Hough Transorm

Hough Transform is a popular technique to detect any shape, if you can represent that shape in mathematical form. It can detect the shape even if it is broken or distorted a little bit.

Enviroment

```
'''
PyCharm 2021.2 (Professional Edition)
Build #PY-212.4746.96, built on July 27, 2021
Licensed to 昶緻 孟
Subscription is active until August 8, 2022.
For educational use only.
Runtime version: 11.0.11+9-b1504.13 x86_64
VM: OpenJDK 64-Bit Server VM by JetBrains s.r.o.
macOS 11.6
GC: G1 Young Generation, G1 Old Generation
Memory: 2048M
Cores: 8
Registry: ide.balloon.shadow.size=0
Non-Bundled Plugins: GrepConsole (12.0.211.6086.4), net.vektah.codeglance (1.5.4), com.mallowigi (50.2.0), mobi.hsz.idea.gitignore (4.3.0), com.chrisrm.idea.MaterialThemeUI (6.9.1), izhangzhihao.rainbow.brackets (6.21)
'''
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
from numba import njit
import cv2
from hw1.main import NeighborhoodOp, PixelOp
from collections import defaultdict
from sklearn.cluster import KMeans, DBSCAN
import time
```

Hough Transform(HT)

Every line on the Cartesian plane can be described as $y = ax + b$, which can be rewritten as $b = y - ax$. The later form means for any given point (x, y), any possible (a, b) solution is a line going through it. As long as we can find out every possible line on the edge image, the most common lines should be the one we want.

Hough Space

In general, the straight line $y = ax + b$ can be represented as a point (a, b) in the parameter space. However, vertical lines pose a problem. They would give rise to unbounded values of the slope parameter a. Thus, for computational reasons, Duda and Hart [5] proposed the use of the **Hesse normal form**:

$\rho = x\cos(\theta) + y\sin\theta,$

where r$\boldsymbol{r}$ is the distance from the origin to the closest point on the straight line, and θ is the angle between the x axis and the line connecting the origin with that closest point.

```
@njit()
def get_hough(img, threshold, t_step):
  unlabel_points = (img >= threshold)
  hough = []
  for x, y in zip(*unlabel_points.nonzero()):
    for theta in np.arange(0, 180, t_step):
      rho = x * np.cos(theta / 180 * np.pi) + y * np.sin(theta / 180 * np.pi)
      hough.append((theta, rho))
      return hough
```

Here, we use *get_hough* to calculate discrete finite lines (θ, ρ) for all the significant points(brighter than *threshold*) with constraint *t_step* .

Accumulator

```
accumulator = defaultdict(int)
for theta, rho in hough_space:
    accumulator[(int(theta), int(rho))] += 1
filtered_points = {k: v for k, v in sorted(accumulator.items(),
                                    key=lambda item: item[1], reverse=True)
                   if v >= sample_threshold}
```

