Rohaan Advani - N19682109

PROJECT 3 - REPORT .

P2. Hough Transform for straight lines without edge orientation

INTRODUCTION:
The Hough Transform is a computer vision technique used for detecting geometric shapes, particularly lines and circles, within an image. It was initially developed for detecting lines in binary images, and later extended to detect other shapes.

The basic idea behind the Hough Transform is to represent the geometric shapes in a parameter space rather than in the image space. This transformation makes it easier to identify shapes, especially when they may be obscured by noise or variations in the image.

Here's a simplified explanation of how the Hough Transform works for detecting lines:

1. Parameterization of Lines: A line in Cartesian coordinates is represented by the equation y = mx + b, where m is the slope and b is the y-intercept. However, this equation can be problematic when dealing with vertical lines (infinite slope). To overcome this, the line equation is often represented in a polar coordinate system as p = xcos(theta) + y sin(theta), where p is the distance from the origin to the closest point on the line, and theta is the angle between the x-axis and the line.

2. Accumulator Array: The Hough Transform uses an accumulator array to represent the parameter space. For line detection, the array is typically a 2D grid where one axis represents the slope m and the other axis represents the y-intercept b.

3. Voting: For each edge point in the binary image, the corresponding parameters m and b of possible lines passing through that point are calculated and the corresponding cells in the accumulator array are incremented.

4. Finding Peaks: After the voting process, cells in the accumulator array with high values indicate parameters that are likely to represent lines in the image. Peaks in the accumulator array correspond to potential lines in the image.

5. Thresholding: To reduce false positives, a threshold is often applied to the accumulator array. Only cells with values above the threshold are considered as detected lines.

6. Parameter Reconstruction: The detected peaks in the accumulator array can be used to reconstruct the parameters m and b of the lines in the original image.

Each point in image space generates a cos-curve which is accumulated in parameter space. The increment can be chosen as 1 for each point.

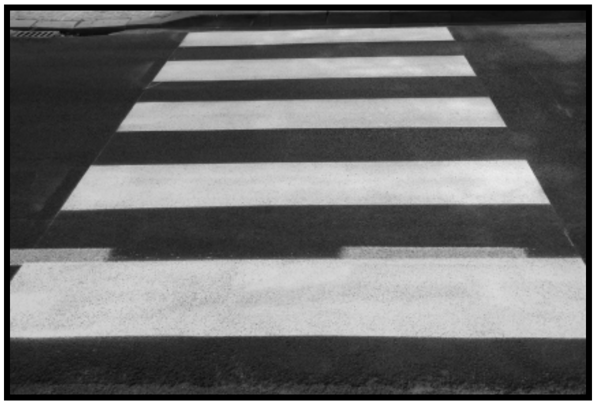
The Hough Transform can be extended to detect other shapes, such as circles, by modifying the parameterization and accumulator array accordingly. Overall, it's a powerful tool for detecting geometric shapes in images and is widely used in computer vision applications like image processing and feature extraction.

METHODOLOGY AND DISCUSSION:

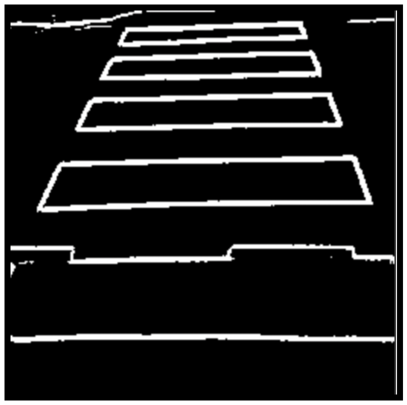
Algorithm for Hough Transform:

```
Initialize accumulator H to all zeros.
For each feature point (x,y) in the image.
        For $\theta = 0$ to 180
                $\rho = x \cos \theta + y \sin \theta$
                $H(\theta, \rho) = H(\theta, \rho) + 1$
Find the value(s) of $(\theta, \rho)$ where $H(\theta, \rho)$ is a local maximum.
The detected line in the image is given by $\rho = x \cos \theta + y \sin \theta$.
```

The above algorithm does not take edge orientation into consideration thus only straight edges will be detected. The problem statement states implementing hough transform without edge orientation information thus we will go ahead with the above algorithm as is. We will apply the algorithm to the below image:



In order to find the hough transform for the above image the first step requires us to grayscale and find the edges in the above image. Edge detection algorithm was implemented by us in Project 2. Following is the edge detection output created by the algorithm I implemented in project 2.

