CSE 573 Final Project Report

Hengtong Zhang *and* Kun Yang
Department of Computer Science and Engineering
University at Buffalo
Buffalo, NY 14260
{hengtong, kunyang}@buffalo.edu

1 Introduction

Hough transform is a feature extraction technique used in computer vision, and digital image processing. The main advantage of Hough transform is its capability to detect shape of the objects, even if they are occluded. According to [1], Hough Transform usually consists of the following steps:

1. Preprocessing input image;
2. Detect edge using some filters;
3. Construct a look-up table for circles, construct and update accumulator;
4. Select possible circles from accumulator;
5. Last display the result

In this project, we are given an image with coins of different sizes (few are overlapped). The task is to detect and display circles around the boundary of the coins.

2 Your Approach

2.1 Technique

The overall techniques could be outlined as following:

(a) Denoise the image using a Gaussian blur filter of 3x3 size

For denoising, we use the following filter:

$$\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

and get the following result (Figure 1):

(b & c) Apply an edge detector of your choice using external libraries, and threshold the image from (b) into a binary image and report the best threshold value

We use *cv2.Canny()* for edge detection and get the edge image (Figure 2):

(d) Pick a suitable range of radii values of circles to be detected and report why/how you picked this range

We choose $10 \sim 60$ for raddi values. By looking into the image, we find that the smallest circle in the image has radius of 19, the largest circle in the image has radius of 51. So we choose $10 \sim 60$, which can cover all possible circles. Obviously, the amount of computation can be reduced if we have smaller range.

(e & f) Apply Hough transform[1][2] to detect circle at every edge pixel from (c) and update the accumulator array.

Figure 1: Denoised Image

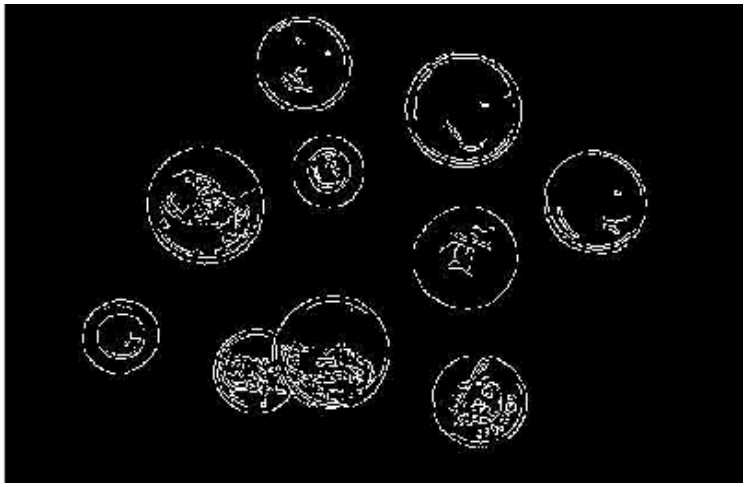

Figure 2:

We divided the system into five parts: preprocessing input image, construct a look-up table for circles, construct and update accumulator, select possible circles from accumulator, and last display the result. Part one and part five are straightforward. We will explain other three in details.

Construct a look-up table for circles: in circle detection using Hough transforms, the major part of computation is updating the accumulator. For every edge pixel, we have to increment a set of circumference points, if we have to re-calculate circumference points for every edge pixel, the run time of our program will be unpractical. Since the relation between a center point and its circumference points are fixed, we can construct a look-up table to accrete the processing of updating accumulator. In our program, we calculate the circumference points for center in (0, 0) with different radius value.

Construct and update accumulator: since we already construct the look-up table, this step is trivia. For every edge pixel and every radius value, we use the look-up table to find the correspond positions in accumulator. Using the look-up table we only need two addition operations to find a circumference point. So the run time is $O(E \cdot R \cdot C)$, where E is the number of edge points, R is the range of circle radius in the image, C is the average number of circumference points.

Select possible circles from accumulator: Ideally, we can choose the highest values in the accumulator to be the circle. However, the circles in the image are not perfect. There could be multiple high

values around the same spot. Without proper selection algorithm, we will get multiple circles for a single coin. To remove the redundant circles, we used a simple criterion to keep the best circles. For every potential circle, we check whether there is another circle within certain range that has higher value in the accumulator, if yes, we remove the weaker one. This algorithm ensures we only obtain one best circle for every coin.

2.2 Software and other methodologies employed

In this project, we use the *convolve2d* and *Canny filter* in OpenCV. Those are allowed according to the project description. We do not use any other packages apart from them.