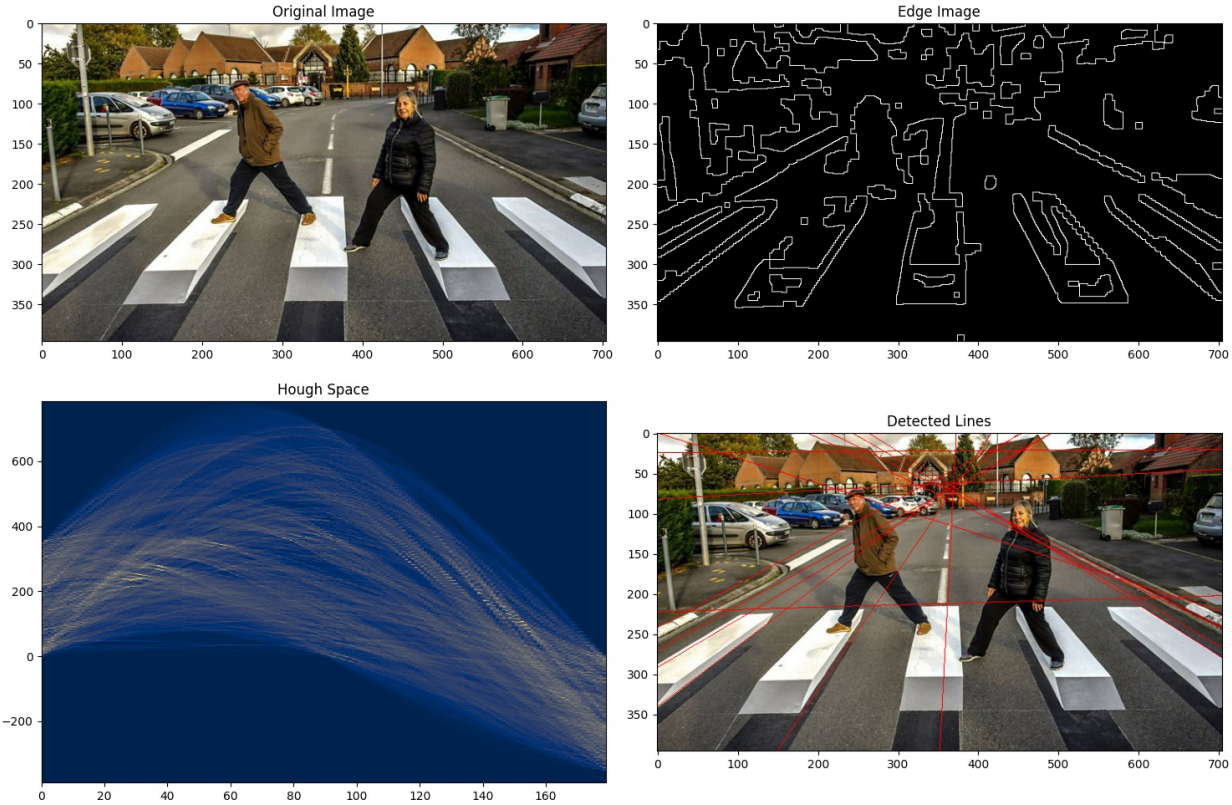
Now, we count lines with *accumulator* (it's the slowest part and I tried a lot, but time cost is still there😢.), sort them and filter them with *sample_threshold* .

Clustering

```
candidate_lines = np.array(list(filtered_points.keys())).reshape(-1, 2)
model = DBSCAN(eps=eps, min_samples=min_samples).fit(candidate_lines)
for label in np.unique(model.labels_):
    if label == -1:
        continue
    theta, rho = np.mean(candidate_lines[model.labels_ == label], axis=0)
    img = draw_hough_line(img, theta, rho, color=100)
    yield theta, rho
return
```

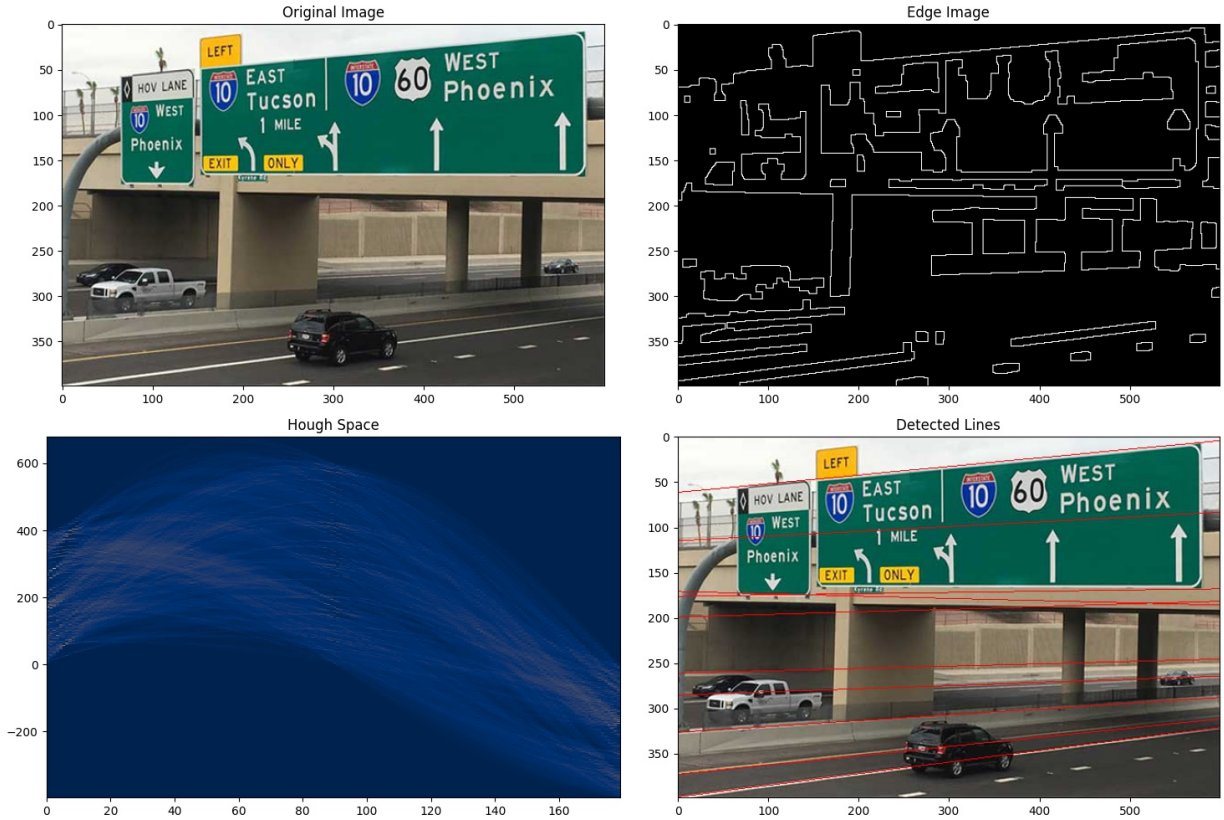
Last, merge those close points, which are almost the same line. I tried many different model, DBSCAN is the most suitable. It can not only cluster correctly but also be good at dealing with noise.

