Digital Image Processing Lab Report 6
CSE 4734

Abdullah Al Jubaer Gem
ID: 210041226
Section: 2B

February 21, 2026

1 Task 1: Hough Transform for Line Detection

1.1 Problem Statement

Apply the Hough Transform (HT) on a color image for detecting straight lines. Write observations by changing the threshold on the accumulation votes.

1.2 Theory and Approach

The Hough Transform detects lines by mapping each edge pixel into a (ρ, θ) parameter space, where ρ is the perpendicular distance from the origin to a line and θ is its orientation. Edge pixels that are collinear accumulate votes at the same (ρ, θ) bin, forming a peak that indicates a detected line. The image is first converted to grayscale and Canny edge detection is applied to produce clean edge pixels. These feed into `cv2.HoughLines`, which internally manages the accumulator and returns all lines whose vote count exceeds the given threshold.

1.3 Implementation

```
# Resize image
img = cv2.resize(img, (390, 270), interpolation = cv2.INTER_LINEAR)

# Convert the img to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Apply edge detection method on the image
edges = cv2.Canny(gray, 100, 150, apertureSize=3)

# This returns an array of r and theta values
lines = cv2.HoughLines(edges, 1, np.pi/180, 175)
```

1.4 Output

Figure 1: Canny Edges

Figure 2: Detected Lines Visualization

1.5 Observations

The Canny output shows clean, well-localised edge pixels along the dominant structural boundaries of the image, which is essential for reliable voting. In the detected lines result, the HT correctly identifies the main straight edges and draws them as full extended lines. The threshold parameter has a direct impact on which lines survive:

- **Lower threshold (e.g., 100–150):** More lines are detected, including weaker or shorter edges and some noise-induced artefacts. This leads to over-detection with many redundant or spurious lines.
- **Higher threshold (e.g., 200+):** Only lines supported by a large number of collinear edge points survive. Detection becomes more selective, but genuinely present lines with partial occlusion or short extent may be missed.
- **Threshold = 175:** A reasonable balance for this image — the dominant structural lines are captured without excessive false positives.

It is also noted that the HT is inherently global: even a partially occluded line accumulates enough votes to be detected, making it robust to gaps in edges.

2 Task 2: Generate the Hough/Accumulator Space

2.1 Problem Statement

Manually generate and display the Hough accumulator space using (ρ, θ) parameters.

2.2 Theory and Approach

This task manually builds the (ρ, θ) accumulator to make the voting mechanism explicit. For each edge pixel, ρ is computed for every discrete angle θ and the corresponding bin in a 2D array is incremented. The accumulator is sized to cover all possible ρ values from $-d$ to $+d$ where d is the image diagonal. The resulting array is visualised as a heatmap — bright regions indicate (ρ, θ) combinations that received many votes, corresponding to dominant lines in the image.

2.3 Implementation

```
height, width = edges.shape
diag_len = int(np.sqrt(height**2 + width**2))
rhos = np.arange(-diag_len, diag_len + 1, 1)
thetas = np.deg2rad(np.arange(-90, 90, 1))
accumulator = np.zeros((len(rhos), len(thetas)), dtype=np.uint64)
```

```
ys, xs = np.nonzero(edges)

for i in range(len(xs)):
    x = xs[i]
    y = ys[i]
    for t_idx in range(len(thetas)):
        theta = thetas[t_idx]
        rho = int(round(x * np.cos(theta) + y * np.sin(theta)))
        rho_idx = rho + diag_len
        accumulator[rho_idx, t_idx] += 1
```

2.4 Output

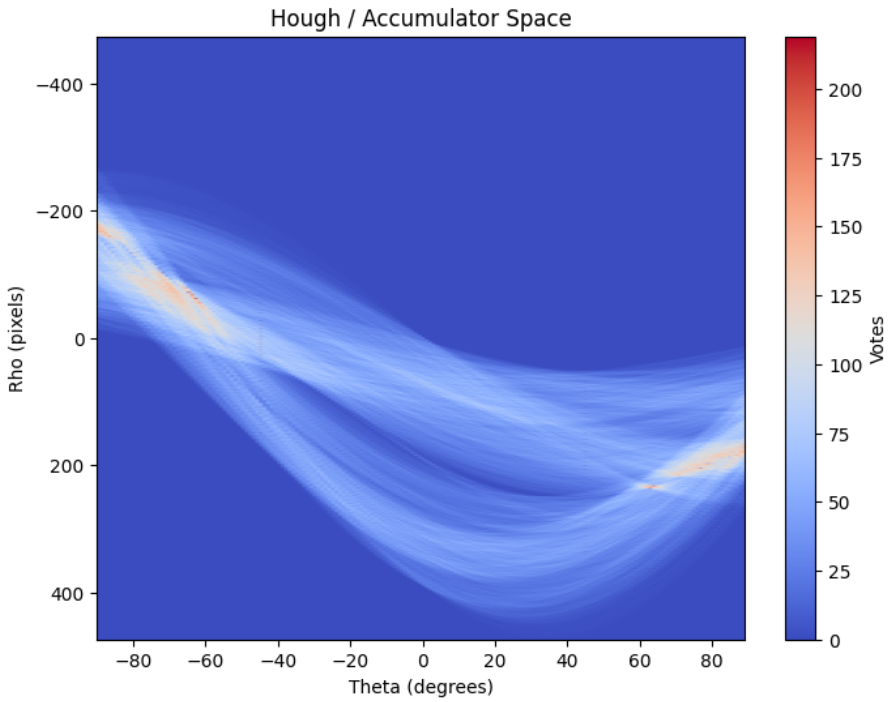

Figure 3: Hough Accumulator Space (ρ–θ parameter space)