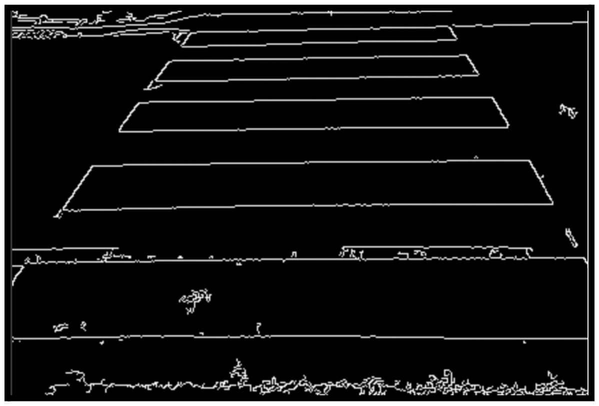


Since edge detection is not in the scope of this problem as an alternative we can use the inbuilt canny edge detection function in the cv2 library to create the edge image from the grayscale input image.

Syntax: cv2.Canny(image, T_lower, T_upper, aperture_size, L2Gradient)

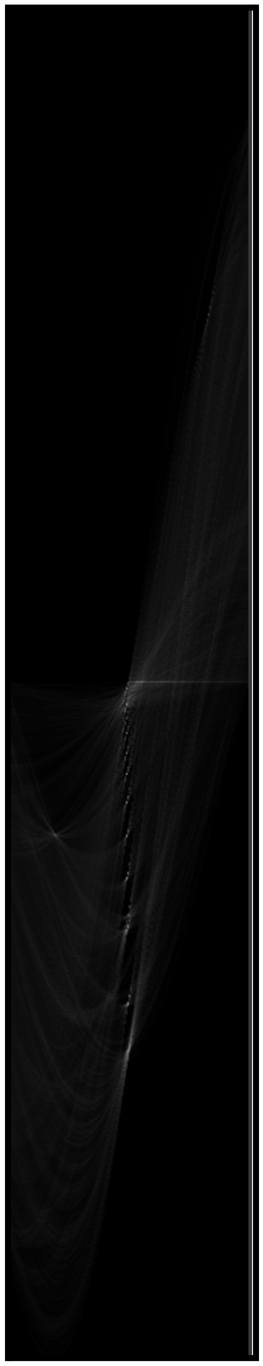
Image: Input image to which Canny filter will be applied
T_lower: Lower threshold value in Hysteresis Thresholding
T_upper: Upper threshold value in Hysteresis Thresholding
aperture_size: Aperture size of the Sobel filter.
L2Gradient: Boolean parameter used for more precision in calculating Edge Gradient.

For the purpose of this experiment, T_lower and T_upper have been set to 50 and 150 respectively. After canny edge detection, below is the edge image created:

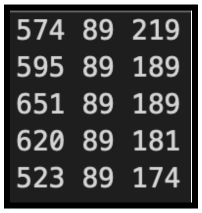


This image is now ready to be passed to the hough transform function. Here, the initially empty accumulator array must be created of size 2*(max_rho) X 180. Here max_rho is square root of height^2 plus width^2 where height and width are that of the image. The maximum value for theta is 180 degrees. As requested in the question, the default values for delta_rho and delta_theta can be 1 pixel step and 1 degree respectively which can be set in the function definition as shown in the code in the appendix. If delta_rho and delta_theta are higher, we get a coarser, less precise accumulator grid but higher processing time for the algorithm.