Result



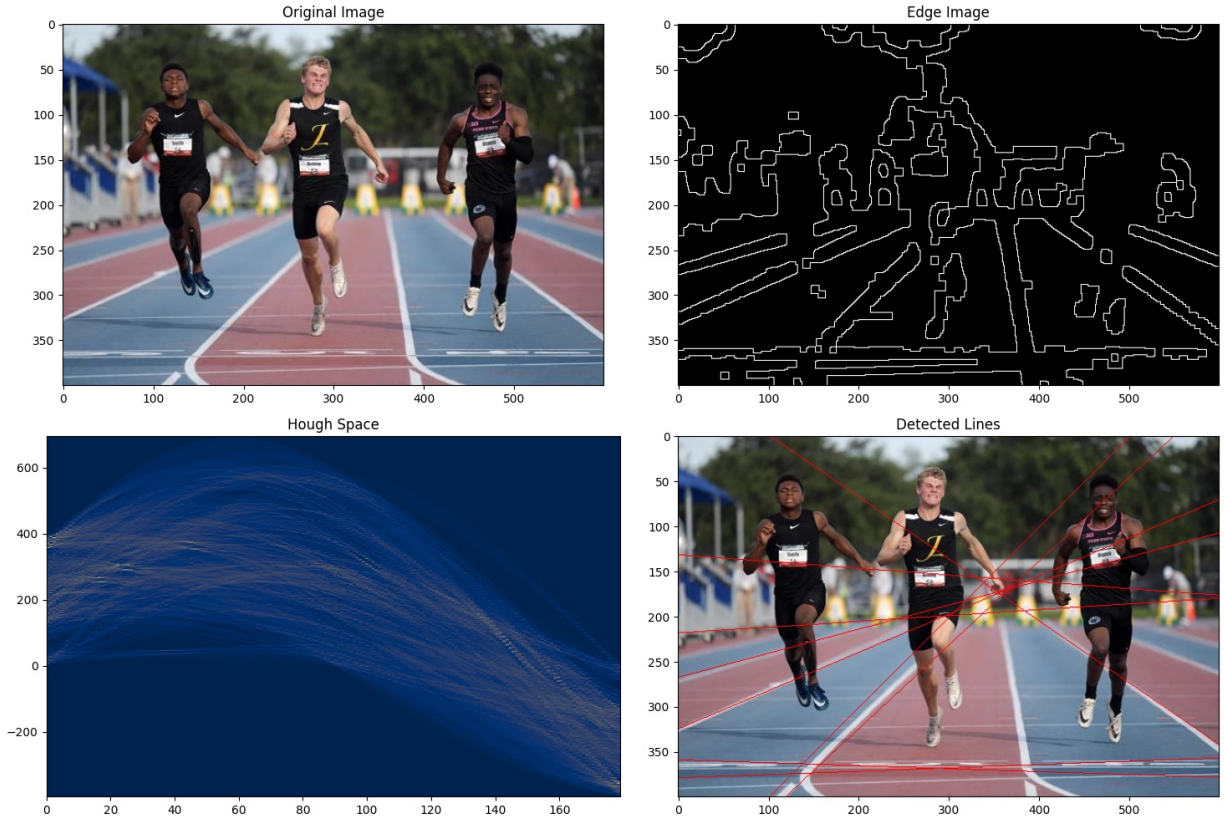

```
hough done in 0.6567180156707764 sec.
re-structure done in 1.1668298244476318 sec.
sort done in 0.0567779541015625 sec.
cluster done in 0.0013828277587890625 sec.
```