3 Outcome and Deviations

3.1 Overall Result

The result is in Figure 3.

Figure 3: Result of Hough Transform

From Figure 3, we can see that our implementation can find all the circles with a accuracy of 100%. So the performance enhancement section is abbreviated.

3.2 Discussion of the outcome

A. Influence of Overlapping

In order to evaluate the influence of overlapping, we pick a new coin image for circle detection.From this image, we can see that the overlapping of circles can significantly influence the final result. The detection accuracy becomes poor (Figure 4).

This is because the overlapping and coin texture largely mislead the Hough Transform algorithm and therefore leads to a poor result.

B. Influence of Missing Part In order to test the robustness of our implementation, we test the algorithm using Missing Part. The result is shown in Figure 5.

C. Efficiency

The searching space for Hough Transform is very large. This is one of the major disadvantage of Hough Transform.

In order to get a better efficiency, we make several optimizations. I will list them in the following:

- Python language optimization: ($73s \rightarrow 63s$)
    1. Use 'xrange' instead for 'range' for the loops.
    2. Use 'zip' function to efficient make an iterator that aggregates elements from each of the iterables.

Figure 4: Experiment on Overlapping circles.

Figure 5: Experiment on Image with missing part.

- Algorithm optimization: ($63s \rightarrow 57s$)
  1. Add advanced justify expressions to avoid useless searching. (line 68 and 99).

4 Explanation of Software and Program Development

4.1 Implementation Analysis

```
# Author: Kun Yang and Hengtong Zhang

import numpy as np
import cv2
import time
import math
from scipy import signal
from scipy.spatial import distance

img = cv2.imread('HoughCircles.jpg', 0)
img_color = cv2.imread('HoughCircles.jpg')
```

```
start = time.time()

# (a)Denoise the image using a Gaussian blur filter
#   By: Kun Yang
Gaussian = np.array([[1, 2, 1],
[2, 4, 2],
[1, 2, 1]])

Denoised_img = signal.convolve2d(img, Gaussian, boundary='symm', mode='same')
Denoised_img /= 16
# cv2.imwrite('Denoised.jpg', Denoised_img)
print('(a) Denoise the image using a Gaussian blur filter......Done')

# (b & c)Apply an edge detector. Threshold the image from (b) into a binary image
#   By: Kun Yang
Denoised_img = np.uint8(Denoised_img)
Edge_img = cv2.Canny(Denoised_img, 30, 100)
# cv2.imwrite('Edge.jpg', Edge_img)
# cv2.imshow('image', Edge_img)
# cv2.waitKey(0)
print('(b & c) Apply an edge detector......Done')

# (d)Pick a suitable range of radii values of circles
#   We choose 10~60 for raddi values, the smallest
#   circle in the image has radius of 19, the largest
#   circle in the image has radius of 51. So 10~60
#   can cover all possible circles, and the amount
#   of computation can be reduced if we have smaller range.
#   By: Kun Yang
LOW = 10
HIGH = 60

X_list = []
Y_list = []
for a in xrange(LOW, HIGH + 1):
        xl = []
        yl = []
for x in xrange(-a, a + 1):
        for y in xrange(-a, a + 1):
                if int(round(math.sqrt(x ** 2 + y ** 2))) == a:
                        xl.append(x)
                        yl.append(y)
                        X_list.append(xl)
                        Y_list.append(yl)

# (e)Apply Hough transform to detect circle
#   Step One: construct the 3D accumulator
#   Step Two: For every edge pixel, modify the corresponding accumulator position
#   Step Three: Scan the accumlator, find the highest values
#   By: Hengtong Zhang and Kun Yang

accumulator = [[[0 for x in xrange(img.shape[0])] for y in xrange(img.shape[1])] for r in xrange(HIGH - LOW + 1)]

for x in xrange(img.shape[0]):
        for y in xrange(img.shape[1]):
                if Edge_img[x][y] == 255:
                        for r in xrange(HIGH - LOW + 1):
                                for (a, b) in zip(X_list[r], Y_list[r]):
                                        if 0 < (x + a) < img.shape[0]:
# Optimize by: Hengtong
                                                if img.shape[1] > (y + b) > 0:
                                                        accumulator[r][y + b][x + a] += 1

X = []
Y = []
R = []
S = []
for r in xrange(HIGH - LOW + 1):
        for y in xrange(img.shape[1]):
                for x in xrange(img.shape[0]):
                        if accumulator[r][y][x] > 90:
                                X.append(x)
                                Y.append(y)
                                R.append(r)
                                S.append(accumulator[r][y][x])

for i in xrange(len(X)):
        for j in xrange(i + 1, len(X)):
                if distance.euclidean((X[i], Y[i]), (X[j], Y[j])) < 20:
                        if S[i] < S[j]:
```

```
                                R[i] = 0
                            else:
                                R[j] = 0
print('(d & e) Apply Hough transform to detect circle......Done')

# (f)Display the circles detected over original image
#   By: Kun Yang
for i in xrange(len(X)):
if R[i] != 0:
for (a, b) in zip(X_list[R[i]], Y_list[R[i]]):
        if 0 < (X[i] + a) < img.shape[0]:
                if img.shape[1] > (Y[i] + b) > 0:   # Optimize by: Hengtong
                        img_color[X[i] + a][Y[i] + b] = [0, 0, 255]  # blue circles

cv2.imwrite('Result.jpg', img_color)
cv2.imshow('image', img_color)
cv2.waitKey(0)
print('(f) Display the circles detected over original image...... time:' + str(time.time() - start))
```

