Project 3

Computer Vision - CS 6643

Out: Nov 13, 2023
Due: Nov 30, 2023 (deadline: 11:59 PM)

A1) Hough Transform for Straight Lines without edge orientation

A2) Creative Part: Hough Transform for Circles

Nidhushan Kanagaraja (N10852881)

A1) Hough Transform for straight lines without edge orientation

1.1 Introduction

Hough Transform is like having a voting system where each point contributes to the possibility of there being a line, and the lines with the most votes are considered as the lines present in the data. This method is helpful when you're not sure about the orientation of the lines you're looking for.

Voting System: Each point on the paper "votes" for possible lines that might pass through it. The more votes a line gets, the more likely it is to be a straight line.

Parameter Space: We use a grid to represent all possible lines. One axis of the grid represents the angle of the line, and the other represents the distance from the origin. Each cell in this grid represents a potential line.

Accumulator Array: As the points vote, the corresponding cells in the grid (accumulator array) get more votes. The cells with the most votes indicate the possible lines.

Thresholding: We look for cells with a high number of votes. These cells correspond to the most likely lines in the data.

1.2 Edge Detection

We use the same edge detection techniques used in project 2 to compute the binary edges of the images.

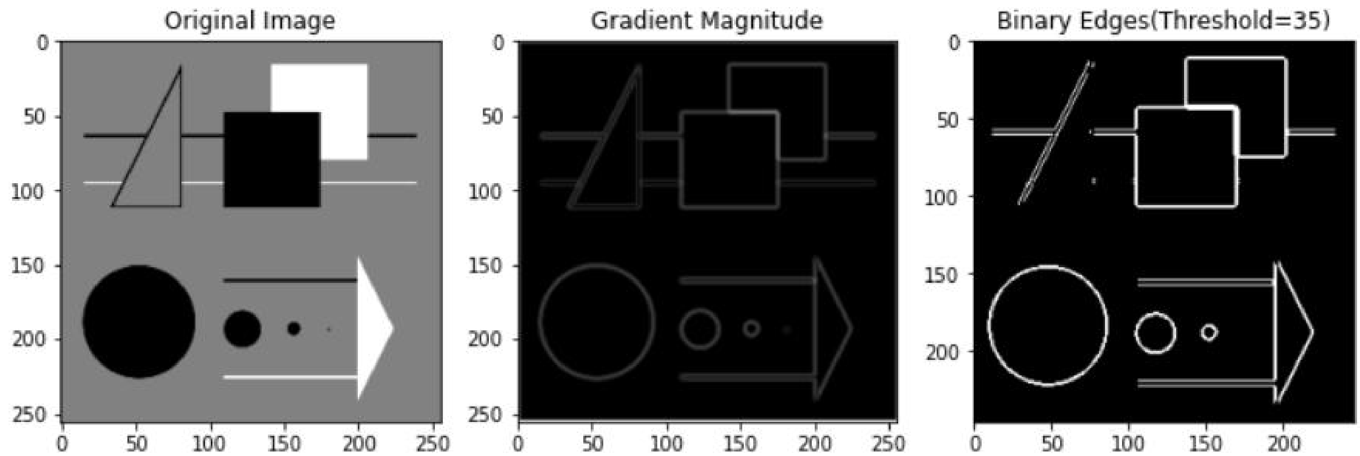


Fig. 1.1 Binary edge image of the original image edges-lines

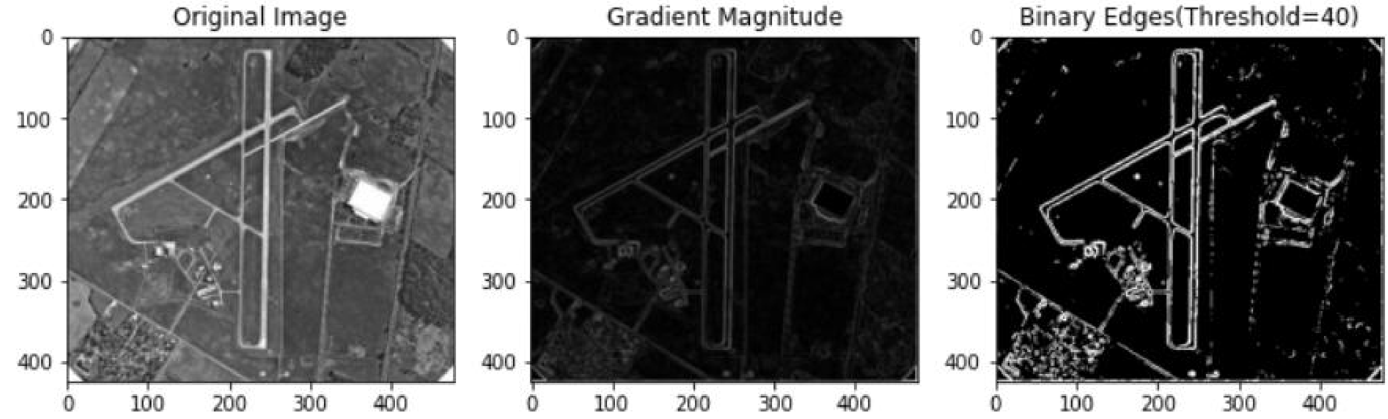


Fig. 1.2 Binary edge image of Original Image runway-ohio

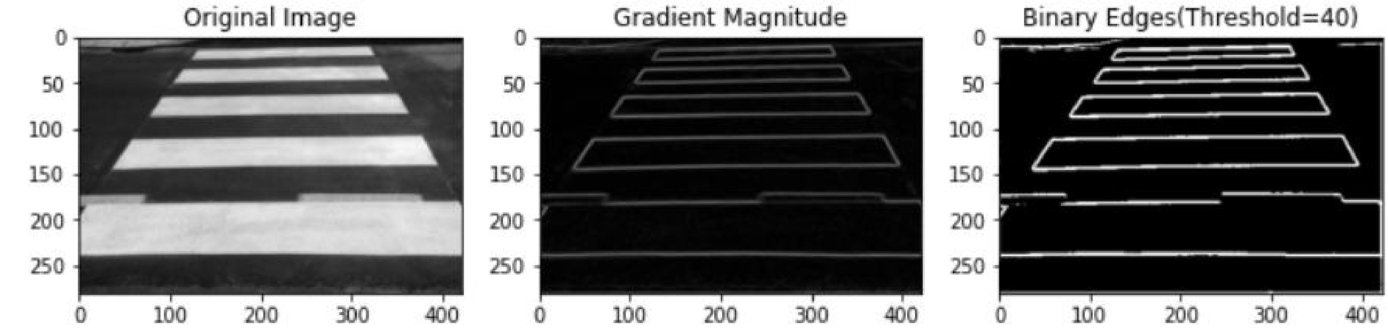


Fig. 1.3 Binary Edge Image of Original Image Crosswalk

1.3 Hough Transform

To explain in detail, these are the steps we take for implementing a hough transform:

We set up the Hough space, which is a mathematical space used to represent possible lines in the image. This involves defining the range of possible line angles(theta_values) and distances from the origin(rho_values). An accumulator is created as a 2D array to accumulate votes for possible lines in the Hough space.

Fig. 1.4 Accumulator for edge-lines

Fig. 1.5 Accumulator for Crosswalk

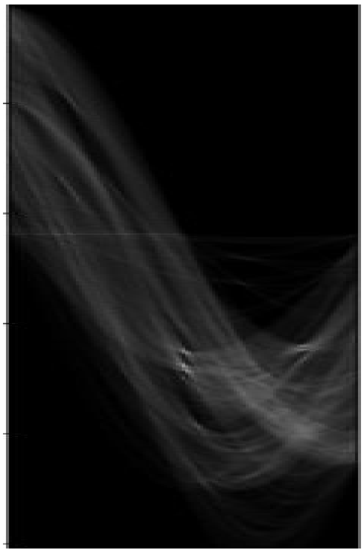


Fig. 1.6 Accumulator for runway-ohio

For each edge pixel in the binary image, we calculate the corresponding lines in the Hough space and accumulate votes. This is done by iterating over the edge pixels, converting them to Hough space, and updating the accumulator.

We then identify potential peaks in the accumulator, which represent possible lines in the image. We check each point in the accumulator against its neighbors to find local maxima. The identified peaks are sorted based on the number of accumulated votes, and the top 15 or so peaks are selected. These peaks correspond to the most prominent lines in the image. The function then draws lines on a copy of the original image using the parameters of the selected peaks.

Fig. 1.7 Line image for Crosswalk

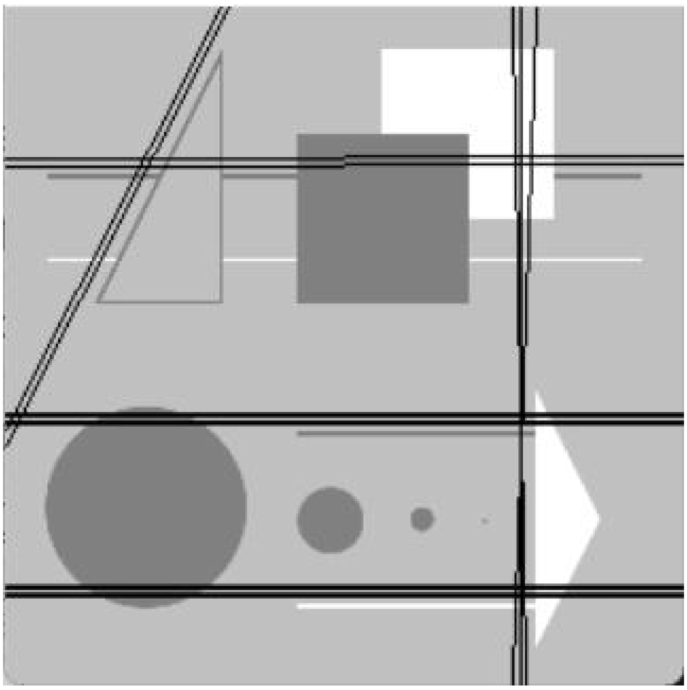


Fig. 1.8 Line image for edge-lines

Fig. 1.9 Line image for runway ohio

As observed the results are not perfect. But when we do edge detection using library functions we get a better result, as shown below. Therefore the issue must be in the edge detection part.

Fig. 1.10 Line image of Crosswalk with library edge detection

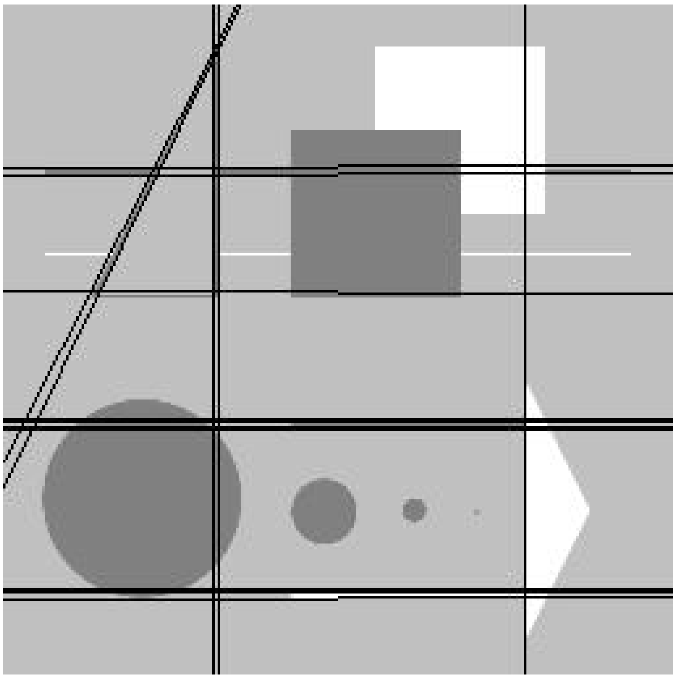


Fig. 1.11 Line image of Edge-lines with library edge detection

Fig. 1.12 Line image of runway ohio with library edge detection

A2) Creative Part - Hough Transform for Circles without edge orientation

2.1 Introduction

Hough Transform for Circles are quite similar to Lines where we change a few things.