The next step of the hough transform is to fill the accumulator array as explained in the above algorithm. After filling the accumulator array for the above input image following accumulator image is obtained:

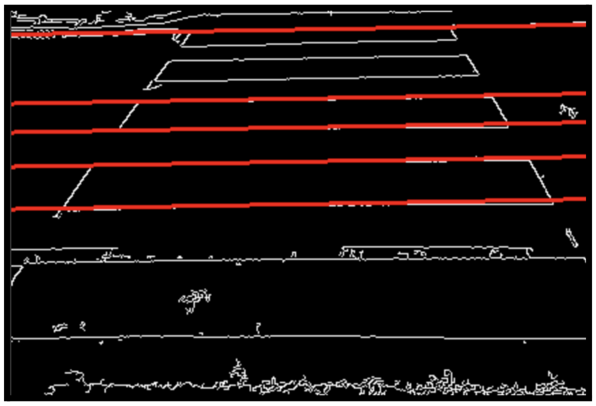


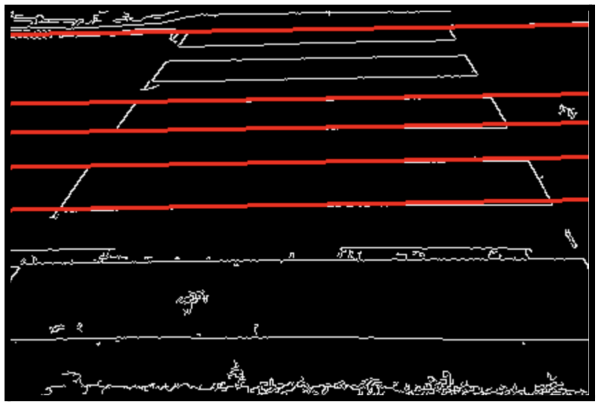
The cosine curves can be easily observed in the above accumulator array. Clusters in the parameter space indicate a higher number of votes. This accumulator array must now be used to find the peaks with the highest number of votes. We can also set a minimum threshold value below which the peaks are discarded. This can be the same value as T_lower in the canny edge method. Further the peaks list can be sorted and limited to the number of lines the user wishes to show overlay of the edges images. This ensures that only the N largest maxima are in consideration. The rho, theta and votes values of these sorted peaks can now be written into a file as requested. Following is the list of the same for the above input image:

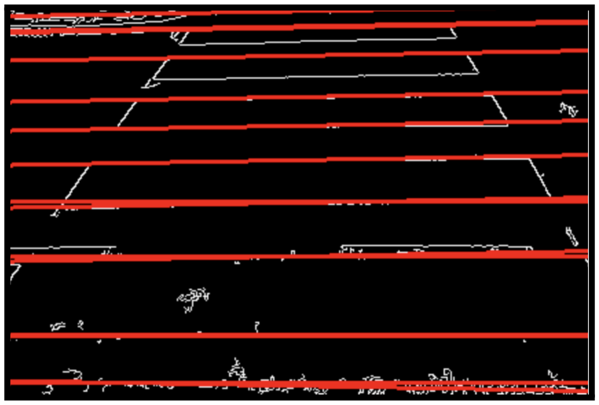


```
574 89 219
595 89 189
651 89 189
620 89 181
523 89 174
```

As the last step of the hough transform function, the sorted peaks list is used to draw lines on the edges images for the edges that create the maximum peaks in the sorted peaks list. Please refer to the appendix for the implementation for the same. Following is the resulting image after drawing the lines on the edges image.

