Practical Assignment № 2

Hough Transform.

Instructing Professor :Sergei Shavetov & Andrei Zhdanov
Author :Xu Miao

May 7, 2022

Contents

1 Introduction

§1.1 Student information

- ★ Name: Xu Miao
- ★ ITMO Number: 293687
- ★ HDU Number: 19322103

§1.2 Purpose.

Study of the transformation to find of geometric primitives.

§1.3 Introduction

In this report, I will complete the content of the following sections:

1) **Search for lines.**
   a) Choose Select three arbitrary images containing lines. Perform to search for straight lines using the Hough transform both for the original image and for the image obtained using any differential operator.
   b) Plot the found lines on the original image.
   c) Mark the start and end points of the lines.
   d) Determine the lengths of the shortest and longest lines, calculate the number of lines found.
2) **Search for circles.**
   a) Select three arbitrary images containing circles. Search for circles of both a certain radius and from a given range using the Hough transform, both for the original image and for the image obtained using any differential operator.
   b) Plot the found circles on the original image.
3) **Optional.**
   a) Implement the classic Hough transform algorithms for lines and circles.
   b) Compare implementation results with ones obtained in the first two points of the assignment.
   c) **Highlight the selected points** in the Hough parameter space.

2 Notations

Table 2.1: Notation used in this report

Symbol	Definition
I	image brightness
R	The luminance component of the R channel of the image
G	The luminance component of the G channel of the image
B	The luminance component of the B channel of the image
m	mean value
s	Standard deviation
R	Relative smoothness
E	entropy

⋆ Other notations instructions will be given in the text.

3 Solution of problem

§3.1 Hough Transform : Search for lines.

§3.1.i Theorem.

find lines: Hough space

Hough transform uses the parameter space to search for geometric primitives. The most common parametric equation of lines is:

$$y = kx + b \tag{3.1}$$

$$x\cos\Theta + y\sin\Theta = \rho \tag{3.2}$$

where ρ - radius vector drawn from the origin to the line; $\Theta-$ inclination angle of the radius vector.

This experiment is based on the sinusoidal parameter space of (3.2),From the following figure Fig.3.1,3.2 , we can intuitively see the mutual conversion relationship between Hough space and Cartesian coordinate system:

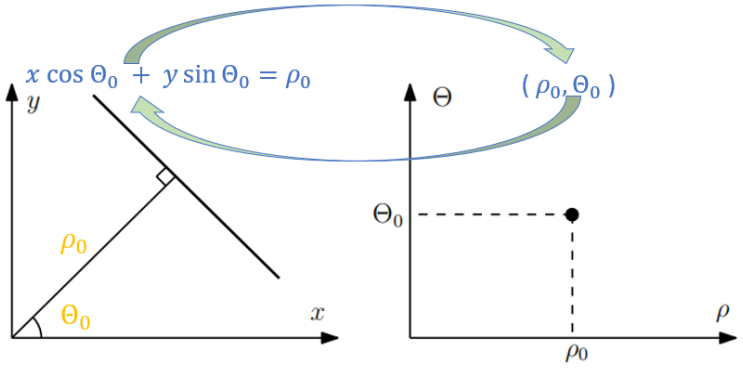

Fig. 3.1. Mapping of straight lines in Hough space

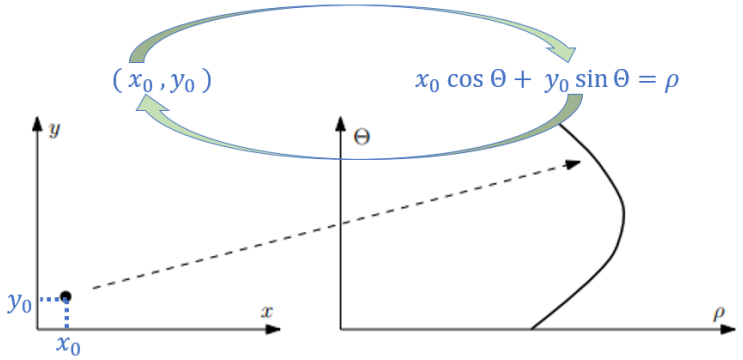

Fig. 3.2. Mapping of points in Hough space

find lines: voting

The classical Hough transform, based on the considered point voting idea, was originally designed to select lines on binary images,The voting process of Hough transform is shown in the following figure:

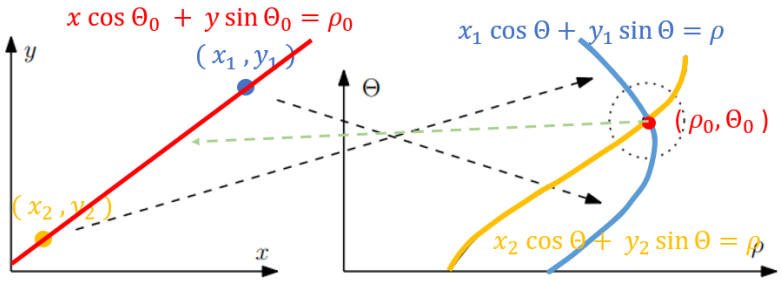

Fig. 3.3. Voting procedure.

After voting, we find the straight line corresponding to all local maxima in the Hough accumulator by setting the threshold, which is the straight line found by the Hough transform.

Note: The Hough transform is invariant to shift, scaling, and rotation.

§3.1.ii Solution

1.Hough Transform : Search for lines.