- In the Hough Transform for lines, the parameters are the slope (or angle) and the intercept of a line.
- The Hough space for lines is a 2D space representing the possible values of slope and intercept and for circles it is a 3D space representing the possible values of x, y, and r.
- The accumulator array is a 3D grid where each cell represents a circle in parameter space.
- Peaks in the accumulator array represent circles in the image.

2.2 Edge Detection

The edge detection part will be the same here, we don't do anything different.

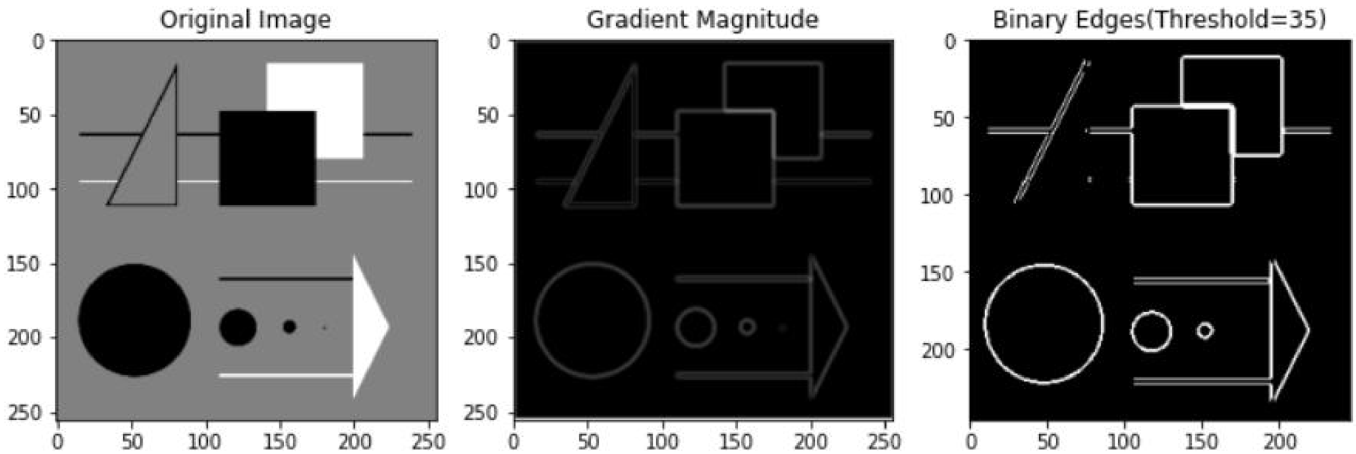


Fig. 2.1 Binary edge image of the original image edges-lines

In this image, we try to find the circles using hough transform.

2.3 Hough Transform

We change the function from a line function to a circle's function. Instead of the usual y=mx+c that we use for lines, we use the following function:

$$(x-a)^2+(y-b)^2=r^2$$

a is the x-coordinate of the circle center.
b is the y-coordinate of the circle center.
r is the radius of the circle.

As expected, for the circle Hough Transform, we see a change in the accumulator as well although the edges and image are the same.

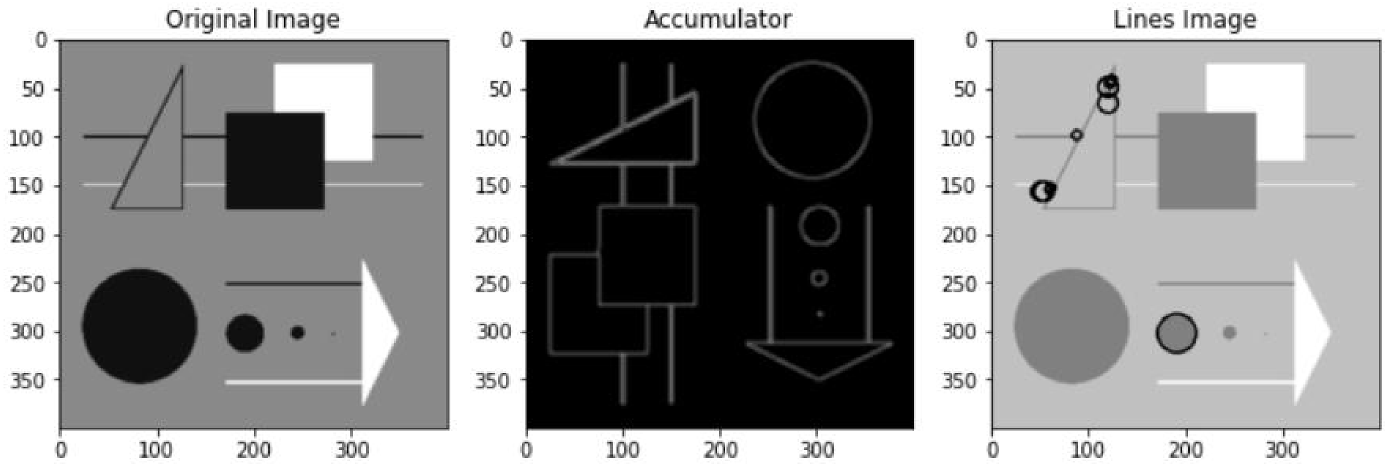


Fig. 2.2 Original vs Accumulator vs Line Image for edge-lines image(r=10-20)