2.




```
hough done in 0.5764601230621338 sec.
re-structure done in 1.461313009262085 sec.
sort done in 0.057756900787353516 sec.
cluster done in 0.0012218952178955078 sec.
```

3.



```
hough done in 0.5234880447387695 sec.
re-structure done in 1.2605960369110107 sec.
sort done in 0.05410480499267578 sec.
cluster done in 0.0018498897552490234 sec.
```

Circle Hough Transform(CHT)

Like **Hough Transform** which form each different line with 2 parameters (θ, ρ), **CHT** also discribes circle with 3 parameters: (a, b, r), where (a, b) is the center position of circle, and r represents the radius.

```
img = cv2.resize(img, (0, 0), fx=scale_rate, fy=scale_rate)
hough = get_hough_circle(img, r_min*scale_rate, r_max*scale_rate, threshold*scale_rate)
```

Because **CHT** is really way more time consuming than **HT**, I tried some little tricks to reduce the time cost. And scaling down image is a quick and eazy way to do it. Sometimes the image is too big and the edge pixels are too much, but the circle features are there no matter what size it is, so I figure out this method, and it's pretty useful.

Hough Space

We basically do the same thing as previous technique but in 3D because of one more parameter. We generate discrete finite possible circle parameters with constraints: *r_min, r_max, r_step, t_step* ...

```
@njit(cache=True, nogil=True)
def get_hough_circle(img, r_min, r_max, threshold=150, r_step=1, t_step=1):
  unlabel_points = (img >= threshold)
  hough = []
  for r in np.arange(r_min, r_max, r_step):
    for i, j in zip(*unlabel_points.nonzero()):
      for theta in np.arange(0, 360, t_step):
        a = i - r * np.cos(theta * np.pi / 180)
        b = j - r * np.sin(theta * np.pi / 180)
        hough.append((a, b, r))
        return hough
```

Here, we also use *numba(@njit)* accelerate the computing because it's really time consuming. I actually did a little expirement testing how long it'll take without accelerator. And it can speed up **80x**.

```
hough time with @njit: 6 sec
hough time without @njit: 8 min
```

Accumulator

```
accumulator = defaultdict(int)
for a, b, r in hough:
    accumulator[(int(a), int(b), int(r))] += 1
filtered_points = {k: v for k, v in sorted(accumulator.items(),
                                           key=lambda item: item[1], reverse=True)
                   if v >= sample_threshold}
```

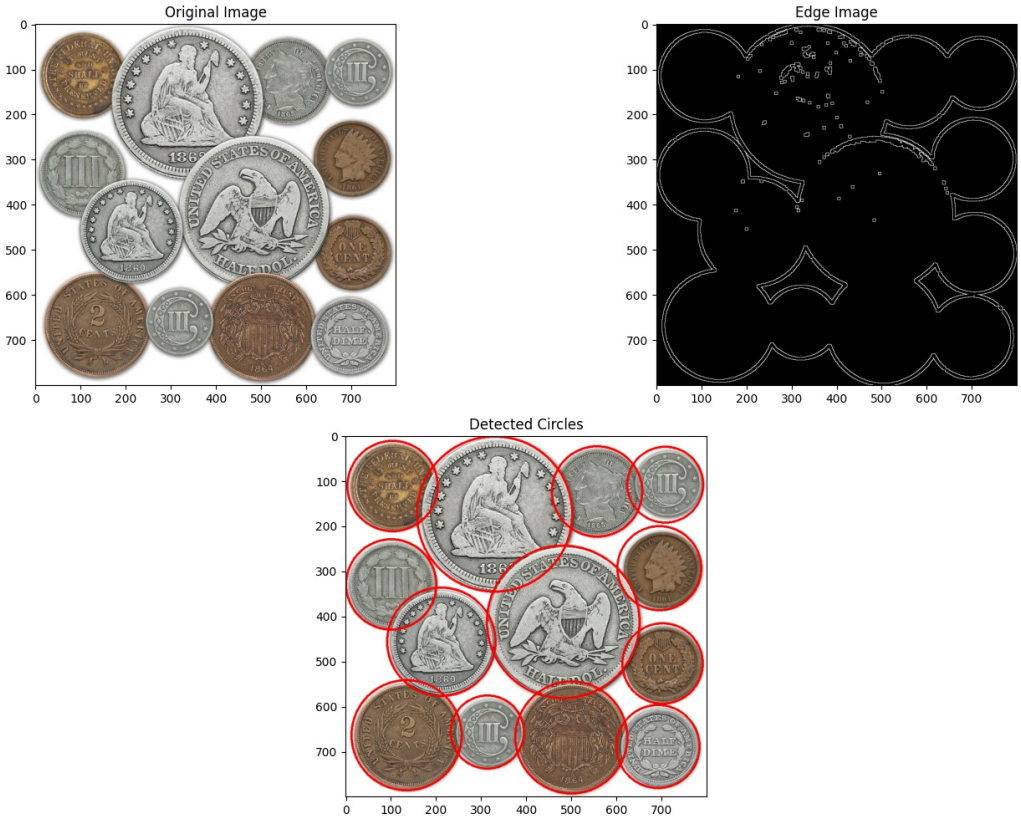
Since the parameter space is 3D, the accumulator matrix would be 3D, too. We can iterate through possible radii; for each radius, we use the previous technique. Finally, find the local maxima in the 3D accumulator matrix. Accumulator array should be $A[x, y, r]$ in the 3D space. Voting should be for each pixels, radius and theta $A[x, y, r] = A[x, y, r] + 1$.