1.a Selection of the original image

Select the original image as shown in Fig.3.1:

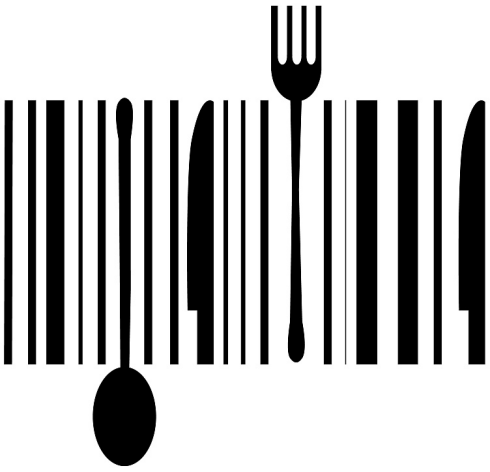

Fig. 3.4. original image1

Fig. 3.5. original image2

Fig. 3.6. original image3

1.b Image processing result

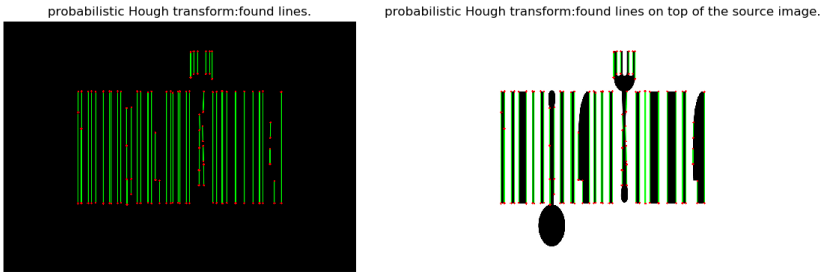

Fig. 3.7. Result image1

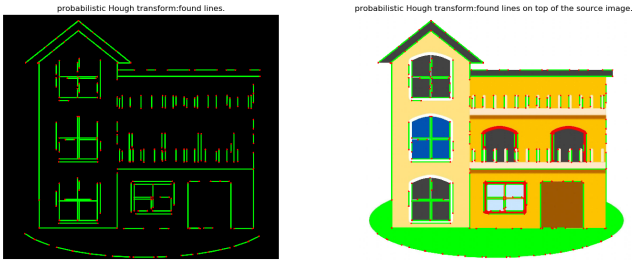

Fig. 3.8. Result image2

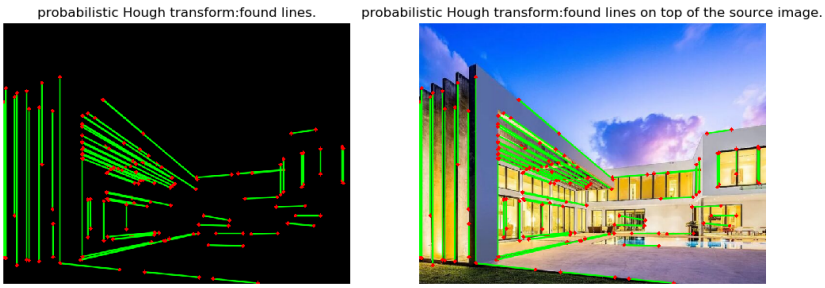

Fig. 3.9. Result image3

1.c full source code

You can click ⋆here⋆ to see the full code for this part.

1.d Comments

The data of the straight lines found in the three pictures are shown in Table 3.1 below:

Table 3.1: Line data found by probabilistic Hough transform

Image	Number	lengths of the shortest lines	lengths of the longest lines
Image1	58	54	406
Image2	161	50	869
Image3	79	50.99	385

- ⋆ Line segments (start and end points) in an image can be found using the Probabilistic Hough Transform.(Fig3.7-3.9),So we calculate the length and number of line segments(Table3.1)
- ⋆ From the image processing results, the disadvantages of the probabilistic Hough transform are: 1. Some line segments may not be found completely.(Fig3.9) 2. Some line segments are incomplete and divided into multiple segments (Fig3.7,3.9)3. Arcs with smaller radians may be misjudged as line segments(Fig3.8)

§3.2 Hough Transform : Search for circles.

§3.2.i Theorem.

find circles: Hough space Using the Hough transform to find a circle with a known radius is similar to finding a straight line
Given the coordinates of four points $P1, P2, P3, P4$, find the coordinates (a, b) of the center of the circle (assuming r is known here). For each point Pi, the parameters (a, b) under the constraints of Pi and r are distributed on a circle, and all Pi are combined, and the intersection formed by the constraints is the desired parameter (a, b),Find the Hough space transformation of a circle with a given radius as shown below:

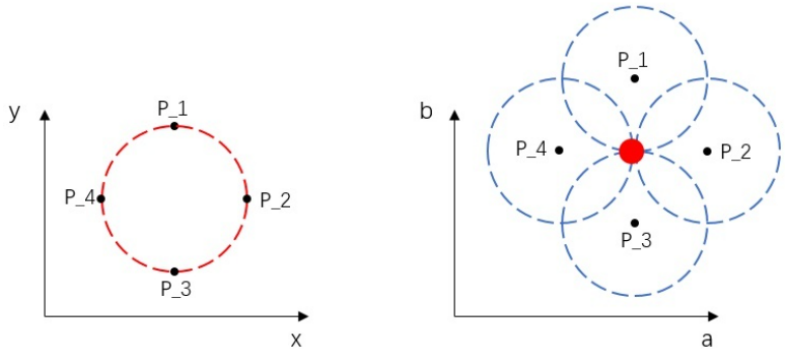

Fig. 3.10. Finding a Circle of Radius R: Mapping of Points in Hough Space

find circles: voting
The classical Hough transform, based on the considered point voting idea, was originally designed to select circles on binary images,The voting process of Hough transform is shown in the following figure:

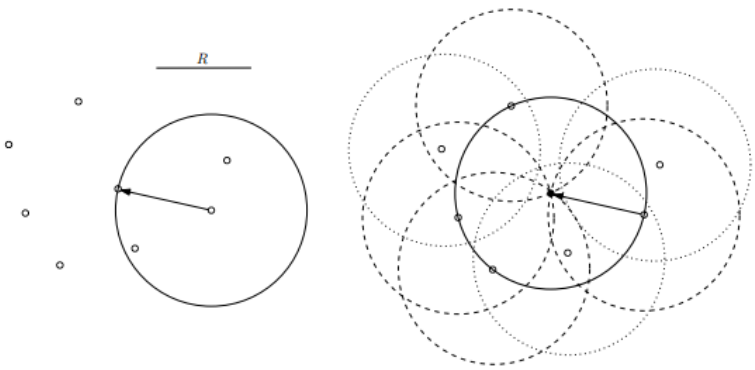

Fig. 3.11. Finding a Circle of Radius R:Voting procedure.