Again the results are a bit inaccurate compared to the results from using library functions:

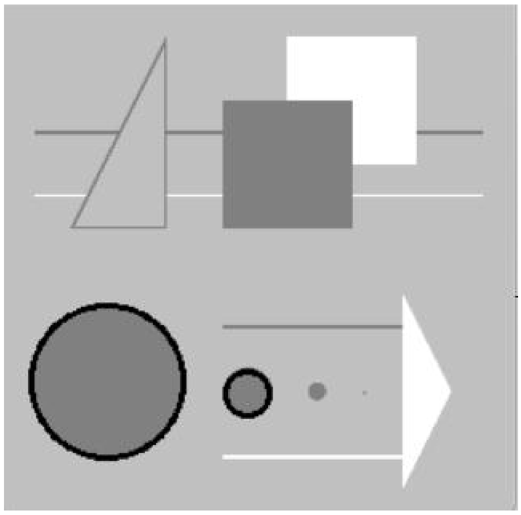


Fig. 2.3 Result of Hough Transform using library functions for edge-lines image

Furthermore, it was quite inefficient to calculate the hough transform of multiple radii as compared to using the hough transform using library functions.

Appendix

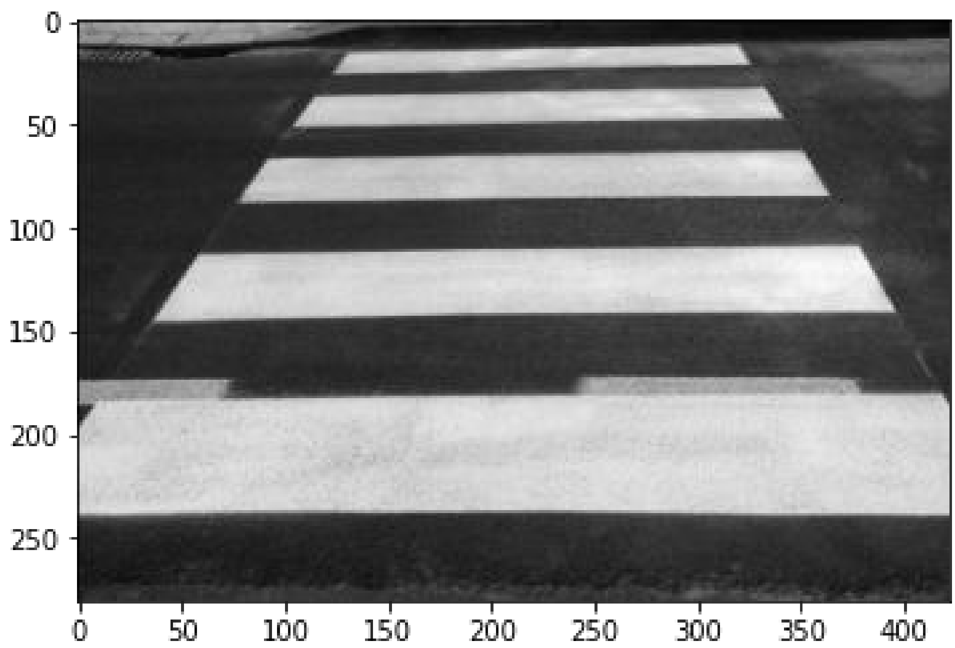
Computer Vision Project 3

1. Hough Transform for straight lines without edge orientation

Binary Edge Detection

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
def convolution(f, I):
    f_height, f_width = f.shape
    I_height, I_width = I.shape
    pad_height, pad_width = f_height // 2, f_width // 2

    I_padded = np.pad(I, ((pad_height, pad_height), (pad_width, pad_width)),
mode='constant')

    im_conv = np.zeros(I.shape, dtype=np.float32)

    for i in range(I_height):
        for j in range(I_width):
            im_conv[i, j] = np.sum(f * I_padded[i:i + f_height, j:j + f_width])

    return im_conv
image = cv2.imread('crosswalk.png', cv2.IMREAD_GRAYSCALE)
plt.imshow(image, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x1afcec7ab50>
```

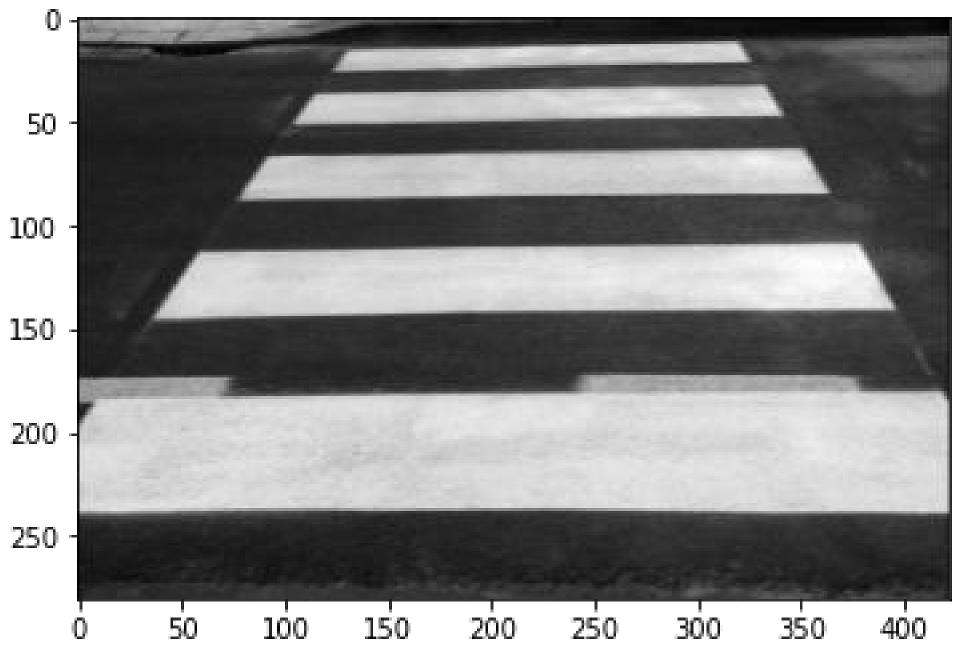


