Lab Report: 09

Title: Line and Circle Detection using Hough Transform Algorithm

Course title: Digital Image Processing Laboratory
Course code: CSE-406
4th Year 1st Semester Examination 2023

Date of Submission: 03/11/2024

Submitted to-
Dr. Md. Golam Moazzam
Professor
Department of Computer Science and Engineering
Jahangirnagar University
&
Dr. Morium Akter
Professor
Department of Computer Science and Engineering
Jahangirnagar University
Savar, Dhaka-1342

Class Roll	Exam Roll	Name
353	202165	Shanjida Alam

Department of Computer Science and Engineering
Jahangirnagar University
Savar, Dhaka, Bangladesh

Experiment No: 01

Experiment Name: Straight Line Detection using Hough Transform Algorithm

Objectives:

1. The objective of this lab is to demonstrate line detection in an image using the Hough Transform in Python with OpenCV.
2. Line detection is essential in computer vision applications for identifying boundaries, lane detection in self-driving cars, and various other applications where linear structures need to be identified.
3. This lab includes preprocessing the image, applying edge detection, detecting lines with the Hough Transform, and visualizing the results.

Code-01: Python

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image = cv2.imread('road.jpg')
inputImage = cv2.imread('road.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray, 50, 150, apertureSize=3)
lines = cv2.HoughLines(edges, 1, np.pi / 180, 200)
if lines is not None:
    for line in lines:
        rho, theta = line[0]
        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(image, (x1, y1), (x2, y2), (0, 0, 255), 2)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title("Detected Lines")
plt.axis('off')
plt.show()
```

Output:

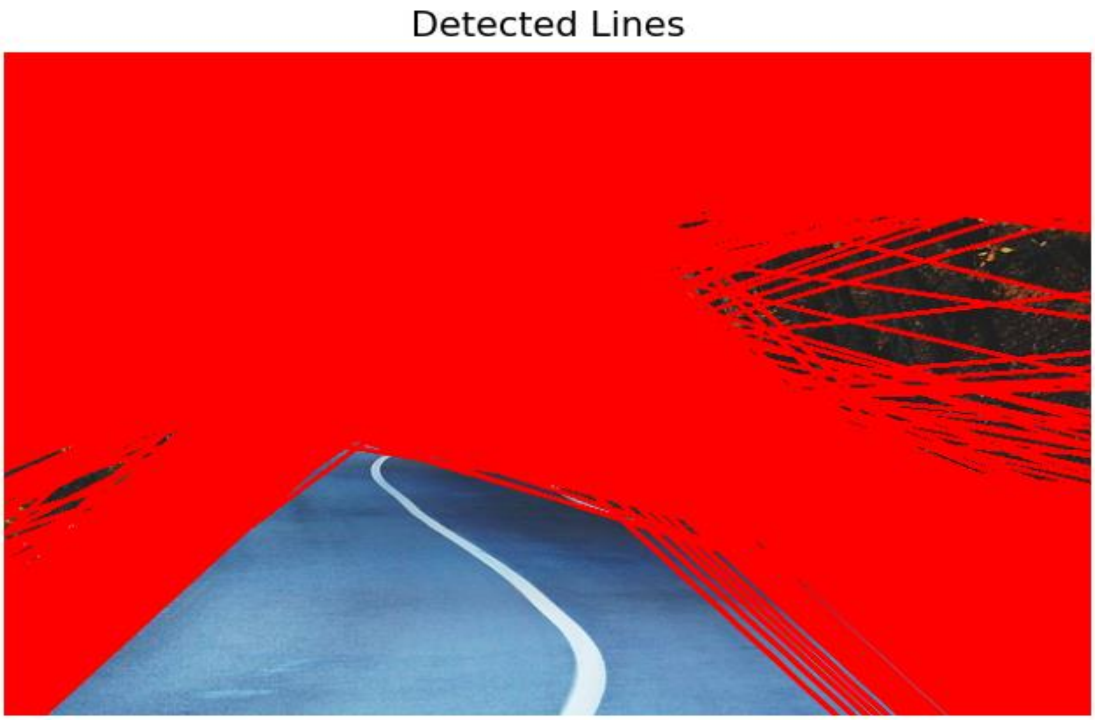

Figure 1.1: Line detection by Hough transform in Python

Explanation:

1. Import libraries.
2. `cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)` converts the image from color (BGR) to grayscale, which is a typical preprocessing step for edge detection and reduces computational load.
3. `cv2.Canny(gray, 50, 150, apertureSize=3)` applies Canny edge detection to the grayscale image. Canny edge detection detects edges in the image based on intensity gradients and returns a binary image with detected edges highlighted.
4. `cv2.HoughLines(edges, 1, np.pi / 180, 200)` is used to detect lines in the edge-detected image.
5. `cv2.line(image, (x1, y1), (x2, y2), (0, 0, 255), 2)` draws the detected line in red ((0, 0, 255)) with a thickness of 2.
6. `plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))` is used to display the image with detected lines in the red color. cv2.cvtColor converts the image from BGR to RGB for correct color display with Matplotlib.

Code-02: MATLAB