After voting, we find the straight line corresponding to all local maxima in the Hough accumulator by setting the threshold, which is the Circle of Radius R found by the Hough transform.
Note: The Hough transform is invariant to shift, scaling, and rotation.

§3.2.ii Solution

1.Hough Transform : Search for lines.

1.a Selection of the original image

Select the original image as shown in Fig.3.1:

Fig. 3.12. original image1

Fig. 3.13. original image2

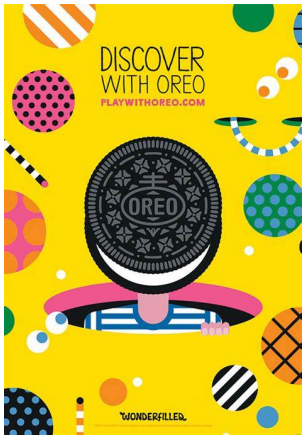

Fig. 3.14. original image3

1.b Image processing result

Fig. 3.15. Result image1

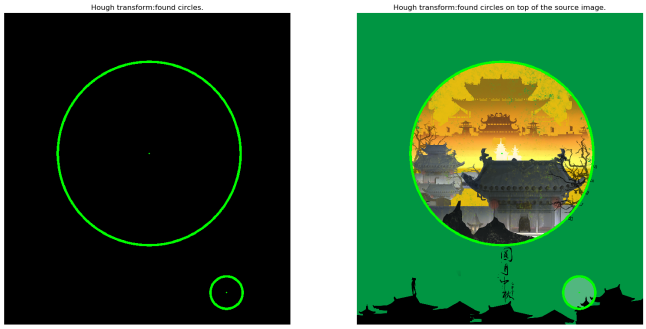

Fig. 3.16. Result image2

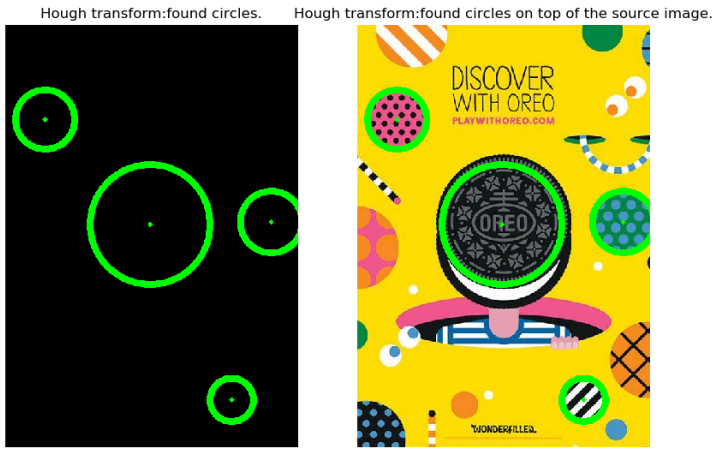

Fig. 3.17. Result image3

1.c full source code

You can click ⋆here⋆ to see the full code for this part.

1.d Comments

- ⋆ It can be seen from Fig3.15-3.17 that the result of finding a circle is better, but in the code, it is necessary to continuously debug and modify the parameters for each picture to achieve better results
- ⋆ The syntax for finding circles using Hough transform in python opencv is:

  circles = cv2.HoughCircles(image, method, dp, minDist[, param1[, param2[, minRadius[, maxRadius]]]]])

  Below are the parameters explained in detail

  1) image: 8-bit, single-channel, grayscale input image
  2) method: HOUGH_GRADIENT and HOUGH_GRADIENT_ALT
  3) dp: The inverse ratio of accumulator resolution and image resolution
  4) minDist: Minimum distance between the centers of the detected circles. All the candidates below this distance are neglected as explained above
  5) param1: it is the higher threshold of the two passed to the Canny edge detector (the lower canny threshold is twice smaller)
  6) param2: it is the accumulator threshold for the circle centers at the detection stage as discussed above.
  7) minRadius: minimum radius that you expect. If unknown, put zero as default.
  8) maxRadius: if -ve, only circle centers are returned without radius search. If unknown, put zero as default.

§3.3 Optional.classcial Hough transform

§3.3.i Solution

1.Classcial Hough transform:find lines

1.a Selection of the original image

Use the same original picture as in 3.1 Probabilistic Hough Transform

1.b Image processing result

To make the parameter space look more obvious I did a histogram equalization of the results in the parameter space.

