Department of Electrical Engineering
2022Spring - Computer Vision

Program Assignment 2

Prepared by: Wang
Instructor: Yang
Date: April 25, 2022

Abstract

The Hough transform (HT) is a technique that locates shapes in images. In particular, it has been used to extract lines, circles, and ellipses (or conic sections). In this assignment, I use Gaussian Blurring of Gray Image and OpenCV Canny Edge detector to detect edges as pre-processing. Then I use HT to detect coins in two figures based on the Mid-Point circle drawing algorithm. The implementation shows that the performance of results is dependent on the parameters of the threshold and the range of the radius.

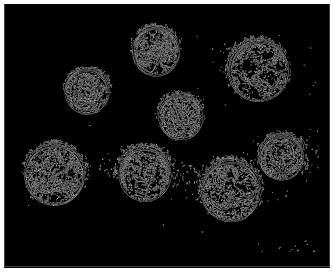
Contents

The Hough Transform is a popular feature extraction technique that converts an image from Cartesian to polar coordinates. Any point within the image space is represented by a sinusoidal curve in the Hough space. In addition, two points in a line segment generate two curves, which are overlaid at a location that corresponds with a line through the image space. Even though this model form is very easy, it is deeply complicated for the case of complex shapes due to noise and shape imperfection, as well as the problem of finding slopes of vertical lines.

1 Pre-processing

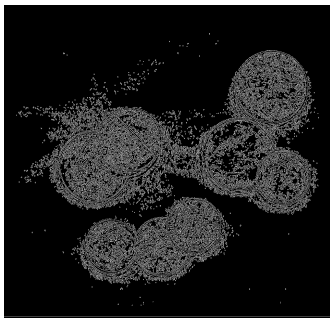
Firstly, I use Gaussian Blurring of Gray Image and OpenCV Canny Edge detector to detect edges as shown in Fig. 1.1 .

(a) The G-B image of example 1.



(b) The canny edged image of example 1.

(c) The G-B image of example 2.



(d) The canny edged image of example 2.

Figure 1.1: Pre-processing

2 Hough Transform Application in Circle Detection

A circle with a constant radius is corresponding to one point(i.e. the coordinates of the circle center) in hough space. Therefore, a point on the circle in image space is corresponding to a circle respectively. To ensure a circle, we need to draw dozens of circles in hough space to vote the most possible circle center. First of all, we need to draw the circles.

2.1 Circle Drawing Algorithm

It is not easy to display a continuous smooth arc on the computer screen as our computer screen is made of pixels organized in matrix form. So, to draw a circle on a computer screen we should always choose the nearest pixels from a printed pixel so as

they could form an arc. There are two algorithms to do this:

1. Mid-Point circle drawing algorithm.

2. Bresenham's circle drawing algorithm

Both of these algorithms use the key feature of a circle that is highly symmetric. So, for the whole 360 degrees of circle we will divide it into 8-parts each octant of 45 degrees. I write two sub-functions as shown below

```
def fill_acc_array_MidPoint(x0, y0, radius):
    x = radius
    y = 0
    while (y < x):
        if (x + x0 < height and y + y0 < width):
            acc_array[x + x0, y + y0, radius] += 1;  # Octant 1
        if (y + x0 < height and x + y0 < width):
            acc_array[y + x0, x + y0, radius] += 1;  # Octant 2
        if (-x + x0 < height and y + y0 < width):
            acc_array[-x + x0, y + y0, radius] += 1;  # Octant 4
        if (-y + x0 < height and x + y0 < width):
            acc_array[-y + x0, x + y0, radius] += 1;  # Octant 3
        if (-x + x0 < height and -y + y0 < width):
            acc_array[-x + x0, -y + y0, radius] += 1;  # Octant 5
        if (-y + x0 < height and -x + y0 < width):
            acc_array[-y + x0, -x + y0, radius] += 1;  # Octant 6
        if (x + x0 < height and -y + y0 < width):
            acc_array[x + x0, -y + y0, radius] += 1;  # Octant 8
        if (y + x0 < height and -x + y0 < width):
            acc_array[y + x0, -x + y0, radius] += 1;  # Octant 7
        y += 1
        decision = np.sqrt(radius**2 -y**2)-(x-0.5)
        if (decision < 0):
            x +=- 1
```