```
sigma = 3
x_range = np.linspace(-int(sigma/2),int(sigma/2),sigma)
# print(x_range)
gaussian_filter = [ (1 / (sigma * np.sqrt(2*np.pi)) * np.exp(-x**2/(2*sigma**2))) for
x in x_range ]
total = sum(gaussian_filter)
gaussian_filter = [[x/total for x in gaussian_filter]]
Gx = np.array(gaussian_filter)
Gy = Gx.reshape(-1,1)

print("Gx =",Gx)
print("Gy =",Gy)
```

```
Gx = [[0.32710442 0.34579116 0.32710442]]
Gy = [[0.32710442]
 [0.34579116]
 [0.32710442]]
```

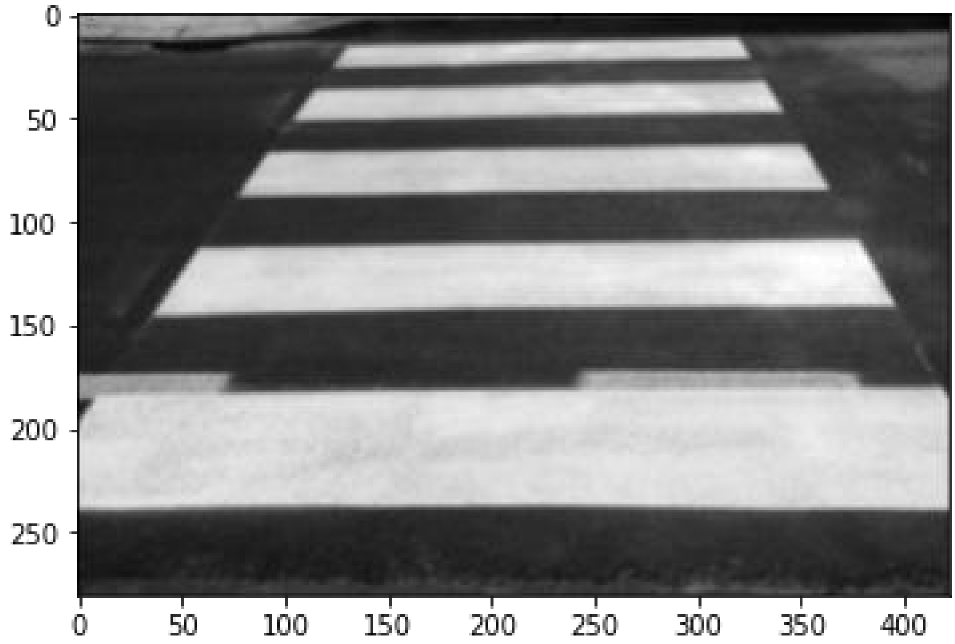
```
image_filtered_x = convolution(Gx, image)
image_filtered_y = convolution(Gy, image_filtered_x)

plt.imshow(image_filtered_x, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x1afd0d69280>
```

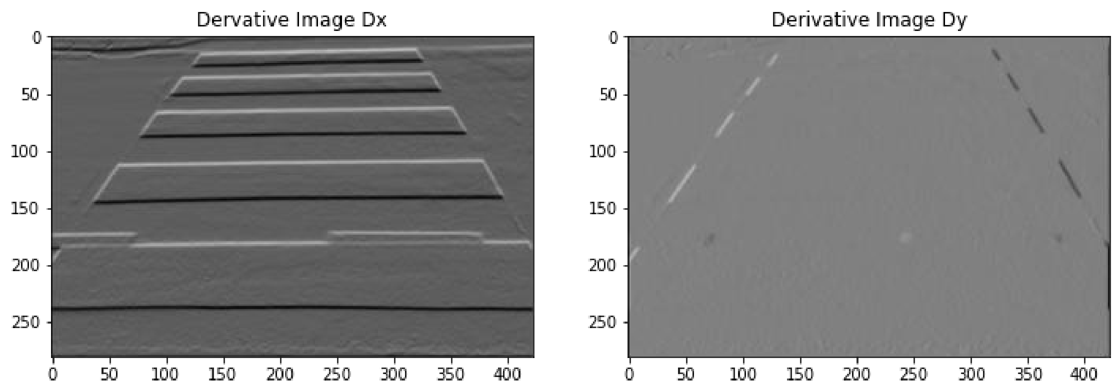