1) image 1

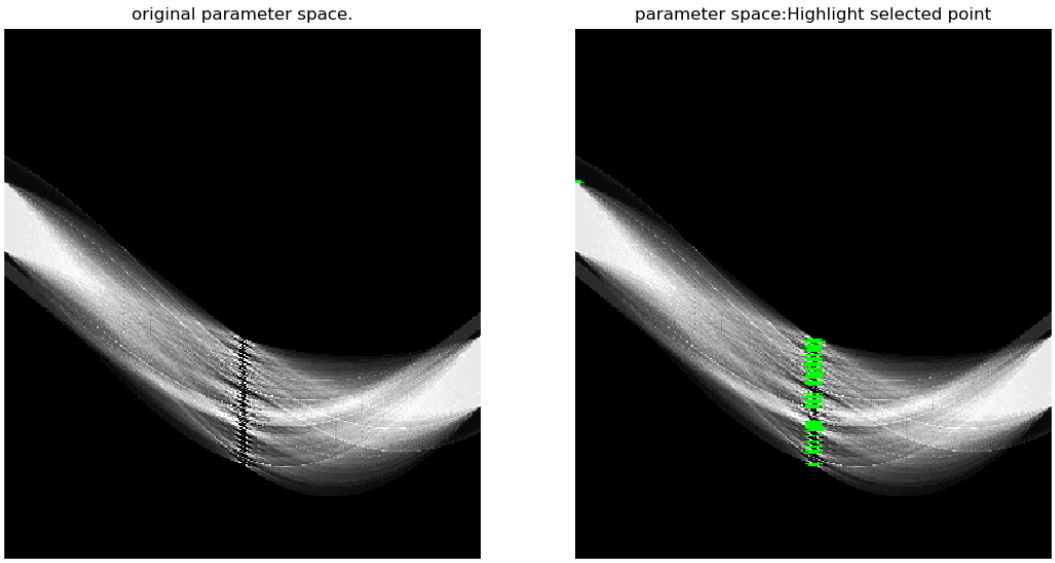

Fig. 3.18. Hough parameter space of original image 1 (after histogram equalization)

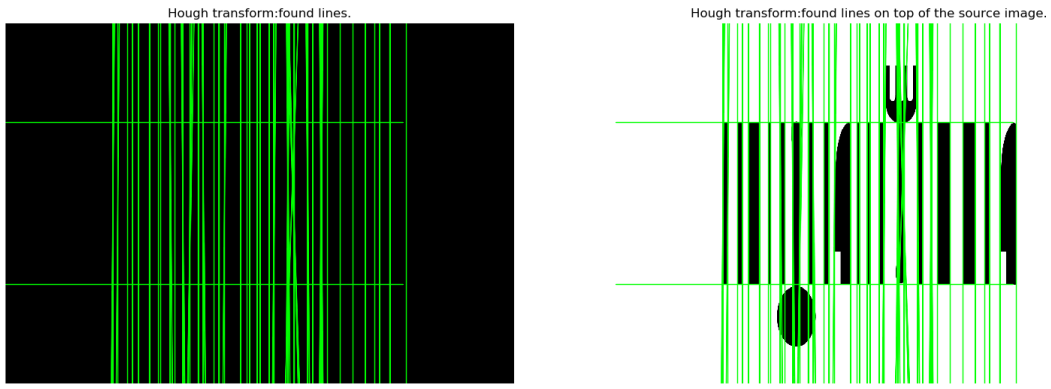

Fig. 3.19. Result image1

2) image 2

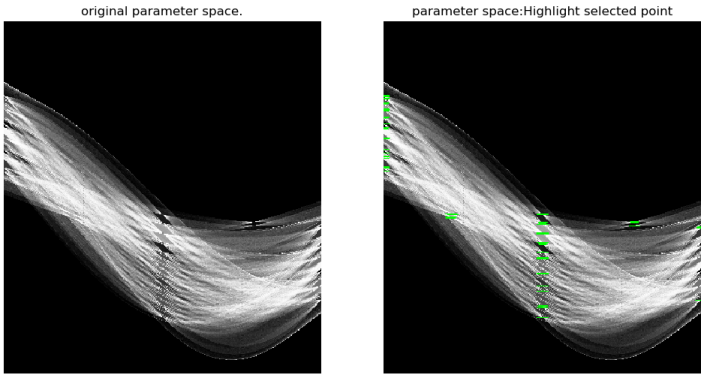

Fig. 3.20. Hough parameter space of original image 2 (after histogram equalization)

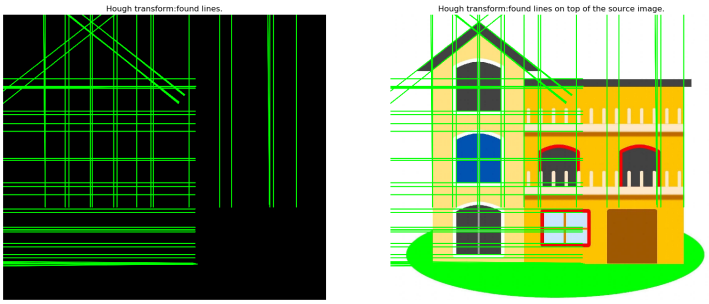

Fig. 3.21. Result image2

3) image 3

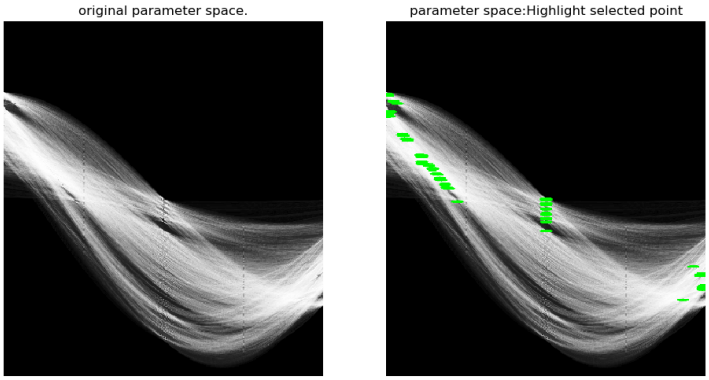

Fig. 3.22. Hough parameter space of original image 3 (after histogram equalization)

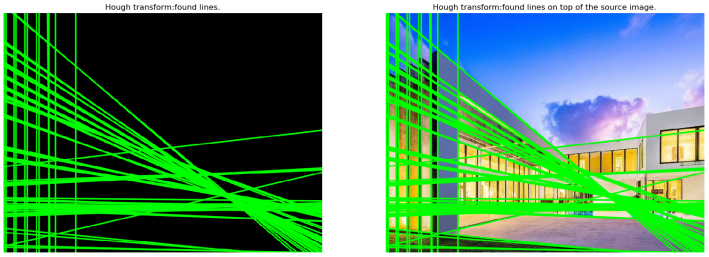

Fig. 3.23. Result image3

1.c full source code

You can click ⋆here⋆ to see the full code for this part.