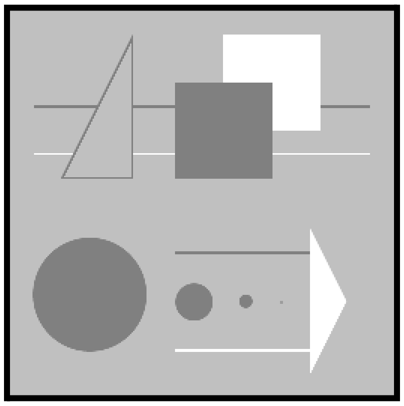

rho = 1, theta = 1

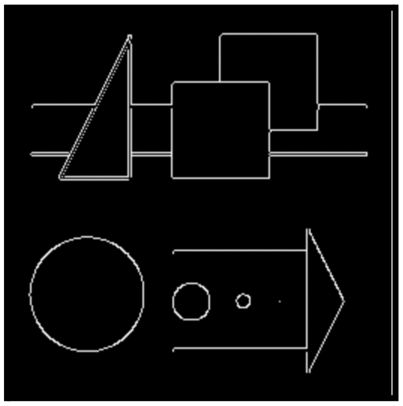
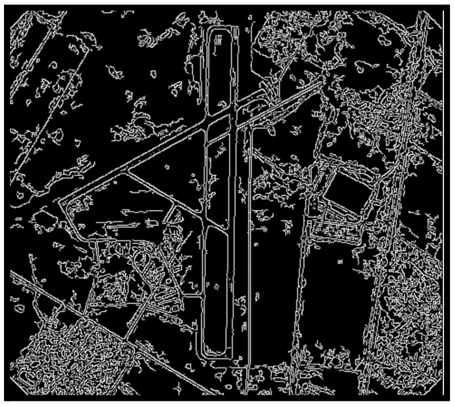

rho = 10, theta = 10


num_lines = 15

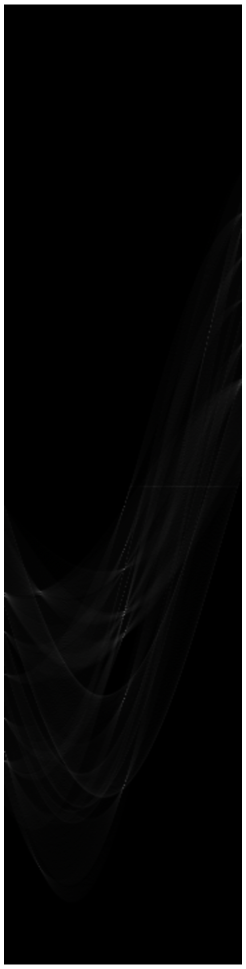
In the above image it can be noticed that multiple lines form over the same edge, this maybe because more peaks find discontinuities in the edges.

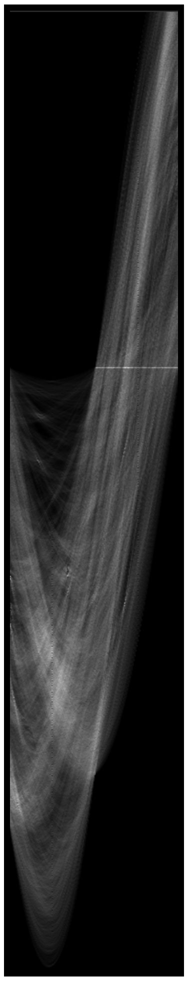
The same process can also be applied to other images. Shown below are input images, edge images, accumulator array images and resultant images from the hough transform function.