```
plt.imshow(image_filtered_y, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x1afd0de0820>
```

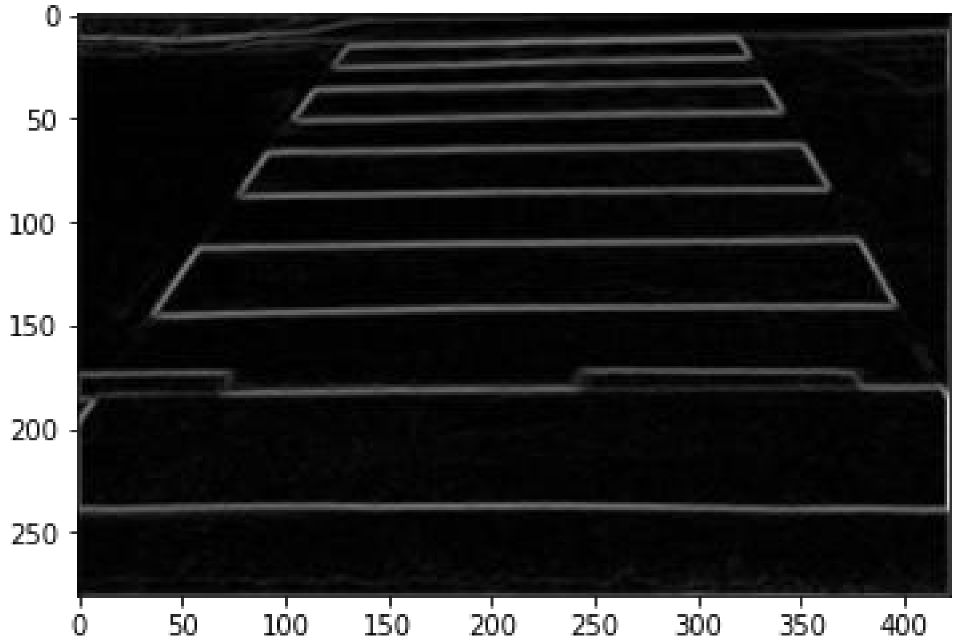


```
derivative_filter_x = np.array([[-1], [0], [1]])
derivative_filter_y = derivative_filter_x.reshape((1, -1))

image_dx = convolution(derivative_filter_x, image_filtered_y)
image_dy = convolution(derivative_filter_y, image_filtered_y)
plt.figure(figsize=(12, 12))

plt.subplot(1, 2, 1)
plt.imshow(image_dx, cmap='gray')
plt.title('Dervative Image Dx')

plt.subplot(1, 2, 2)
plt.imshow(image_dy, cmap='gray')
plt.title('Derivative Image Dy')

plt.show()
```

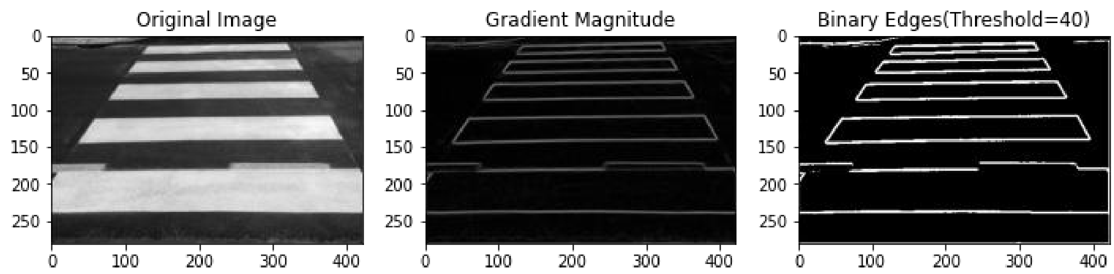



```
gradient_magnitude = np.sqrt(image_dx ** 2 + image_dy ** 2)
plt.imshow(gradient_magnitude, cmap='gray')
```

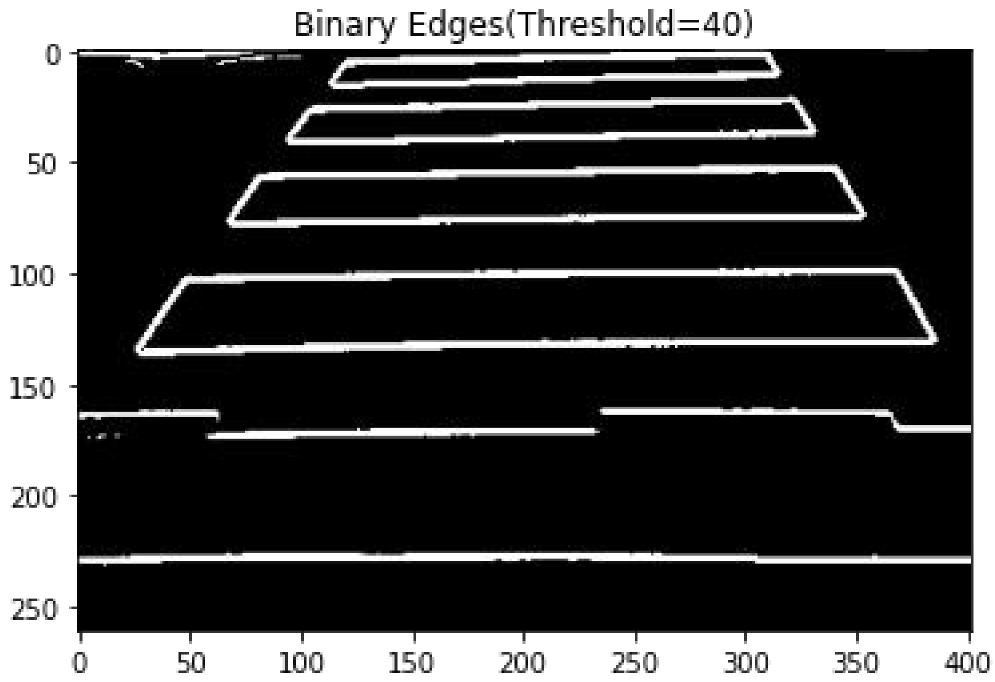
<matplotlib.image.AxesImage at 0x1afd0f255b0>