1.d Comments

- ⋆ The Hough parameter space can be visualized using the classical Hough transform.
- ⋆ When using the classic Hough transform to find a straight line, the endpoint of the line segment cannot be found to calculate the length of the line segment, only the straight line can be found.

2.Classcial Hough transform:find circles

1.a Selection of the original image

chose the original image 1 in 3.2 to do the classic Hough transform.

1.b Image processing result

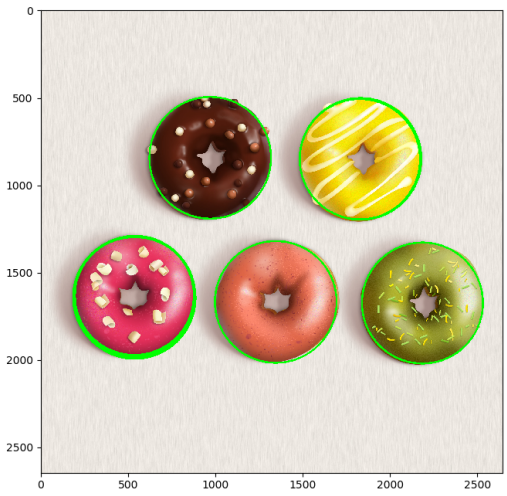

Fig. 3.24. Result image1

A total of 13 circles were found in the figure (including overlapping circles),The parameters for finding the circle are shown in the following table:

Table 3.2: Parameters for finding circles in original image 1

Number	Center of circle(x_0, y_0)	Radius$R$
1	(1828, 849)	348
2	(537, 1636)	346
3	(532, 1640)	348
4	(968, 843)	348
5	(534, 1640)	346
6	(536, 1644)	348
7	(536, 1639)	344
8	(537, 1633)	344
9	(2182, 1673)	346
10	(1824, 851)	344
11	(535, 1637)	344
12	(534, 1630)	340
13	(1343, 1667)	348

It can be seen from Table 3.2 that the radius of the circle found is: 340, 344, 346, 348:

Plot the Hough stacker parameter space under these radii:

1. $R = 340$

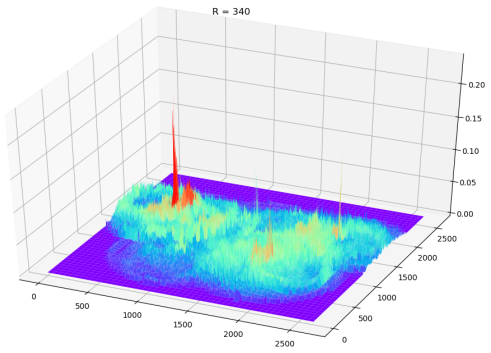

Fig. 3.25. $R = 340$ Parameter Space

2. $R = 344$

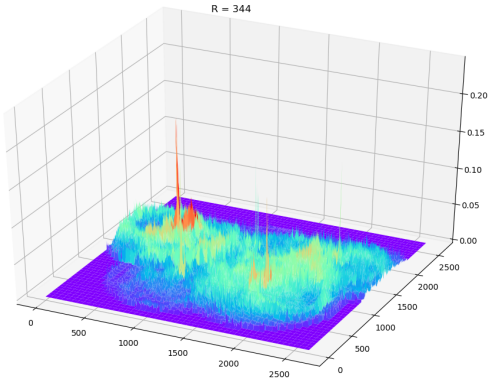

Fig. 3.26. $R = 344$ Parameter Space

3. $R = 346$

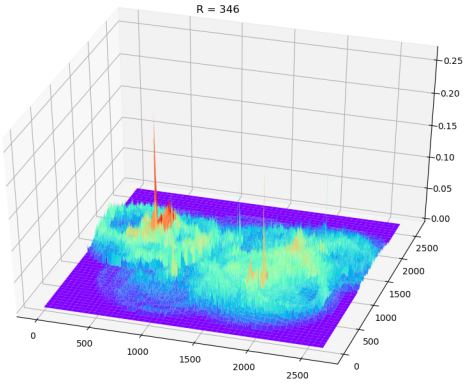

Fig. 3.27. $R = 346$ Parameter Space

4. $R = 348$

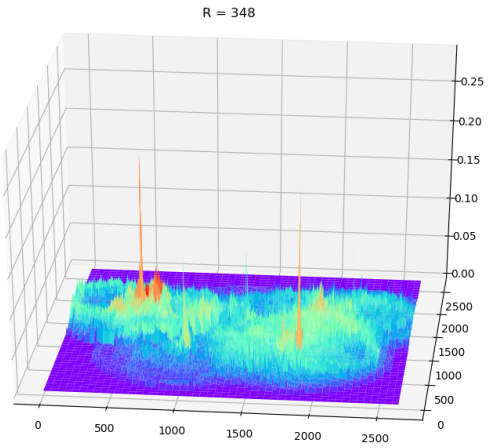

Fig. 3.28. $R = 348$ Parameter Space

1.c full source code

You can click ⋆here⋆ to see the full code for this part.