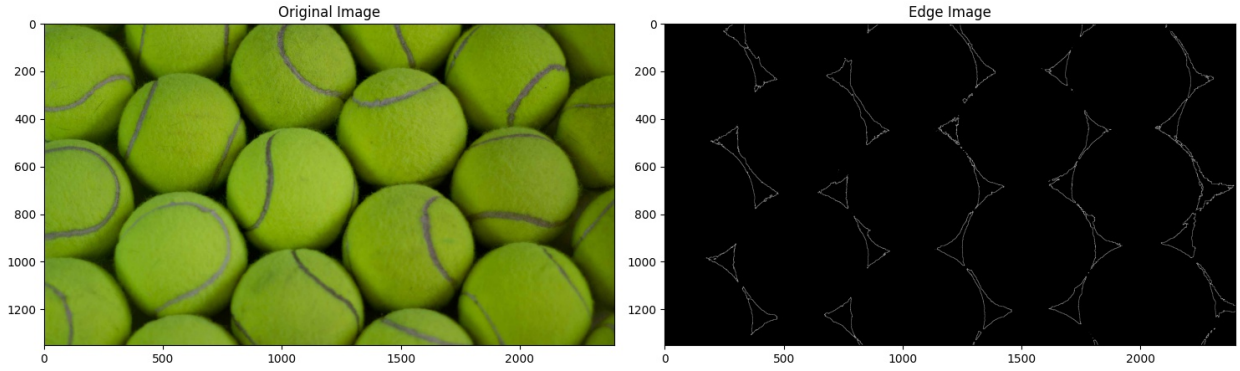
Clustering

```
model = DBSCAN(eps=eps*scale_rate, min_samples=min_samples)
candidate_circle_center = np.array(list(filtered_points.keys())).reshape(-1, 3)
del filtered_points
model.fit(candidate_circle_center)
for label in np.unique(model.labels_):
    if label == -1:
        continue
    a, b, r = np.mean(candidate_circle_center[model.labels_ == label], axis=0)
    yield a/scale_rate, b/scale_rate, r/scale_rate
return
```

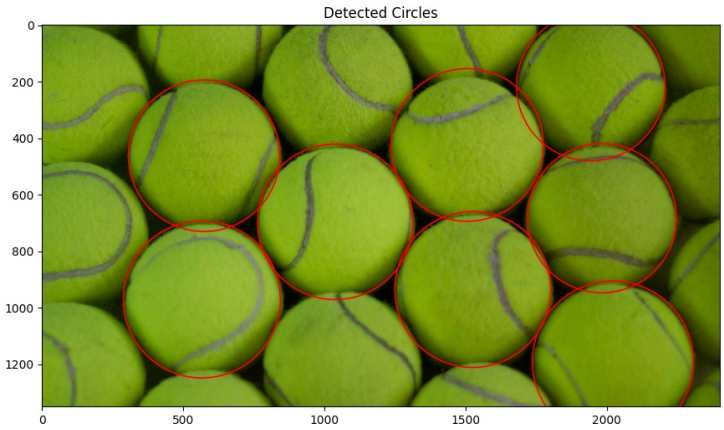
Last, we also use ***DBSCAN*** for clustering.

Result

1. 


Original Image
Edge Image
0
200
400
600
800
1000
1200
0
500
1000
1500
2000

2.