Listing 2.1: Mid-point algorithm

```
def fill_acc_array_Bresenham(x0,y0,radius):
    x = radius
    y=0
    decision = 1-x
    while(y<x):
        if(x + x0<height and y + y0<width):
            acc_array[ x + x0,y + y0,radius]+=1; # Octant 1
        if(y + x0<height and x + y0<width):
            acc_array[ y + x0,x + y0,radius]+=1; # Octant 2
        if(-x + x0<height and y + y0<width):
            acc_array[-x + x0,y + y0,radius]+=1; # Octant 4
        if(-y + x0<height and x + y0<width):
            acc_array[-y + x0,x + y0,radius]+=1; # Octant 3
        if(-x + x0<height and -y + y0<width):
            acc_array[-x + x0,-y + y0,radius]+=1; # Octant 5
        if(-y + x0<height and -x + y0<width):
            acc_array[-y + x0,-x + y0,radius]+=1; # Octant 6
        if(x + x0<height and -y + y0<width):
            acc_array[ x + x0,-y + y0,radius]+=1; # Octant 8
        if(y + x0<height and -x + y0<width):
            acc_array[ y + x0,-x + y0,radius]+=1; # Octant 7
        y+=1
        if(decision<=0):
            decision += 2 * y + 1
        else:
            x=x-1
            decision += 2 * (y - x) + 1
```

Listing 2.2: Bresenham algorithm

2.2 Implementation

In this implementation, I choose to use the Mid-Point algorithm to draw circles and it takes a lot of time to draw circles in hough space. Therefore, I saved hough space as *acc_array.npy* after processing. The visualization of the circle drawing algorithm is attached in Movie S1.

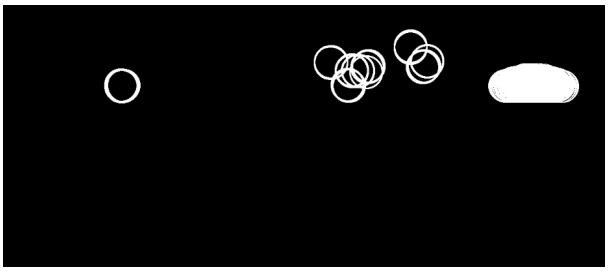


Figure 2.1: Screenshot of Movie S1

After drawing the circle on every edge point, we need to select the most crossed point of those circles as it is voted as the center of the circle in the image space. We draw those points with radius as **the detected circle**.

I set the *circleradius* from 25 to 55 meanwhile the $VotingThresholds$ is 90. The result of example 2 is shown in Fig 2.2 :

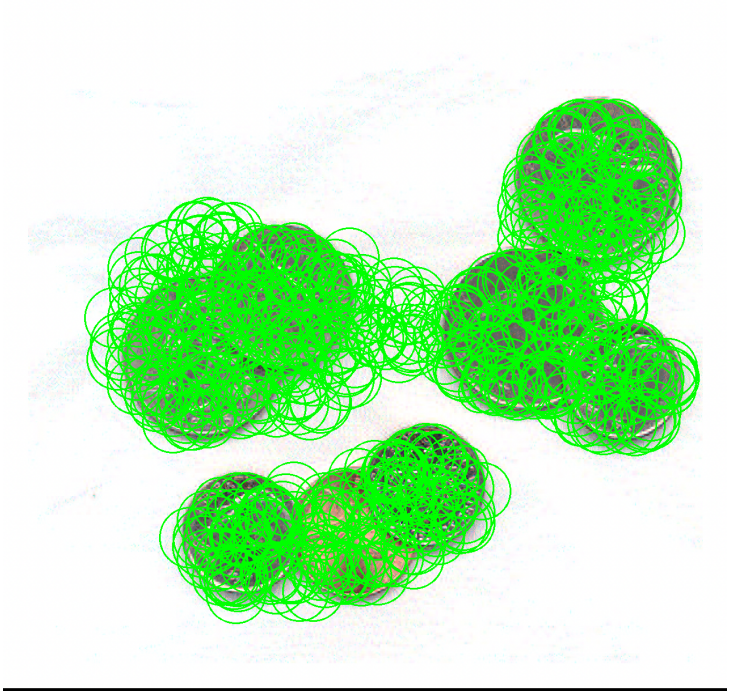


Figure 2.2: Test of example 2

The result is not ideal because of two reasons :

1. The radius of circles is not included in the range.

2. The voting threshold is not high enough.

Then I set the radius from 100 to 150 and run the program. After processing, I combine the hough space with the previous *acc_array.npy* file. After trying different thresholds, the best result is equal to 280 as shown in Fig: 2.3

Figure 2.3: Result of example 2

The example 1 is much simpler, the optimal parameters are radius from 50 to 140 and 245 for threshold as illustrated in Fig 2.4.

Figure 2.4: Result of example 1

3 Conclusion

I draw the conclusion with three points :

1. The performance of results depended on the parameters of the threshold and the range of the radius.

2. It takes five hours or more to draw circles in hough space. However, the CPU usage is only 13%. Rewriting the recurrent circle drawing as parallel processing would accelerate the process.

3. Detecting circles, lines, and rectangles are the fundament. As the saying goes: Great oaks from little acorns grow. We acknowledge enough basic features. The HT algorithm is able to detect many complicated objects such as motorcycles, cars, people, and so on.

List of Figures

List of Listings