1.d Comments

- ⋆ The Hough parameter space can be visualized using the classical Hough transform.
- ⋆ Using skimage's Hough transform to find the parameters of the circle is more difficult to adjust, but it is more flexible because the voting parameter space can be directly seen. It is recommended to use Hough transform in opencv to find circles when there are no special requirements

4 Conclusion

In this exercise assignment, I learned the following application methods:

- ⋆ If we want to operate on the Hough parameter space and have special image processing requirements (such as adjusting the threshold size of voting), it is recommended to use the transform of skimage to process the image.
- ⋆ The main principle of the Hough transform is to find common locus of points.
- ⋆ The classical Hough transform, based on the considered point voting idea.
- ⋆ The Hough transform is invariant to shift, scaling, and rotation.

Note: The characteristics of each method are described in the previous comments and will not be repeated here

5 The answer to the questions

1) What is the main principle of the Hough transform?

The main principle of the Hough transform is to find common locus of points.

2) May the Hough transform be used to find arbitrary contours that cannot be described analytically?

No, because Hough transform uses the parameter space to search for geometric primitives.

3) What are the **recurrent** and **generalized** Hough transforms?

recurrent Hough transforms :

a) Is used to obtain information about the straight lines passing through each point ano their parameters

b) Sliding Hough transform parameter stacking on the image using a window of $W \times W$

generalized Hough transforms :

a) The Generalized Hough Transform (GHT) was proposed by DanaH.Ballard in 1981, and it was adjusted based on the principle of template matching on the basis of the Hough transform. The generalized Hough transform does not require an analytical formula that can give the shape to be detected, it can detect any given shape.

b) The considered approach can be generalized to the case when the object is allowed to rotate, in addition to shifting (invariance to rotation)

c) Hough transform on the global image

4) What are the ways of parametrization in the Hough transform?

1. Perimeter points (m, n):

a) Lines are described by a pair of end points lying on the perimeter of the $N \times N$ grid.

b) Obviously, the number of points will be equal to $4N$ or (taking into account symmetry) $1/2 \cdot (4N \times 4N) = 8N^2$ lines

c) Since a quarter of these points lie on the same straight line (side of the square), the final size of the parameter array will be $6N^2$

3. Tilt and offset (a, d):

a) One point of intersection of the straight line with the grid perimeter $n(0 \leq n < N)$ is used

b) and an angle defined by the perimeter point $a(-N + 1 \leq a \leq N - 1)$ so that straight line passing through the grid center and point a is parallel to a given one

c) The accumulator array contains $4N^2$ elements

recurrent Hough transforms :

a) The angle a is defined by the direction of a line from the grid center to a perimeter point

b) The displacement of the line vertically or horizontally from the center is fixed using the distance from the center to the intersection of the straight line with the Oy or Ox axes

c) Generates $3N^2$ or $4N^2$ accumulator cells

d) The (a, d)-parametrization is closely related to the (ρ, θ) - parametrization and with the parameters of the equation $y = ax + b$, where a is interpreted as the slope of a straight line

I

Appendix

A Complete source code

§A.1 calculation

§A.2 1.Hough transform : Search for lines

and you can click ⋆here⋆ to return to reading the report

Python

```
import math
import cv2
import matplotlib.pyplot as plt
import numpy as np
from skimage import transform
I = cv2.imread('3.jpg',cv2.IMREAD_COLOR)
#preprocess an image with Canny algorithm:
Iedge= cv2.Canny(I,200,250,None,3)
#Hough parameter
angles = np.linspace(-np.pi/2,np.pi/2,360,endpoint=False)
Ih,theta,rho = transform.hough_line(Iedge,theta = angles)
Ih = cv2.resize(Ih.astype(np.float32)/np.max(Ih),(Ih.shape[1],400))
#Hough transform
lines = cv2.HoughLines(Iedge,1,np.pi/180,180)
'''
returned value: all (rho,theta)
x*cos(theta) + y*sin(theta) = rho
'''
# probabilistic Hough transform
linesP = cv2.HoughLinesP(Iedge,1,np.pi/180,50,None,50,4)
'''
returned value: the endpoint of the line found
x*cos(theta) + y*sin(theta) = rho
'''
Iout = I.copy()
Ioutp = I.copy()
Ib = np.zeros_like(I)
Ipb = np.zeros_like(I)
#Hough transform(draw lines)
if lines is not None:
    for i in range(0,len(lines)):
        rho = lines[i][0][0]
        theta = lines[i][0][1]
        a,b = math.cos(theta) ,math.sin(theta)
        x0 ,y0 = a*rho ,b*rho
        pt1 = np.int32((x0 - 1000*b, y0 + 1000*a))
```

```
        pt2 = np.int32((x0 + 1000*b, y0 - 1000*a))
        cv2.line(Iout,pt1,pt2,(0,255,0),2,cv2.LINE_AA)
        cv2.line(Ib,pt1,pt2,(0,255,0),2,cv2.LINE_AA)
#probabilistic Hough transform(draw lines)
min = 10000
max = 0
minn = 0
maxn = 0
number = linesP.shape[0]

if linesP is not None:
    for i in range(0,len(linesP)):
        l = linesP[i][0]
        distance = math.hypot(l[0]-l[2],l[1]-l[3])
        if distance >max:
            max = distance
        if distance < min:
            min = distance
        print((l[0],l[1]),(l[2],l[3]),distance)
        cv2.line(Ioutp,(l[0],l[1]),(l[2],l[3]),(0,255,0),5,cv2.LINE_AA)
        cv2.circle(Ioutp,(l[0],l[1]),2,(0,0,255),5)
        cv2.circle(Ioutp,(l[2],l[3]),2,(0,0,255),5)
        cv2.line(Ipb,(l[0],l[1]),(l[2],l[3]),(0,255,0),5,cv2.LINE_AA)
        cv2.circle(Ipb,(l[0],l[1]),2,(0,0,255),5)
        cv2.circle(Ipb,(l[2],l[3]),2,(0,0,255),5)
I = cv2.cvtColor(I,cv2.COLOR_BGR2RGB)
Iedge = cv2.cvtColor(Iedge,cv2.COLOR_BGR2RGB)
Ib = cv2.cvtColor(Ib,cv2.COLOR_BGR2RGB)
Iout = cv2.cvtColor(Iout,cv2.COLOR_BGR2RGB)
Ioutp = cv2.cvtColor(Ioutp,cv2.COLOR_BGR2RGB)
Ipb = cv2.cvtColor(Ipb,cv2.COLOR_BGR2RGB)
print('number of lines:',number,'\n')
print('lengths of the longest lines:',min,'\n')
print('lengths of the shortest lines:',max,'\n')

plt.subplot(121),plt.title('The original image.'),plt.axis ('off')
plt.imshow(I)
plt.subplot(122),plt.title('Source image processed by the Canny
    algorithm.'),plt.axis ('off')
plt.imshow(Iedge)
plt.show()
plt.waitforbuttonpress
plt.imshow(Ih),plt.title(' parameter space.'),plt.axis ('off')
plt.show()
plt.waitforbuttonpress
plt.subplot(121),plt.title('Hough transform:found lines.'),plt.axis ('
    off')
plt.imshow(Ib)
plt.subplot(122),plt.title('Hough transform:found lines on top of the
    source image.'),plt.axis ('off')
```

```
plt.imshow(Iout)
plt.show()
plt.waitforbuttonpress
plt.subplot(121),plt.title('probabilistic Hough transform:found lines.'),plt.axis ('off')
plt.imshow(Ipb)
plt.subplot(122),plt.title('probabilistic Hough transform:found lines on top of the source image.'),plt.axis ('off')
plt.imshow(Ioutp)
plt.show()
plt.waitforbuttonpress
```

