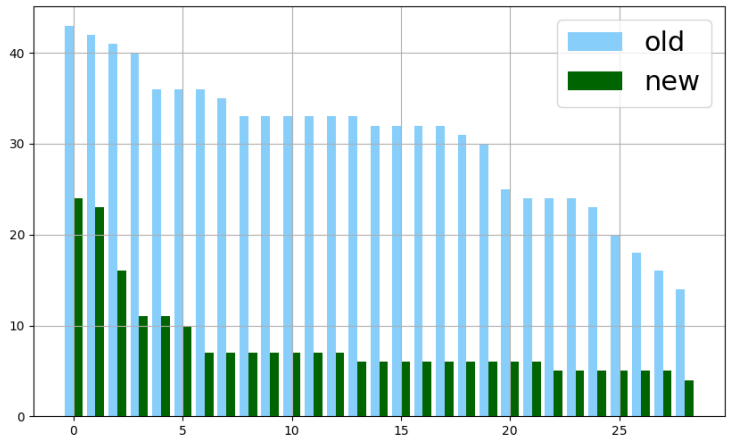
我们对探测到的关键点进行判断，若是有任两个关键点之间小于一定的距离 min_45distance，则判断当前图像为 45 度，进入重新检测函数。

我们采用更加贴合 45 度十字的 minAreaRect 函数，在允许矩形可以旋转的前提下，绘制出可包围主轮廓的最小矩形。并采用新的判断端点方法：对主轮廓像素点进行遍历，将最靠近矩形四个顶点的四个像素点判断为端点，并根据该四点绘制两交叉直线，交点为中心点。实现效果如下：

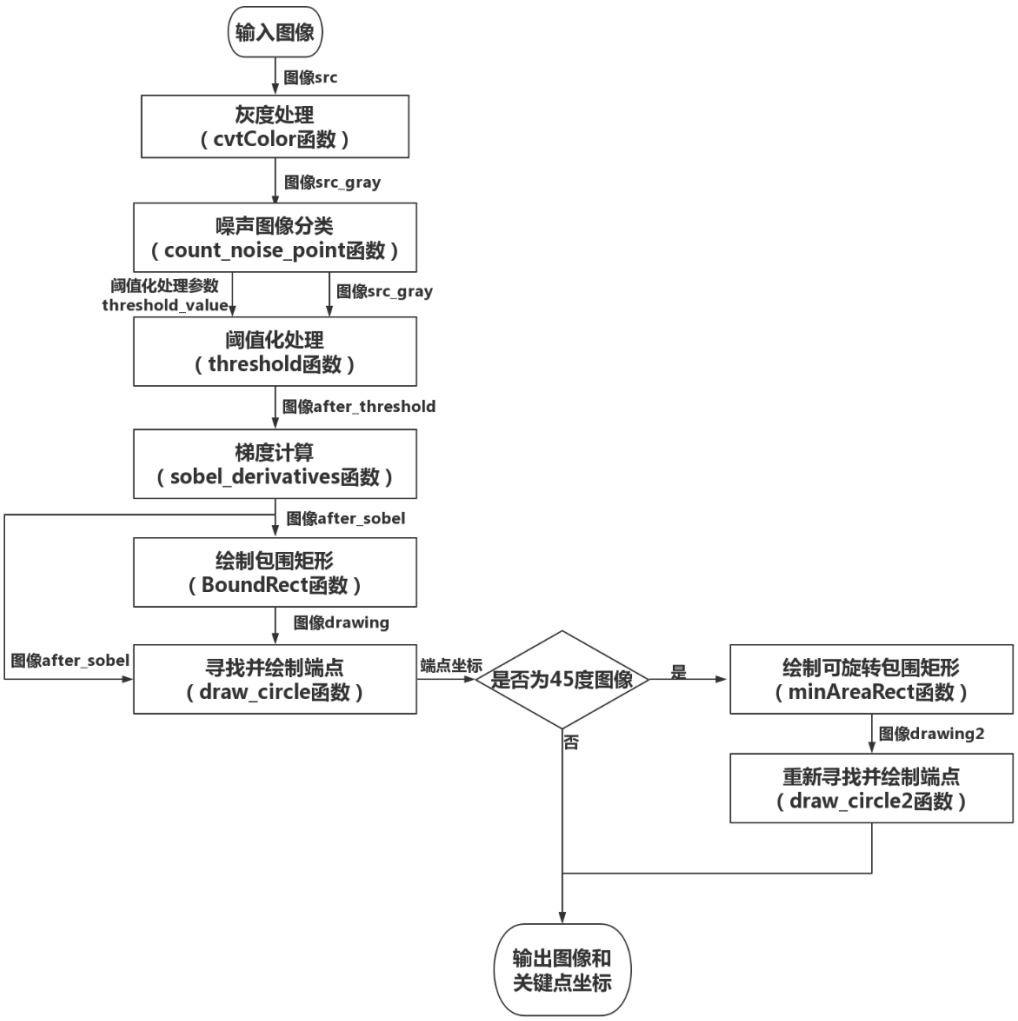
3.采用指针的方式进行遍历

在程序的运行过程中，最初使用的是 image.at(row, col)函数对图像的像素点进行遍历。该遍历方法可使得代码简洁可读性高，但运行速度缓慢。在调试算法的后期，我们改为采用指针的方式实现所有的像素遍历操作。运行速度提高至 2-4 倍。下图为采用指针前和采用指

针后对于 29 张训练图像的运行时间（单位为 ms）对比：



四. 算法设计流程图



五. 运行结果

算法的精度较高，可准确识别出所有 29 张训练图像的关键点。

算法的运行速度较快，所有的图片的识别时间均在 20ms 以内。下图为最终算法在 29 张训练图像上的测试时间（单位为 ms）：