Detected Circles
0
200
400
600
800
1000
1200
0
500
1000
1500
2000

Code List

```python
import time
from collections import defaultdict

import cv2
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
from numba import njit
from sklearn.cluster import DBSCAN

from hw1.main import NeighborhoodOp, PixelOp


class Generator:
    def __init__(self, gen):
        self.gen = gen

    def __iter__(self):
        self.value = yield from self.gen


@njit()
def drawHoughLine(img, theta, rho, color=(255, 0, 0)):
    if 45 <= theta <= 135:
        for x in range(img.shape[0]):
            y = int((rho - x * np.cos(theta / 180 * np.pi)) // np.sin(theta / 180 * np.pi))
            if 0 <= y < img.shape[1]:
                img[x, y] = color
    else:
        for y in range(img.shape[1]):
            x = int((rho - y * np.sin(theta / 180 * np.pi)) / np.cos(theta / 180 * np.pi))
            if 0 <= x < img.shape[0]:
                img[x, y] = color
    return img


def getHoughLines(img, sample_threshold=int(5e4), eps=3, min_samples=5, **kwargs):
    @njit()
    def getHough(img, threshold, t_step):
        unlabel_points = (img >= threshold)
        hough = []
        for x, y in zip(*unlabel_points.nonzero()):
            for theta in np.arange(0, 180, t_step):
                rho = x * np.cos(theta / 180 * np.pi) + y * np.sin(theta / 180 * np.pi)
                hough.append((theta, rho, img[x, y]))
```

```
            return hough

        t = time.time()
        hough_space = getHough(img, **kwargs)
        elapsed = time.time() - t
        print(f'hough done in {elapsed} sec.')

        t = time.time()
        accumulator = defaultdict(int)
        all_sample = []
        for (theta, rho, weight) in hough_space:
            accumulator[(int(theta), int(rho))] += 1
            all_sample.append((theta, rho))

        del hough_space
        elapsed = time.time() - t
        print(f"re-structure done in {elapsed} sec.")

        t = time.time()
        filtered_points = {k: v for k, v in sorted(accumulator.items(),
                                                   key=lambda item: item[1], reverse=True)
                           if v >= sample_threshold}
        del accumulator
        elapsed = time.time() - t
        print(f"sort done in {elapsed} sec.")

        t = time.time()
        candidate_lines = np.array(list(filtered_points.keys())).reshape(-1, 2)
        del filtered_points
        model = DBSCAN(eps=eps, min_samples=min_samples).fit(candidate_lines)
        elapsed = time.time() - t
        print(f"cluster done in {elapsed} sec.")

        for label in np.unique(model.labels_):
            if label == -1:
                continue
            theta, rho = np.mean(candidate_lines[model.labels_ == label], axis=0)
            # img = drawHoughLine(img, theta, rho, color=100)
            yield theta, rho
        return all_sample


    def draw_hough_circle(img, a, b, r, color=(255, 0, 0)):
        for theta in np.arange(0, 360, 1):
            x = a + r * np.cos(theta / 180 * np.pi)
            y = b + r * np.sin(theta / 180 * np.pi)
            if (0 <= x < img.shape[0]) and (0 <= y < img.shape[1]):
                img[int(x), int(y)] = color
        return img
```

```python
def hough_circle(img, r_min, r_max, threshold=140, sample_threshold=80, eps=1, min_samples=8, scale_rate=0.5):
    @njit(cache=True, nogil=True)
    def get_hough_circle(img, r_min, r_max, threshold=150., r_step=1, t_step=1):
        unlabel_points = (img >= threshold)
        hough = []
        for r in np.arange(r_min, r_max, r_step):
            for i, j in zip(*unlabel_points.nonzero()):
                for theta in np.arange(0, 360, t_step):
                    a = i - r * np.cos(theta * np.pi / 180)
                    b = j - r * np.sin(theta * np.pi / 180)
                    hough.append((a, b, r))
        return hough

    t = time.time()
    img = cv2.resize(img, (0, 0), fx=scale_rate, fy=scale_rate)
    hough = get_hough_circle(img, r_min * scale_rate, r_max * scale_rate, threshold * scale_rate)
    elapsed = time.time() - t
    print(f'hough done in {elapsed} sec.')

    t = time.time()
    accumulator = defaultdict(int)