4.2 Lesson learned

In this project, we learned the importance of algorithm optimization. Hough transform is very powerful technique for shape detection, but exponential growth of the accumulator data and the increase of the number of parameters restrict its usability. When we implement the algorithm, we optimize the runtime by construct efficient data structure and use more efficient syntax. In image processing, run time is critical. In many case, if the program takes too long, the program is useless.

5 Bonus

5.1 Implementation and Analysis

We perform circle detection without using accumulator array, randomly select four edge pixels in the image and determine whether there is a possible circle in the image[3]. In this section, we will show the overall algorithms for the bonus part, and analyze the performance.

The overall algorithm is as following (Algorithm 1):

Data: The binary image with edge detected.
Result: Circles in the image.
Store all edge pixels to the set V and initialize the failure counter f to be 0.Let T_f,T_{min}, T_a, T_d, and T_r. be the five given thresholds. retained in V .;
while *not Finished* **do**
  if $f = T_f$ *or* $|V| < T_{min}$ **then**
    break;
  else
    We randomly pick four pixels v_i, i =1,2,3,4, out of V.When v_i has been chosen, set$V = V - \{v_i\}$.;
    assert1 = There exist a circle determined by 3 points and distance between any two of the three agent pixels is larger than Ta ;
    assert2 = the distance between the fourth pixel and the boundary of the possible circle is larger than T_d. ;
    if *assert1 and assert2* **then**
      Assume C_{ijk} is the possible circle.;
      Set the counter C to be 0. ;
      for *each* v_l *in* V **do**
        if $d_{l \to ijk} \leq T_d$ **then**
          $C+ = 1$;
          take v_k out of V.;
        end
      end
      for *each circle candidate* **do**
        if *If* $C \geq 2 * \pi * r_{ijk} * T_r$ **then**
          The circle should be the circle deserve detection. $f = 0$;
        end
      end
    else
      Put v_i, i =1,2,3,4, back to V;
      $f+ = 1$;
    end
  end
end

Algorithm 1: Bonus Algorithm

Note: In our implementation, for the query If $C \geq 2 * \pi * r_{ijk} * T_r$, we do not adopt the one in the algorithm. Actually we change it into If $C \geq 3 * r_{ijk}$. And we filter out the circles that are too large or too small.

The overall performance is as Figure 6:

Figure 6: Bonus result.

5.2 Lesson learned

The intuition behind the algorithm is that:

- We can determine a circle from 3 points.
- It is difficult to randomly select 4 points that all lie on a single circle.
- It is even more difficult for the pixels lies on a circle, which is determined by 4 random points, to achieve the fact that they are mostly on an detected edge If $C \geq 2 * \pi * r_{ijk} * T_r$.

However, the algorithm just do not work as well as it seems. Sometimes, the algorithm may misfinding some circles.

6 Task Division

Tasks for Kun Yang

(1) Denoise the image. (2) Edge detection and thresholding. (3) Apply Hough transform.

Tasks for Hengtong Zhang

(1) Code optimization, debug. (2) Visualization and Evaluation. (3) Bonus section.

References

[1] Milan Sonka, Vaclav Hlavac, and Roger Boyle, *Image processing, analysis, and machine vision*, 2014.

[2] HK Yuen, John Princen, John Illingworth, and Josef Kittler, "Comparative study of hough transform methods for circle finding," *Image and vision computing*, 1990.

[3] Teh-Chuan Chen and Kuo-Liang Chung, "An efficient randomized algorithm for detecting circles," *Computer Vision and Image Understanding*, 2001.