```matlab
image = imread('flower.jpeg');
gray = rgb2gray(image);
edges = edge(gray, 'Canny');
[H, T, R] = hough(edges);
P = houghpeaks(H, 5, 'threshold', ceil(0.3 * max(H(:))));
lines = houghlines(edges, T, R, P, 'FillGap', 20, 'MinLength', 40);
figure, imshow(image), hold on
for k = 1:length(lines)
    xy = [lines(k).point1; lines(k).point2];
    plot(xy(:, 1), xy(:, 2), 'LineWidth', 2, 'Color', 'red');
    plot(xy(1, 1), xy(1, 2), 'x', 'LineWidth', 2, 'Color', 'yellow');
    plot(xy(2, 1), xy(2, 2), 'x', 'LineWidth', 2, 'Color', 'green');
end
title('Detected Lines using Hough Transform');
hold off;
```

Output:

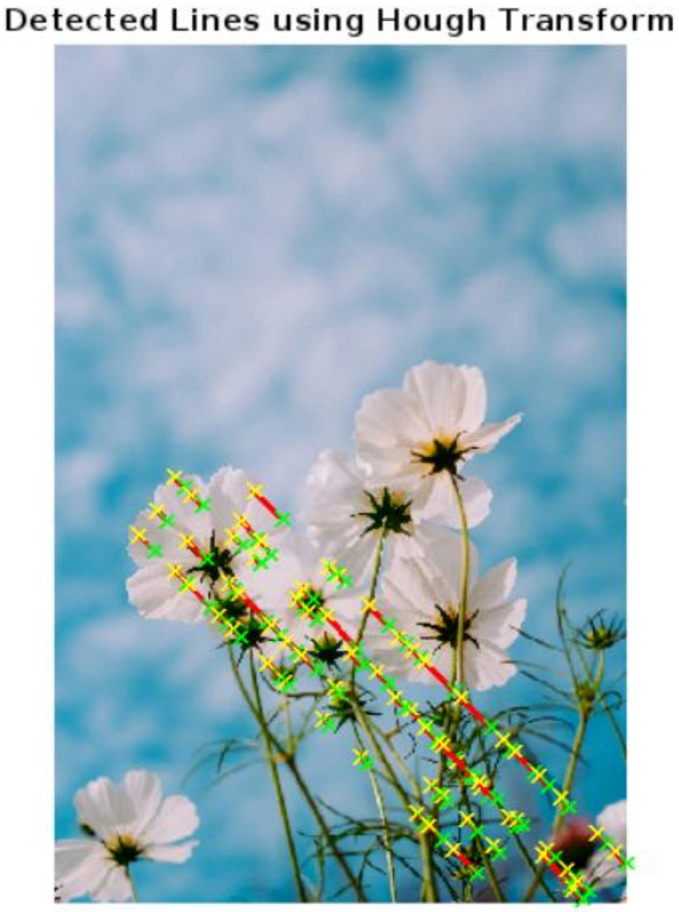

Figure 1.2: Line detection by Hough transform in MTLAB

Explanation:

1. The `edge()` function with the `'Canny'` method detects edges in the grayscale image. Canny edge detection is a popular method for finding edges in an image by identifying areas with strong intensity gradients.
2. The `hough()` function applies the Hough Transform to the edge-detected image, producing a Hough matrix `H`, and vectors `T (theta)` and `R (rho)`.
3. `houghpeaks()` identifies the most significant peaks in the Hough matrix `H`, which represent potential lines.
4. `FillGap` specifies the maximum gap between line segments to connect them into a single line, while `MinLength` sets the minimum length for a line to be considered.

Experiment No: 02

Experiment Name: Circle Detection using Hough Transform

Objectives:

1. The purpose of this lab is to demonstrate circle detection in an image using the Hough Transform with OpenCV in Python.
2. Detecting circular shapes is essential in various applications, including object tracking, detecting traffic signs, and analyzing medical images.
3. In this lab, we preprocess an image, apply a circular Hough Transform, and visualize the detected circles on the image.

Code-01: Python

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image = cv2.imread('road.jpg')
inputImage = cv2.imread('road.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
gray = cv2.GaussianBlur(gray, (9, 9), 2)
circles = cv2.HoughCircles(
   gray,
   cv2.HOUGH_GRADIENT,
   dp=1.2,
   minDist=30,
   param1=50,
   param2=30,
   minRadius=10,
   maxRadius=80
)
if circles is not None:
    circles = np.uint16(np.around(circles))
    for i in circles[0, :]:
        center = (i[0], i[1])
        radius = i[2]
        cv2.circle(image, center, radius, (0, 255, 0), 3)
        cv2.circle(image, center, 3, (0, 0, 255), 3)
plt.figure(figsize=(10, 10))
plt.subplot(2, 2, 1)
plt.imshow(cv2.cvtColor(inputImage, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
plt.axis('off')
plt.subplot(2, 2, 2)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title("Detected Circles")
plt.axis('off')
plt.show()
```