    for a, b, r in hough:
        accumulator[(int(a), int(b), r)] += 1
    del hough
    elapsed = time.time() - t
    print(f"re-structure done in {elapsed} sec.")

    t = time.time()
    filtered_points = {k: v for k, v in sorted(accumulator.items(),
                                               key=lambda item: item[1], reverse=True)
                       if v >= sample_threshold}
    del accumulator
    elapsed = time.time() - t
    print(f"sort done in {elapsed} sec.")

    # Cluster
    model = DBSCAN(eps=eps * scale_rate, min_samples=min_samples)
    candidate_circle_center = np.array(list(filtered_points.keys())).reshape(-1, 3)
    del filtered_points
    model.fit(candidate_circle_center)
    for label in np.unique(model.labels_):
        if label == -1:
            continue
        a, b, r = np.mean(candidate_circle_center[model.labels_ == label], axis=0)
        # img = draw_hough_circle(img, a, b, r, color=100)
        yield a / scale_rate, b / scale_rate, r / scale_rate
    return


def export_result(image=None, edge=None, samples=None, result=None, filename=None):
```

```
    figure = plt.figure(figsize=(15, 10))
    subplot1 = figure.add_subplot(2, 2, 1)
    subplot1.title.set_text("Original Image")
    subplot1.imshow(image)

    subplot2 = figure.add_subplot(2, 2, 2)
    subplot2.title.set_text("Edge Image")
    subplot2.imshow(edge, 'gray')

    if samples is not None:
        x = np.array(samples).reshape(-1, 2)[:, 0]
        y = np.array(samples).reshape(-1, 2)[:, 1]
        subplot3 = figure.add_subplot(2, 2, 3)
        subplot3.title.set_text("Hough Space")
        subplot3.hist2d(x, y, bins=(180, 600), cmap='cividis')

        subplot4 = figure.add_subplot(2, 2, 4)
        subplot4.title.set_text("Detected Lines")
        subplot4.imshow(result)
    else:
        subplot4 = figure.add_subplot(2, 1, 2)
        subplot4.title.set_text("Detected Circles")
        subplot4.imshow(result)

    plt.tight_layout()
    plt.savefig('output/' + filename)


if __name__ == '__main__':
    kernel = np.ones((3, 3))
    # load image
    img_signs = np.array(Image.open('input/signs.jpg').convert('RGB'))
    gray_signs = np.array(Image.open('input/signs.jpg').convert('L'))

    # get edge
    blur_signs = NeighborhoodOp.filtering(gray_signs, mode='gaussian')
    mask = PixelOp.GrayLevelTransform.threshold(blur_signs, 150, 190, binary=True,
invert=False)
    mask = NeighborhoodOp.morphologyEx(mask, kernel=kernel, mode='custom1')
    erosion = mask
    for _ in range(2):
        dilation = NeighborhoodOp.morphologyEx(erosion, kernel=kernel, mode='dilation',
iteration=4)
        erosion = NeighborhoodOp.morphologyEx(dilation, kernel=kernel, mode='erosion',
iteration=2)
    edge_signs = NeighborhoodOp.filtering(erosion, mode='laplacian-
corner').astype(np.uint8)

    # get lines
    hough_lines = Generator(getHoughLines(edge_signs, threshold=150, sample_threshold=80,
                                          eps=8, min_samples=3, t_step=1))
    result_signs = img_signs.copy()
```

```
    for theta, rho in hough_lines:
        result_signs = drawHoughLine(result_signs, theta, rho)
    export_result(img_signs, edge_signs, hough_lines.value, result_signs, 'out_sign.jpg')

    # 2.
    # load image
    img_crossing = np.array(Image.open('input/crossing.jpg').convert('RGB'))
    gray_crossing = np.array(Image.open('input/crossing.jpg').convert('L'))

    # get edge
    blur_crossing = NeighborhoodOp.filtering(gray_crossing, mode='gaussian')
    mask_crossing = PixelOp.GrayLevelTransform.threshold(blur_crossing, 130, 160, binary=True, invert=False)
    mask_crossing = NeighborhoodOp.morphologyEx(mask_crossing, kernel=kernel, mode='custom1')
    erosion_crossing = mask_crossing
    for _ in range(2):
        dilation_crossing = NeighborhoodOp.morphologyEx(erosion_crossing, kernel=kernel, mode='dilation', iteration=4)
        erosion_crossing = NeighborhoodOp.morphologyEx(dilation_crossing, kernel=kernel, mode='erosion', iteration=2)
    edge_crossing = NeighborhoodOp.filtering(erosion_crossing, mode='laplacian-corner').astype(np.uint8)

    # get lines
    result_crossing = img_crossing.copy()
