Programming assignment report

520030910342 柳纪宇

1. Problem 1

1.1. binary image

Taking 'two_objects.png' as input, choose a threshold of 128 to construct a binary image. For those pixel whose value is less than the threshold, I turn them to 0. Otherwise, I turn the value to 255.

Save the picture as 'two_objects_bianry.png'. The binary image is shown in Picture 1.

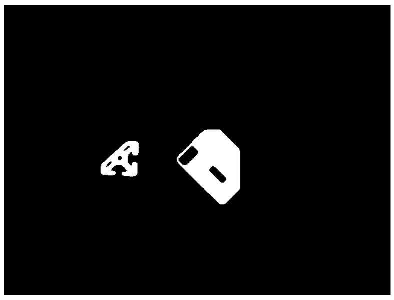

图 1: *binary image*

1.2. labeled image

In this part, I write a two-pass method to find connected components in a binary image.

I create a class Union_Find with function find() and union(), which is used for the union of labels with different values but belong to the same component in the second pass.

In the algorithm, first, I pad the binary image with a circle of zeros. Then we go through the first pass by traversing the numpy array and check three neighbors of a point img(x,y) img(x-1,y), img(x-1,y-1) and img(x,y-1).

If img(x,y) is 0, then skip to the next loop.

If img(x,y) is not 0 and three neighbors are all zeros, then we label (x,y) as the current label and let label += 1.

If img(x,y) is not 0 and img(x-1,y-1) is not 0, then we let img(x,y) = img(x-1,y-1).

If img(x,y) is not 0 and only img(x-1,y) or img(x,y-1) is not 0 in the three neighbors, then we let img(x,y) = img(x-1,y) or img(x,y-1).

If img(x,y) is not 0 and img(x-1,y) and img(x,y-1) are both not 0 but with different values. Then we let img(x,y) be img(x-1,y) and do a union operation in Union_Find with img(x-1,y) and img(x,y-1).

Then we go through the second pass: traversing the image and converting the value of each pixel to its root in the Union_Find.

Finally, we adjust the pixel value of the output image to make it a differentiable gray image and save it as 'two_objects_gray.png'. The labeled image is shown in Picture 2.

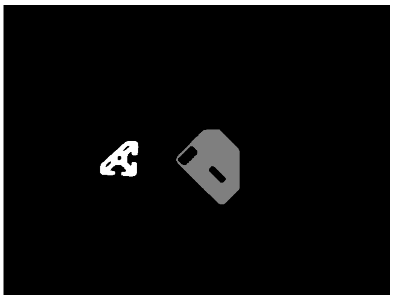

图 2: *labeled image*

1.3. attribute list

In this part, we iterate all component values. For each value, we are going to find all pixels with this value by traversing the image array to get x_list and y_list which contain the position of each point in this component.

Then we can get to position of this component by simply computing the average number of x_list and y_list.

To get the orientation of the component, we need to iterate through x_list and y_list to compute a,b,c. By the equation $\theta_1 = atan2(b/(a-c))/2$, we compute the orientation with E_{min}.

Let $\theta_2 = \theta_1 + \pi/2$, then compute E with $\theta = \theta_2$ to get E_{max}. Then we let roundedness $= E_{min}/E_{max}$.

My output list is as below:

['position': (349.33298470388286, 215.45365407242775), 'orientation': 0.31083980333269323, 'roundedness': 0.5336319534756389, 'position': (195.3160469667319, 222.3820939334638), 'orientation': 0.6875462936637149, 'roundedness': 0.4799636466920348]

2. Problem 2

2.1. detect edges

In this part, I use two 3x3 sobel kernels to convolve with padded image to get two values G_x and G_y. Then let the magnitude of a pixel be the sum of the absolute value of G_x and G_y in this pixel. After getting a magnitude array, I use matplotlib to draw a heat map of the 2-dim numpy array, saved as 'coins_edge_magnitude.png'. The image is shown as Picture 3.

The kernel values in the sobel edge detector are

$$\begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

for G_x and G_y respectively.

2.2. hough circles

First, I use a threshold value to decide whether a pixel belongs to edges or not. After trying some values, finally I choose **400** to be the threshold value, because with it we won't loss too much information and can still remove most of the noise. The edge image is shown as Picture 4 and be saved as 'coin_edges.png'. Then I construct an accumulator array to decide where is the center of circles are and what is their radius. For each pixel(x0, y0) marked as edge point, I traverse every point in the circle centered around itself with certain radius value, and add one to accum_array[radius][y0][x0]. After we go through every radius value in the radius value list, we get an complete accumulator array.

The accumulator array is too long and the vast majority of its values is zero, so I am not going to print it here.

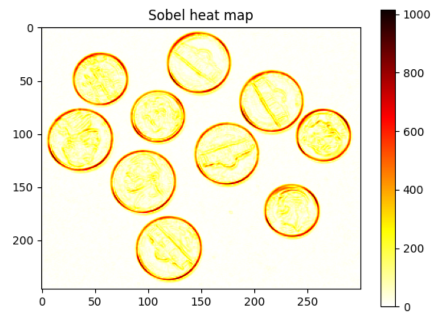

图 3: *Magnitude heat image*

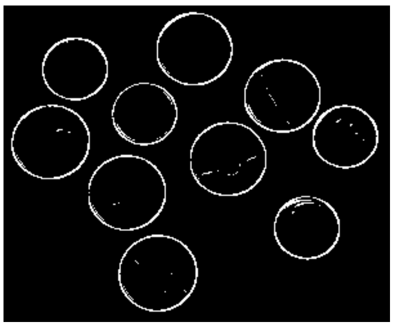
图 4: *edge image*

2.3. find circles

In this part, I find circles by applying hough threshold on the accumulator array. For those who are larger than the threshold, I take down their position and correspond radius, combining them it a tuple and appending it to a list.

Then I use cv2.circle() to draw those circles in the origin image, saved as 'coins.circles.png'. The image is shown as Picture 5. The hough threshold I set is 220.

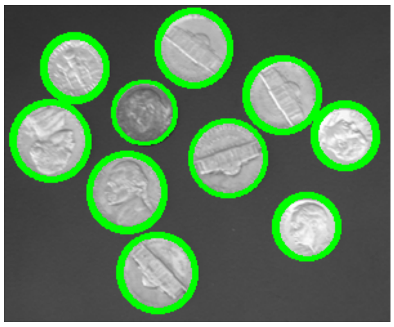

图 5: *Magnitude heat image*