§A.3 2.Hough transform : Search for circles

and you can click ⋆here⋆ to return to reading the report

Python

```
import cv2
import matplotlib.pyplot as plt
from numpy import zeros_like
import numpy as np
I = cv2.imread('10.jfif',cv2.IMREAD_COLOR)
#preprocess an image with Canny algorithm:
Iedge= cv2.cvtColor(I,cv2.COLOR_BGR2GRAY)
Ib = zeros_like(I)
Iout = I.copy()

circles = cv2.HoughCircles(Iedge,cv2.HOUGH_GRADIENT,1,160,param1=100,param2=79,minRadius=0,maxRadius=0)
print('circles.shape:',circles.shape)
print('circles:\n',circles)
circles = np.uint16(np.around(circles))
for cir in circles[0]:
    print('circle.shape:',cir.shape,'circle:',cir)
    cv2.circle(Ib,(cir[0],cir[1]),cir[2],(0,255,0),10)
    cv2.circle(Ib,(cir[0],cir[1]),2,(0,255,0),3)
    cv2.circle(Iout,(cir[0],cir[1]),cir[2],(0,255,0),10)
    cv2.circle(Iout,(cir[0],cir[1]),2,(0,255,0),3)

I = cv2.cvtColor(I,cv2.COLOR_BGR2RGB)
Iedge = cv2.cvtColor(Iedge,cv2.COLOR_BGR2RGB)
Ib = cv2.cvtColor(Ib,cv2.COLOR_BGR2RGB)
Iout = cv2.cvtColor(Iout,cv2.COLOR_BGR2RGB)

plt.subplot(121),plt.title('The original image.'),plt.axis ('off')
plt.imshow(I)
plt.subplot(122),plt.title('Source image processed by the Canny algorithm.'),plt.axis ('off')
```

```
plt.imshow(Iedge)
plt.show()
plt.waitforbuttonpress
plt.subplot(121),plt.title('Hough transform:found circles.'),plt.axis ('off')
plt.imshow(Ib)
plt.subplot(122),plt.title('Hough transform:found circles on top of the source image.'),plt.axis ('off')
plt.imshow(Iout)
plt.show()
plt.waitforbuttonpress
```