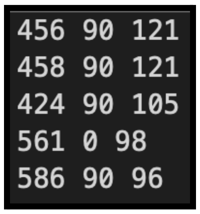


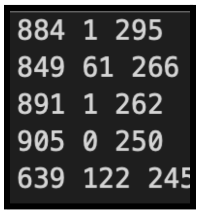
Rohaan Advani - N19682109



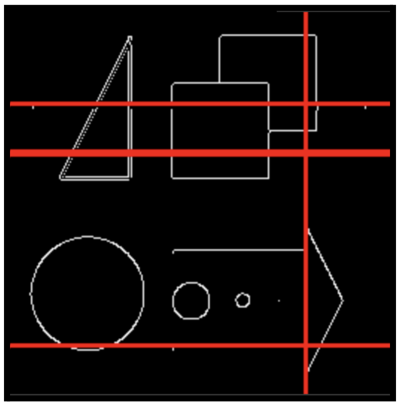


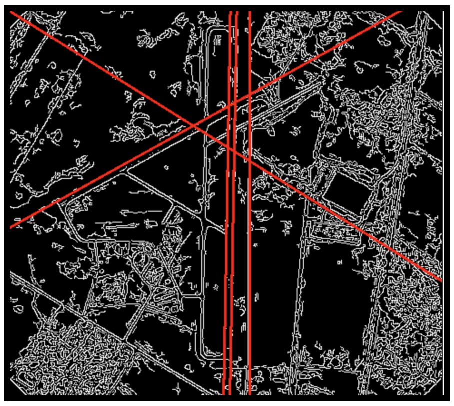


```
456 90 121
458 90 121
424 90 105
561 0 98
586 90 96
```



```
884 1 295
849 61 266
891 1 262
905 0 250
639 122 245
```





From the above images, it can be observed that the shape image has an almost empty accumulator array while the airport image has a relatively filled accumulator array. This is due to the high number of edges in the airport image. The shape image result is incorrect. This may be because the algorithm is not taking edge orientations into consideration.

P3. Hough Transform for circles of specific size

INTRODUCTION:
The Hough Transform for circles is specifically designed to identify circles within an image, even if they are partially obscured, noisy, or incomplete. Here's an overview of how the Hough Transform for circles works:

1. Edge Detection: Before applying the Hough Transform, it's common to perform edge detection on the image using techniques like the Canny edge detector. This helps in highlighting the prominent edges in the image.

2. Parameter Space: The Hough Transform works by representing each point in the image in a parameter space that corresponds to the parameters of the shape being detected. For circles, the parameters typically include the center coordinates (x, y) and the radius (r). Each point in the image contributes to a set of circles in the parameter space.

3. Accumulator Array: An accumulator array is used to keep track of the frequency of parameter values. Each point in the parameter space accumulates votes based on the points in the image that could belong to a circle with the corresponding parameters.

4. Voting: For each edge pixel in the edge-detected image, the Hough Transform algorithm votes for possible circle parameters in the accumulator array. This involves considering all possible circles that could pass through a given edge pixel.

5. Thresholding: After the voting process, the accumulator array is examined, and a threshold is applied to identify the most significant circles. The peaks in the accumulator array correspond to potential circles in the image.

6. Circle Extraction: The peaks in the accumulator array represent potential circles in the image. The corresponding parameters (center coordinates and radius) of these circles can be extracted for further analysis or visualization.

7. Post-Processing: To refine the results, additional post-processing steps may be applied, such as non-maximum suppression to eliminate redundant circle detections.

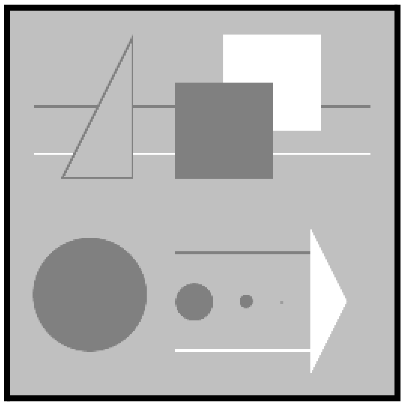
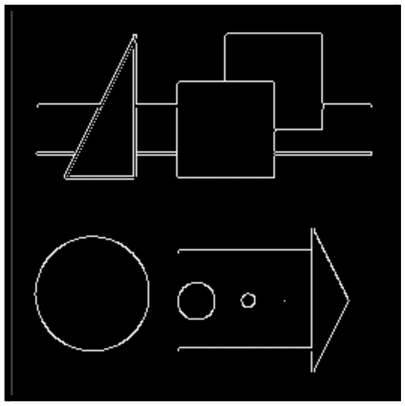
It's worth noting that the Hough Transform for circles is sensitive to parameters like the radius, and the algorithm may need to be applied multiple times with different radius values to detect circles of various sizes. Additionally, choosing an appropriate threshold is crucial for balancing sensitivity and specificity in circle detection.