    hough_lines = Generator(getHoughLines(edge_crossing, threshold=150, sample_threshold=80,
                                          eps=8, min_samples=3, t_step=1))
    for theta, rho in hough_lines:
        result_crossing = drawHoughLine(result_crossing, theta, rho)
    export_result(img_crossing, edge_crossing, hough_lines.value, result_crossing, 'out_crossing.jpg')

    # 3.
    # load data
    img_sport = np.array(Image.open('input/sport.jpg').convert('RGB'))
    gray_sport = np.array(Image.open('input/sport.jpg').convert('L'))

    # get edge
    blur_sport = NeighborhoodOp.filtering(gray_sport, mode='gaussian')
    mask_sport = PixelOp.GrayLevelTransform.threshold(blur_sport, 160, 170, binary=True, invert=False)
    erosion_sport = mask_sport
    for _ in range(2):
        dilation_sport = NeighborhoodOp.morphologyEx(erosion_sport, kernel=kernel, mode='dilation', iteration=4)
        erosion_sport = NeighborhoodOp.morphologyEx(dilation_sport, kernel=kernel, mode='erosion', iteration=2)

    edge_sport = NeighborhoodOp.filtering(erosion_sport, mode='laplacian-corner').astype(np.uint8)
```

```
236
237     # get lines
238     result_sport = img_sport.copy()
239     hough_lines = Generator(getHoughLines(edge_sport, threshold=150, sample_threshold=60,
240                                           eps=9, min_samples=8, t_step=1))
241     for theta, rho in hough_lines:
242         result_sport = drawHoughLine(result_sport, theta, rho)
243     export_result(img_sport, edge_sport, hough_lines.value, result_sport,
    'out_sport.jpg')
244
245     # CHT
246     # 1.
247     # load image
248     img_coins = np.array(Image.open('input/coins.jpg').convert('RGB'))
249     gray_coins = np.array(Image.open('input/coins.jpg').convert('L'))
250
251     # get edge
252     mask_coins = PixelOp.GrayLevelTransform.threshold(gray_coins, 215, 245, binary=True,
    invert=True)
253     kernel = np.ones((3, 3))
254     erosion_coins = mask_coins
255     for _ in range(2):
256         dilation_coins = NeighborhoodOp.morphologyEx(erosion_coins, kernel,
    mode='dilation')
257         erosion_coins = NeighborhoodOp.morphologyEx(dilation_coins, kernel,
    mode='erosion')
258     erosion_coins = NeighborhoodOp.morphologyEx(erosion_coins, kernel, mode='erosion')
259     edge_coins = NeighborhoodOp.filtering(erosion_coins, mode='laplacian-
    corner').astype(np.uint8)
260
261     # get circles
262     result_coins = img_coins.copy()
263     for a, b, r in hough_circle(edge_coins, 80, 200, threshold=80, sample_threshold=60,
264                                 eps=6, min_samples=4, scale_rate=.25):
265         cv2.circle(result_coins, (int(b), int(a)), int(r), (255, 0, 0), 3)
266     export_result(img_coins, edge_coins, result=result_coins, filename='out_coins.jpg')
267
268     # 2.
269     # load image
270     img_balls = np.array(Image.open('input/ball3.jpg').convert('RGB'))
271     gray_balls = np.array(Image.open('input/ball3.jpg').convert('L'))
272
273     # get edge
274     blur_balls = NeighborhoodOp.filtering(gray_balls, mode='gaussian')
275     mask_balls = PixelOp.GrayLevelTransform.threshold(blur_balls, 40, 190, binary=True,
    invert=True)
276     kernel = np.ones((3, 3))
277     mask_balls = NeighborhoodOp.morphologyEx(mask_balls, kernel=kernel, mode='custom1')
278     erosion_balls = mask_balls
279     for _ in range(2):
280         dilation_balls = NeighborhoodOp.morphologyEx(mask_balls, kernel, mode='dilation',
    iteration=1)
```

```
281         erosion_balls = NeighborhoodOp.morphologyEx(dilation_balls, kernel,
    mode='erosion', iteration=1)
282
283     edge_balls = NeighborhoodOp.filtering(erosion_balls, mode='laplacian-
    corner').astype(np.uint8)
284
285     # get circles
286     result_balls = img_balls.copy()
287     for a, b, r in hough_circle(edge_balls, 200, 300, threshold=80, sample_threshold=25,
288                                 eps=10, min_samples=3, scale_rate=.2):
289         cv2.circle(result_balls, (int(b), int(a)), int(r), (255, 0, 0), 3)
290
291     export_result(img_balls, edge_balls, samples=None, result=result_balls,
    filename='out_balls.jpg')
292
```