§A.4 3. Classcial Hough transform:find lines

and you can click ⋆here⋆ to return to reading the report

Problem 1

```
import math
import cv2
import matplotlib.pyplot as plt
import numpy as np
from skimage import transform

# Read an image from file and preprocess
I = cv2.imread('3.jpg',cv2.IMREAD_COLOR)
Iedge = cv2.Canny(I,50,200,None,3)
#Do Hough transform
angles = np.linspace(-np.pi/2,np.pi/2,360,endpoint=False)# -90~90
ps,theta,rho = transform.hough_line(Iedge,theta=angles)
mmin = np.min(rho)
#Find lines with Hough transform
Ih,theta,rho = transform.hough_line_peaks(ps,theta,rho,min_distance=0,
    min_angle=0, threshold=0.3*np.max(ps))
#Histogram equalization of parameter space
ps = np.array(ps,dtype='uint8')
ps = cv2.equalizeHist(ps)
#Conversion of parameter space to three-channel display
# (easy to highlight selected points)
img2 = np.zeros((ps.shape[0],360,3),int)
img2[:,:,0] = ps
img2[:,:,1] = ps
img2[:,:,2] = ps
#Create and output image
Iout = I.copy()
Ib = np.zeros_like(I)
if theta is not None:
    for i in range(0,len(theta)):
        a = math.cos(theta[i])
```

```
        b = math.sin(theta[i])
        #Highlight selected points in the parameter space
        cv2.circle(img2,((int(round(theta[i]*180/math.pi)+90))*2,int(
            round(rho[i]-mmin))),2,(0,255,0),10)
        x0,y0 = a*rho[i],b*rho[i]
        pt1 = np.int32((x0-1000*b,y0+1000*a))
        pt2 = np.int32((x0+1000*b,y0-1000*a))
        cv2.line(Iout,pt1,pt2,(0,255,0),2,cv2.LINE_AA)
        cv2.line(Ib,pt1,pt2,(0,255,0),2,cv2.LINE_AA)
#Zoom parameter space image
img3 = np.zeros((400,360,3))
img3[:,:,0] = cv2.resize(img2[:,:,0].astype(np.float32)/np.max(img2
    [:,:,0]),(img2[:,:,0].shape[1],400))
img3[:,:,1] = cv2.resize(img2[:,:,1].astype(np.float32)/np.max(img2
    [:,:,1]),(img2[:,:,1].shape[1],400))
img3[:,:,2] = cv2.resize(img2[:,:,2].astype(np.float32)/np.max(img2
    [:,:,2]),(img2[:,:,2].shape[1],400))
img2 = np.zeros((400,360,3))
ps = cv2.resize(ps.astype(np.float32)/np.max(ps),(ps.shape[1],400))
img2[:,:,0] = ps
img2[:,:,1] = ps
img2[:,:,2] = ps
I = cv2.cvtColor(I,cv2.COLOR_BGR2RGB)
Iedge = cv2.cvtColor(Iedge,cv2.COLOR_BGR2RGB)
Ib = cv2.cvtColor(Ib,cv2.COLOR_BGR2RGB)
Iout = cv2.cvtColor(Iout,cv2.COLOR_BGR2RGB)
#display image
plt.subplot(121),plt.title('The original image.'),plt.axis ('off')
plt.imshow(I)
plt.subplot(122),plt.title('Source image processed by the Canny
    algorithm.'),plt.axis ('off')
plt.imshow(Iedge)
plt.show()
plt.waitforbuttonpress

plt.subplot(121),plt.title('original parameter space.'),plt.axis ('off'
    )
plt.imshow(img2)
plt.subplot(122),plt.title('parameter space:Highlight selected point'),
    plt.axis ('off')
plt.imshow(img3)
plt.show()
plt.waitforbuttonpress

plt.subplot(121),plt.title('Hough transform:found lines.'),plt.axis ('
    off')
plt.imshow(Ib)
plt.subplot(122),plt.title('Hough transform:found lines on top of the
    source image.'),plt.axis ('off')
plt.imshow(Iout)
```

```
plt.show()
plt.waitforbuttonpresss
```

§A.5 4.Classcial Hough transform:find circles

and you can click ⋆here⋆ to return to reading the report

Python

```
import cv2
import numpy as np
from skimage import transform,color
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from skimage.draw import circle_perimeter

I = cv2.imread('6.jpg',cv2.IMREAD_COLOR)
Iedge = cv2.Canny(I,80,160,None,3)
Igray = cv2.cvtColor(I,cv2.IMREAD_COLOR)

hough_radiu = np.arange(340, 350, 2)
hough_res = transform.hough_circle(Iedge,hough_radiu)
r340 = hough_res[0]
r344 = hough_res[2]
r346 = hough_res[3]
r348 = hough_res[4]
accums, cx, cy, radiu = transform.hough_circle_peaks(hough_res,
   hough_radiu,total_num_peaks=13)
# Draw them
fig, ax = plt.subplots(ncols=1, nrows=1, figsize=(20, 10))
Iout = I.copy()
for center_y, center_x, radius in zip(cy, cx, radiu):
    cv2.circle(Iout,(center_x,center_y),radius,(0,255,0),10)
    print((center_x,center_y),radius,'\n')
Iout = cv2.cvtColor(Iout,cv2.COLOR_BGR2RGB)
ax.imshow(Iout, cmap=plt.cm.gray)
plt.show()


I = cv2.cvtColor(I,cv2.COLOR_BGR2RGB)
Iedge = cv2.cvtColor(Iedge,cv2.COLOR_BGR2RGB)

plt.subplot(121),plt.title('The original image.'),plt.axis ('off')
plt.imshow(I)
plt.subplot(122),plt.title('Source image processed by the Canny
   algorithm.'),plt.axis ('off')
plt.imshow(Iedge)
plt.show()
plt.waitforbuttonpress
```

```
39
40 figure = plt.figure()
41 axes = Axes3D(figure)
42 X = np.arange(0, 2645, 1)
43 Y = np.arange(0, 2645, 1)
44
45 X, Y = np.meshgrid(X, Y)
46 Z = r340
47 axes.plot_surface(X, Y, Z,cmap='rainbow')
48 axes.set_title('R = 340')
49 plt.show()
50
51 figure = plt.figure()
52 axes = Axes3D(figure)
53 X = np.arange(0, 2645, 1)
54 Y = np.arange(0, 2645, 1)
55
56 X, Y = np.meshgrid(X, Y)
57 Z = r344
58 axes.plot_surface(X, Y, Z,cmap='rainbow')
59 axes.set_title('R = 344')
60 plt.show()
61
62 figure = plt.figure()
63 axes = Axes3D(figure)
64 X = np.arange(0, 2645, 1)
65 Y = np.arange(0, 2645, 1)
66
67 X, Y = np.meshgrid(X, Y)
68 Z = r346
69 axes.plot_surface(X, Y, Z,cmap='rainbow')
70 axes.set_title('R = 346')
71 plt.show()
72
73 figure = plt.figure()
74 axes = Axes3D(figure)
75 X = np.arange(0, 2645, 1)
76 Y = np.arange(0, 2645, 1)
77
78 X, Y = np.meshgrid(X, Y)
79 Z = r348
80 axes.plot_surface(X, Y, Z,cmap='rainbow')
81 axes.set_title('R = 348')
82 plt.show()
```