METHODOLOGY AND DISCUSSION:

Algorithm for Hough Transform:

```
Initialize accumulator H to all zeros.
For each feature point (x,y) in the image.
        For radius in range min_range to max_range
                For theta in range 0 to 360 degrees
                        a = x - radius * cos(theta)
                        b = y - radius * sin(theta)
                        H(b, a, radius - min_radius) += 1
Find the value(s) of (b, a, radius) where H(b, a, radius) is a local maximum.
Draw the circle using center and radius information.
```

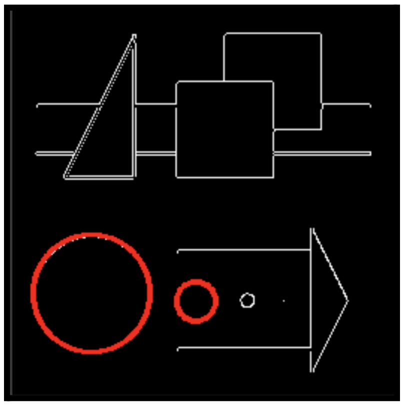
We will use the below image to detect circles in a specific radius range. Following is the corresponding edge image on the right.

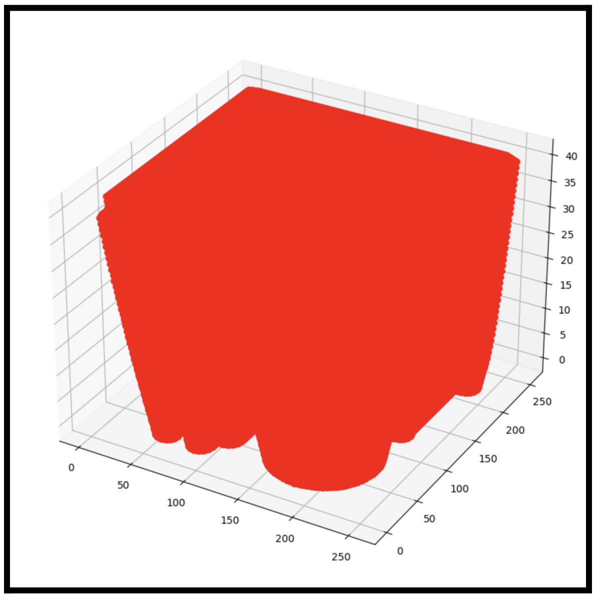
The edges image is now passed to the hough transform function. Here, the initially empty accumulator array must be created of size height, width, max_radius - min_radius + 1. Here height and width are that of the input image and max_radius and min_radius are user entered parameters. Circles within this radius bound will be detected in the parameter space. Delta radius can be set to 1 pixel size. In the next step the accumulator array will be filled by voting.

We can also set a minimum threshold value below which the peaks are discarded. This can be the same value as T_lower in the canny edge method. Further the peaks list can be sorted and limited to the number of lines the user wishes to show overlay of the edges images. This ensures that only the N largest maxima are in consideration.

As the last step of the hough transform function, the sorted peaks list is used to draw lines on the edges images for the edges that create the maximum peaks in the sorted peaks list. Please refer to the appendix for the implementation for the same. Following is the resulting image after drawing the lines on the edges image.



On creating the 3D plot for the above detected circles following is the plot we obtain:



Appendix: Code Implementation

```
import random
import string
import cv2
from google.colab.patches import cv2_imshow
import numpy as np

def random_string(K):
    return ''.join(random.choice(string.ascii_letters) for i in range(K))


def hough_transform(edges, delta_rho=1, delta_theta=np.pi/180, threshold=50, num_lines=5):
    height, width = edges.shape
    max_rho = int(np.sqrt(height**2 + width**2))
    accumulator = np.zeros((2*max_rho, 180), dtype=np.uint64)
    for y in range(height):
        for x in range(width):
            if edges[y, x] > 0:
                for theta in range(0, 180):
                    rho = int(x * np.cos(np.radians(theta)) + y * np.sin(np.radians(theta)))
                    accumulator[rho + max_rho, theta] += 1
    peaks = np.argwhere(accumulator > threshold)
    sorted_peaks = peaks[np.argsort(accumulator[peaks[:, 0], peaks[:, 1]])[::-1]][:num_lines]
    num_votes = []
    file1 = open(random_string(3)+'.txt', 'w+')
    for i in range(len(sorted_peaks)):
        rho2, theta2 = sorted_peaks[i][0], sorted_peaks[i][1]
        nv = accumulator[rho2][theta2]
        num_votes.append(nv)
        file1.write(str(rho2) + " ")
        file1.write(str(theta2) + " ")
        file1.write(str(nv) + "\n")
    result_image = cv2.cvtColor(edges, cv2.COLOR_GRAY2BGR)
    for peak in sorted_peaks:
        rho, theta = peak[0] - max_rho, peak[1]
        a = np.cos(np.radians(theta))
        b = np.sin(np.radians(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(result_image, (x1, y1), (x2, y2), (0, 0, 255), 2)
    return accumulator, result_image

image = cv2.imread('crosswalk.png', cv2.IMREAD_GRAYSCALE)
edges = cv2.Canny(image, 50, 150)
accumulator, result_image = hough_transform(edges)
cv2_imshow(image)
cv2_imshow(edges)
cv2_imshow(accumulator.astype(np.uint8))
cv2_imshow(result_image)

image = cv2.imread('edges-lines-orig.png', cv2.IMREAD_GRAYSCALE)
edges = cv2.Canny(image, 50, 150)
accumulator, result_image = hough_transform(edges)
cv2_imshow(image)
```

```python
cv2_imshow(edges)
cv2_imshow(accumulator.astype(np.uint8))
cv2_imshow(result_image)

image = cv2.imread('mnn4-runway-Ohio.jpg', cv2.IMREAD_GRAYSCALE)
edges = cv2.Canny(image, 50, 150)
accumulator, result_image = hough_transform(edges)
cv2_imshow(image)
cv2_imshow(edges)
cv2_imshow(accumulator.astype(np.uint8))
cv2_imshow(result_image)

def hough_transform_circles(edges, min_radius, max_radius, delta_radius=1, threshold=100, num_circles=5):
    height, width = edges.shape
    accumulator = np.zeros((height, width, max_radius - min_radius + 1), dtype=np.uint64)
    for y in range(height):
        for x in range(width):
            if edges[y, x] > 0:
                for radius in range(min_radius, max_radius + 1, delta_radius):
                    for theta in range(0, 360):
                        a = x - int(radius * np.cos(np.radians(theta)))
                        b = y - int(radius * np.sin(np.radians(theta)))
                        if 0 <= a < width and 0 <= b < height:
                            accumulator[b, a, radius - min_radius] += 1
    peaks = np.argwhere(accumulator > threshold)
    sorted_peaks = peaks[np.argsort(accumulator[peaks[:, 0], peaks[:, 1], peaks[:, 2]])[::-1]][:num_circles]
    result_image = cv2.cvtColor(edges, cv2.COLOR_GRAY2BGR)
    for peak in sorted_peaks:
        x, y, radius = peak[1], peak[0], peak[2] + min_radius
        cv2.circle(result_image, (x, y), radius, (0, 0, 255), 2)
    return accumulator, result_image

image = cv2.imread('edges-lines-orig.png', cv2.IMREAD_GRAYSCALE)
min_radius = 10
max_radius = 50
delta_radius = 1
threshold = 100
num_circles = 5
edges = cv2.Canny(image, 50, 150)
accumulator, result_image = hough_transform_circles(edges, min_radius, max_radius, delta_radius, threshold,
num_circles)
cv2_imshow(image)
cv2_imshow(edges)
cv2_imshow(result_image)

def plot_accumulator_3d(accumulator):
    fig = plt.figure(figsize=(16,10))
    ax = fig.add_subplot(111, projection='3d')
    z, y, x = accumulator.nonzero()
    ax.scatter(z, y, x, c='r', marker='o')
plt.show()
plot_accumulator_3d(accumulator)
```