```python
threshold = 40
binary_edges = (gradient_magnitude > threshold).astype(np.uint8) * 255

cv2.imshow('Original Image', image.astype(np.uint8))
cv2.imshow('Gradient Magnitude', gradient_magnitude.astype(np.uint8))
cv2.imshow('Binary Edges', binary_edges)
cv2.waitKey(0)
cv2.destroyAllWindows()

plt.figure(figsize=(12, 12))
plt.subplot(1, 3, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')

plt.subplot(1, 3, 2)
plt.imshow(gradient_magnitude, cmap='gray')
plt.title('Gradient Magnitude')

plt.subplot(1, 3, 3)
plt.imshow(binary_edges, cmap='gray')
plt.title('Binary Edges(Threshold=40)')

plt.show()
```

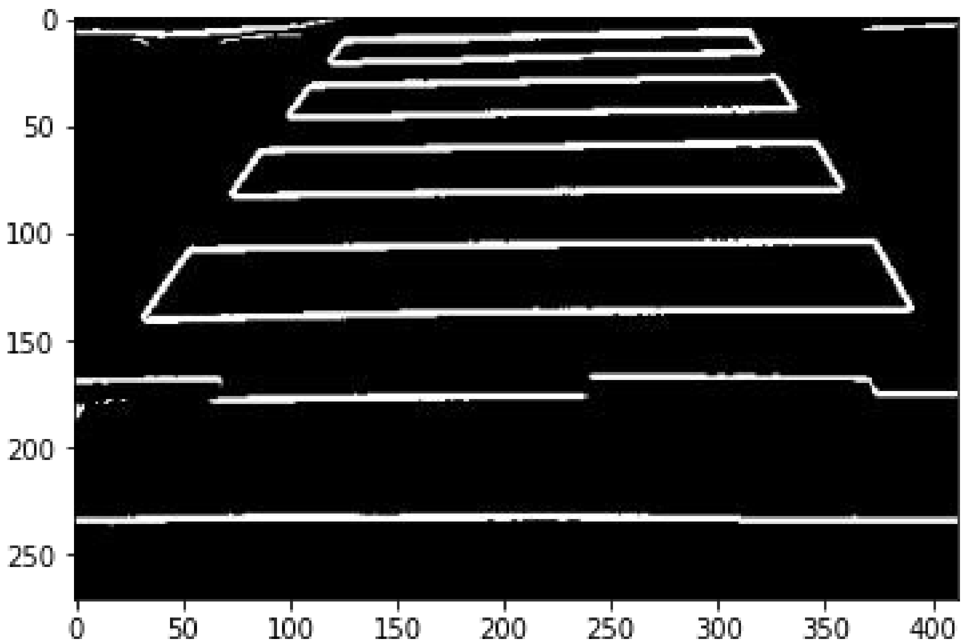


```
binary_edges = binary_edges[5:-5, 5:-5]

plt.imshow(binary_edges, cmap='gray')
plt.title('Binary Edges(Threshold=40)')
Text(0.5, 1.0, 'Binary Edges(Threshold=40)')
```



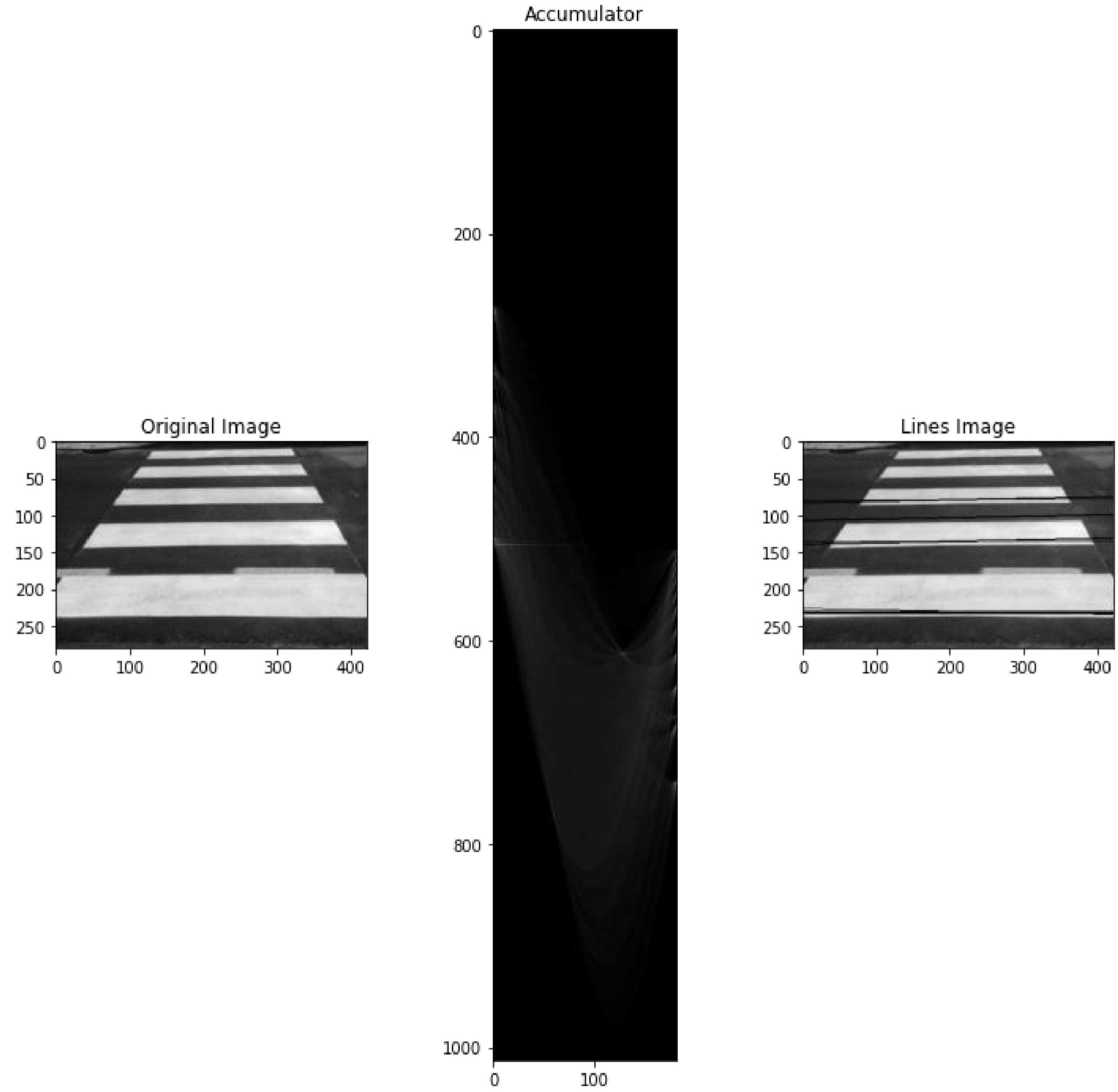

