Deniz Sökmen
S003244
Computer Vision
16/01/2015

DICE RECOGNIZER

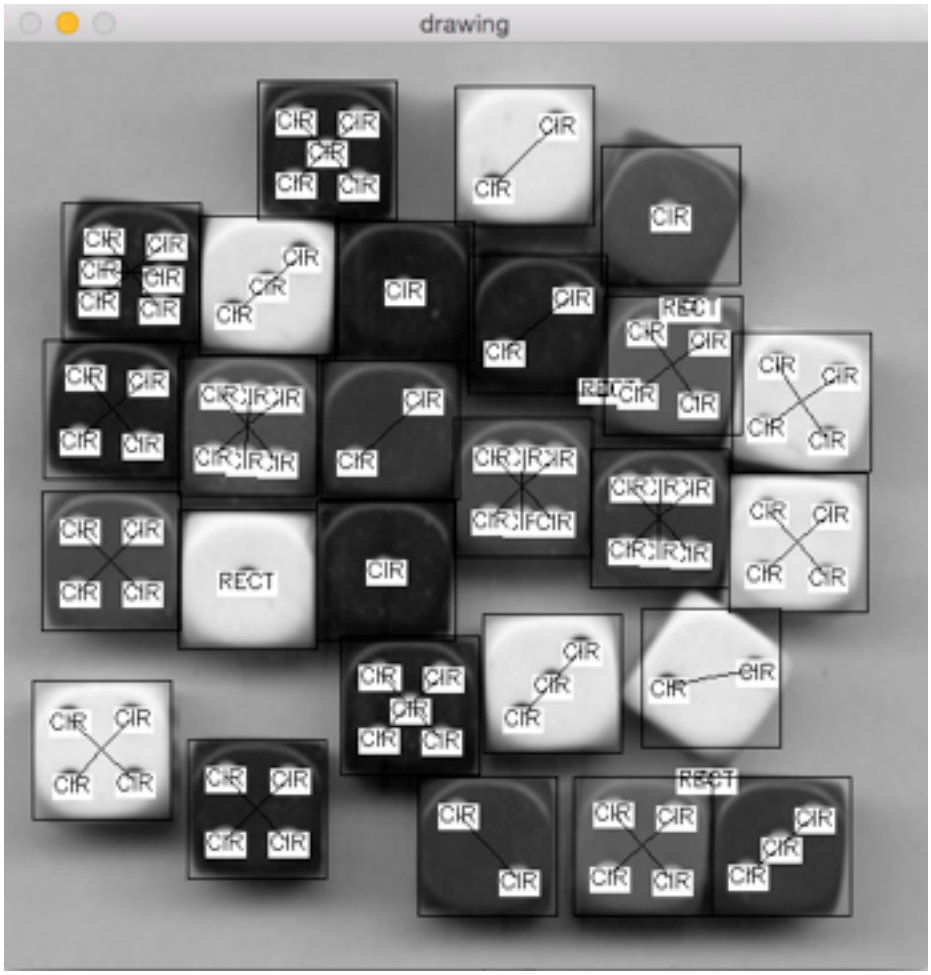
In this report, I will talk about the methods I have tried in take-home exam for dice recognition. It seemed pretty easy at the first sight but as my dataset expanded, I have realized that the techniques I had used in my first solution were not robust to different parameters.

The very first technique I have used was canny edge detection, because it could handle the noisy background for some images. Then I thought that I could use floodFill function of OpenCV to fill every hole in the image and then sort the areas I have filled by their sizes so that I could accept every small sized area as dots. Before I use floodFill, I used dilation so that I would not get scattered circles from Canny edge detection. At first, it worked and I was able to get number of dots in an image. The next step was to use mean-shift clustering (k-means would not work as I had very few information about the image). But the problem here was that I had no information about the sizes of dice. Therefore this solution was not robust to different sizes. I needed more information, not only the number of dots.

Basically, clustering and edge detection were not enough and robust for reading dice. Then I realized that the dice are clusters themselves already, so I looked for a technique to recognize rectangles on the image. Canny was helpful again at that point. After canny, I tried to apply corner detection but quickly abandoned that idea. By the time I have realized that canny was not enough for removing the background noises so I applied OTSU threshold on the image before using canny. I have found out that there is a technique in OpenCV to recognize various shapes: contours and shape descriptors. It was able to extract vertices from polygons and put bounding boxes on them. So basically four vertices meant that the shape was possibly a rectangle, in order to check that I calculated the cosine of the angle between edges. A perfect rectangle should have 90 degrees between it's edges, but as the dice are not perfect rectangles and I set an arbitrary threshold to allow close values. I have also accepted shapes with more than 6 vertices as circles. After that point, I have decided to approach the problem differently. Rectangles were the clusters and circles were the dots on the dice. Using that information, I was able to count the dice correctly and it was robust to size and rotation. I simply traversed every circle and mapped them in appropriate rectangles they are in. But another problem raised at that point. Applying canny on the hard example that is provided in the assignment description was not able to find edges of the most of the dice. Therefore the algorithm fails to detect squares and the whole algorithm crumbles because of the free circles that do not belong in any rectangle. Even in the easy example canny would fail to detect a closed shape.

This time, I have not given up on this technique and instead of overfitting the parameters according to specific examples, I have made the algorithm much more robust. To recall the first solution I have tried in the second paragraph, the problem in that solution was the lack of information like the size of dice. But with this technique, I was able to find the size for at least one of the dice. So this is the step two for the last technique I have used, this step is only used if there are free circles which are not in any rectangles, therefore still need to be clustered. Basically this step is kind of a post-processing. I implemented a simple custom clustering based on that information. Remember that canny was not able to detect all of the edges for rectangles and because of that there were free circles that are not in any rectangle. At first, I was not able to cluster the circles but now I have the bounding box for at least one die (it does not matter which one). The simple clustering I have written works like that: Pick middle point of a free circle, compare it with the average of the points in each cluster (create a cluster if there is no cluster and put it in), if the distance between the picked point and the average of the cluster is less than half of the diagonal of a die's bounding box, add it in that cluster, if not, put it in a new cluster.

Now that I can create rectangles that I was not able to create using canny and shape descriptors. I have clusters of circles now, I take average point of each cluster and simply put a rectangle that is centered on that point and has a size of any die on the image. Those new rectangles will contain the free circles. Lastly, traverse free circles and put them in appropriate rectangles just like in step 1.



Finally, all of the circles are mapped to a rectangle. If there are still free circles, it is due to the bias of the algorithm and necessity of a fine-tuning on thresholds.

I am able to guess close number of dots on dice as long as there is at least one rectangle the algorithm can recognize. There are of course a lot of work to do to make the algorithm more robust

to rotation. Because currently it needs at least one die that is not much rotated in order to find it's bounding box.

References

Tutorial for detecting simple shapes using contour and shape descriptors, Retrieved from:

https://opencv-code.com/tutorials/detecting-simple-shapes-in-an-image/