Output:

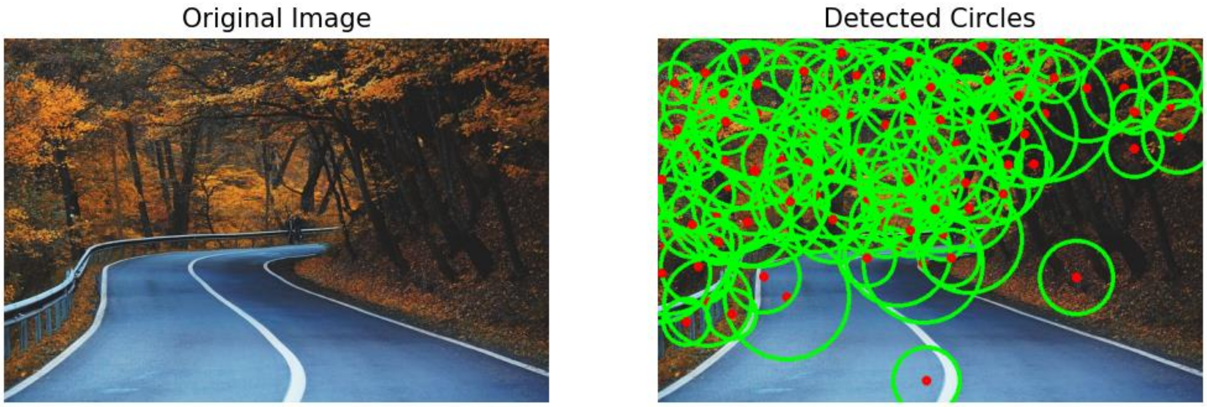

Figure 2.1: Circle Detection by Hough Transform in python

Explanation:

1. The code imports cv2 for image processing, `numpy` for numerical operations, and `matplotlib.pyplot` for displaying images.
2. The image is converted to grayscale using `cv2.cvtColor()`, a standard preprocessing step to reduce the computational load by eliminating color information.

3. A Gaussian blur with a `kernel size of (9, 9)` and standard deviation 2 is applied to smooth the image. This step helps reduce noise, making circle detection more accurate by blurring out minor image details.
4. The `cv2.HoughCircles()` function applies the Hough Circle Transform to detect circular shapes.
5. If any circles are detected, their properties are rounded and converted to integers using `np.uint16(np.around(circles)).`

Code-02: MATLAB

```
image = imread('flower.jpeg');
gray = rgb2gray(image);
[centers, radii, metric] = imfindcircles(gray, [20 80], 'Sensitivity', 0.9,
'EdgeThreshold', 0.1);
imshow(image);
hold on;
viscircles(centers, radii, 'EdgeColor', 'r');
title('Detected Circles using Hough Transform');
hold off;
```

Output:

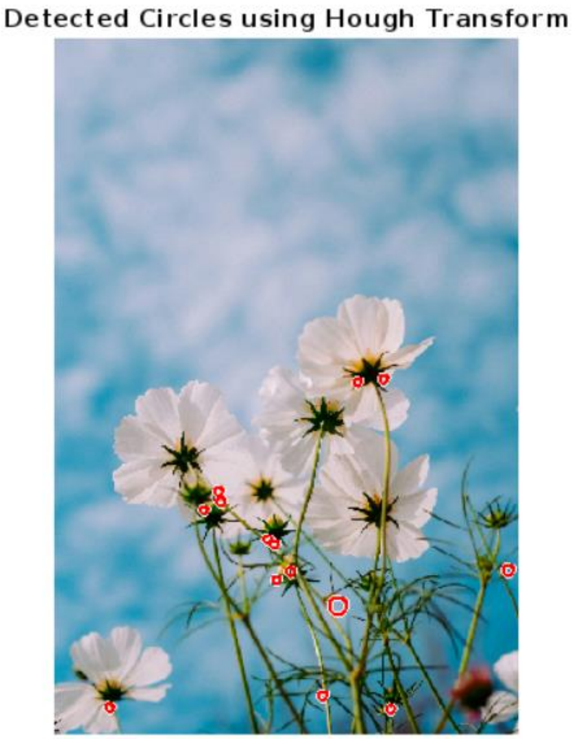

Figure 2.2: Circle Detection by Hough Transform in MATLAB

Explanation:

1. The `imfindcircles` function uses the Hough Circle Transform to detect circles in the grayscale image.
2. `[20 80] that` specifies the minimum and maximum radius range of circles to detect (between 20 and 80 pixels).
3. `'Sensitivity', 0.9` that sets the sensitivity level for detection. Higher sensitivity (close to 1) allows the detection of fainter circles but may increase false positives.
4. `'EdgeThreshold', 0.1` that determines the edge threshold used to detect circles. Lower values allow detecting weaker edges.
5. `imshow(image)` displays the original image, and `hold on` ensures that additional plots (e.g., circles) can be overlaid on this image.
6. `viscircles` overlays the detected circles on the image.