```
def hough_transform(image, delta_rho=1, delta_theta=1, threshold=10, num_peaks=15):
    edges = binary_edges
    plt.imshow(edges, cmap='gray')

    max_rho = int(np.sqrt(np.square(image.shape[0]) + np.square(image.shape[1])))
    theta_values = np.deg2rad(np.arange(-90, 91, delta_theta))
    rho_values = np.arange(-max_rho, max_rho + 1, delta_rho)

    accumulator = np.zeros((len(rho_values), len(theta_values)), dtype=np.uint64)

    y_nonz, x_nonz = np.nonzero(edges)
    for i in range(len(x_idxs)):
        x = x_nonz[i]
        y = y_nonz[i]
        for j in range(len(theta_values)):
            rho = int(x * np.cos(theta_values[j]) + y * np.sin(theta_values[j]))
            rho_ind = np.argmin(np.abs(rho_values - rho))
            accumulator[rho_idx, j] += 1

    peaks = []

    for i in range(1, accumulator.shape[0] - 1):
        for j in range(1, accumulator.shape[1] - 1):
            if accumulator[i, j] > accumulator[i - 1:i + 2, j - 1:j + 2].max():
                peaks.append((i, j))

    peaks = sorted(peaks, key=lambda x: accumulator[x[0], x[1]],
reverse=True)[:num_peaks]
```

```
    peaks = np.column_stack(np.unravel_index(np.argsort(accumulator.ravel())[-
num_peaks:], accumulator.shape))

    lines_image = image.copy()
    for peak in peaks:
        rho, theta = rho_values[peak[0]], theta_values[peak[1]]
        a = np.cos(theta)
        b = np.sin(theta)
        x0 = a * rho
        y0 = b * rho
        x1 = int(x0 + 1000 * (-b))
        y1 = int(y0 + 1000 * (a))
        x2 = int(x0 - 1000 * (-b))
        y2 = int(y0 - 1000 * (a))
        cv2.line(lines_image, (x1, y1), (x2, y2), (0, 0, 255), 1)

    return accumulator, peaks, lines_image
accumulator, peaks, lines_image = hough_transform(image)

cv2.imshow('Original Image', image)
cv2.imshow('Accumulator', cv2.normalize(accumulator, None, 0, 255,
cv2.NORM_MINMAX).astype(np.uint8))
cv2.imshow('Lines Image', lines_image)
cv2.imshow('Binary Edges', binary_edges)
cv2.waitKey(0)
cv2.destroyAllWindows()
```




```
plt.figure(figsize=(12, 12))
plt.subplot(1, 3, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')

plt.subplot(1, 3, 2)
plt.imshow(cv2.normalize(accumulator, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8),
cmap='gray')
```

```
plt.title('Accumulator')

plt.subplot(1, 3, 3)
plt.imshow(lines_image, cmap='gray')
plt.title('Lines Image')

plt.show()
```




```
def hough_circle_transform(image, min_radius, max_radius, delta_radius=1, threshold=10,
num_peaks=15):
    edges = binary_edges

    max_radius_index = int((max_radius - min_radius) / delta_radius) + 1
    accumulator = np.zeros((image.shape[0], image.shape[1], max_radius_index),
dtype=np.uint64)
    print("1")
    edge_points = np.column_stack(np.nonzero(edges))
    for center_x in range(image.shape[1]):
#         print("first")
        for center_y in range(image.shape[0]):
            for radius_index in range(max_radius_index):
                radius = min_radius + radius_index * delta_radius
```

```
                for point in edge_points:
                    x, y = point
                    if (x - center_x)**2 + (y - center_y)**2 == radius**2:
                        accumulator[center_y, center_x, radius_index] += 1
#     print("2")
    peaks = np.column_stack(np.unravel_index(np.argsort(accumulator.ravel())[-
num_peaks:], accumulator.shape))
#     print("3")
    circles_image = image.copy()
    for peak in peaks:
        center_x, center_y, radius_index = peak
        radius = min_radius + radius_index * delta_radius
        cv2.circle(circles_image, (center_x, center_y), radius, (0, 0, 255), 2)
#     print("4")
    return accumulator, peaks, circles_image
image = cv2.imread('shapes.jpg', cv2.IMREAD_GRAYSCALE)
min_radius = 2
max_radius = 20
delta_radius = 1

accumulator, peaks, circles_image = hough_circle_transform(image, min_radius,
max_radius, delta_radius)

cv2.imshow('Original Image', image)
cv2.imshow('Circles Image', circles_image)
cv2.waitKey(0)
cv2.destroyAllWindows()

plt.figure(figsize=(12, 12))
plt.subplot(1, 3, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')

acc = cv2.normalize(accumulator, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)
plt.subplot(1, 3, 2)
plt.imshow(acc[:, :, 0], cmap='gray')
plt.title('Accumulator')

plt.subplot(1, 3, 3)
plt.imshow(circles_image, cmap='gray')
plt.title('Lines Image')

plt.show()
```