2.5 Observations

The accumulator visualisation reveals the structure of the voting process clearly. Each edge point contributes a sinusoidal trace across the (ρ, θ) plane. Where many sinusoids intersect — visible as bright hotspots in the heatmap — a dominant line exists in the image. The concentrated bright peaks correspond precisely to the straight lines detected in Task 1. The background shows a low, diffuse level of votes contributed by edge pixels that belong to curved or noisy regions, none of which accumulate enough support to form a clear peak. The 1° angular resolution used here provides fine discrimination between lines at slightly different orientations, though it also makes the computation significantly slower compared to coarser resolutions.

3 Task 3: Detected Straight Lines via Accumulator Thresholding

3.1 Problem Statement

Apply a threshold to the manually built accumulator, list the detected line parameters, and draw the corresponding lines on the input image.

3.2 Theory and Approach

With the accumulator built, line detection is a matter of thresholding — any bin whose vote count exceeds a set fraction of the global maximum is treated as a detected line. A relative threshold (here 80% of the peak) keeps the criterion adaptive to image content. Each surviving (ρ, θ) bin is converted back to a pair of image-space endpoints and drawn on the original image.

3.3 Implementation

```
threshold = 0.8 * np.max(accumulator)

detected_lines = []
for rho_idx in range(accumulator.shape[0]):
    for theta_idx in range(accumulator.shape[1]):
        if accumulator[rho_idx, theta_idx] >= threshold:
            rho = rhos[rho_idx]
            theta = thetas[theta_idx]
            detected_lines.append((rho, theta))

# Draw lines
for rho, theta in detected_lines:
    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(img_lines, (x1, y1), (x2, y2), (0, 0, 255), 2)
```

3.4 Output

Figure 4: Detected Lines with 80% Threshold on Accumulator

3.5 Observations

Using 80% of the peak accumulator value as the threshold retains only the lines with the strongest collinear edge support in the image. The result closely matches the OpenCV HoughLines output from Task 1, confirming the correctness of the manual implementation. The lines are drawn as infinite extensions across the image rather than segments, which is a direct consequence of the parametric representation — the transform gives a line's orientation and offset but carries no information about where along the line edge points actually lie. Reducing the threshold (e.g., to 50%) would surface a larger number of weaker lines, many of which correspond to shorter or noisier edges that do not represent genuine dominant structure.

4 Task 4: Circle Detection using Hough Transform

4.1 Problem Statement

Implement circle detection using the Hough Transform for a fixed radius: detect edges, build the circle accumulator, display it, and draw detected circles at different threshold levels.

4.2 Theory and Approach

For circle detection with a fixed radius r, the Hough Transform uses a 2D accumulator over possible circle centres (a, b). Each edge pixel votes for all centres it could belong to by sweeping θ from 0 to 360 and incrementing the corresponding bin. Where many edge pixels from a circular arc agree on the same centre, a peak forms in the accumulator. Thresholding this accumulator extracts candidate centres, and circles of radius r are drawn around each one.

4.3 Implementation

```
# Edge Detection
edges = cv2.Canny(gray, 100, 150)

# Accumulator for fixed radius
radius = 30
accumulator = np.zeros((height, width), dtype=np.uint64)
ys, xs = np.nonzero(edges)
for i in range(len(xs)):
    x = xs[i]
    y = ys[i]
    for theta in range(0, 360, 5):
        t = np.deg2rad(theta)
        a = int(x - radius * np.cos(t))
        b = int(y - radius * np.sin(t))
        if 0 <= a < width and 0 <= b < height:
            accumulator[b, a] += 1

# Threshold and draw
for threshold in thresholds:
    centers = []
    for y in range(height):
        for x in range(width):
            if accumulator[y, x] > threshold:
                centers.append((x, y))
    img_copy = img.copy()
    for (x, y) in centers:
        cv2.circle(img_copy, (x, y), radius, (0, 0, 255), 2)
        cv2.circle(img_copy, (x, y), 2, (0, 255, 0), 3)
```

4.4 Output

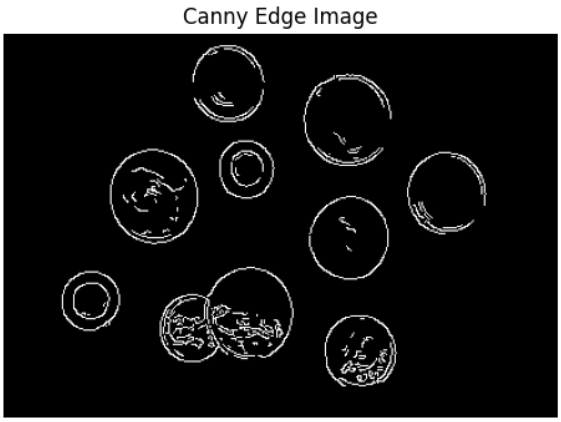

Figure 5: Canny Edges for Circle Detection

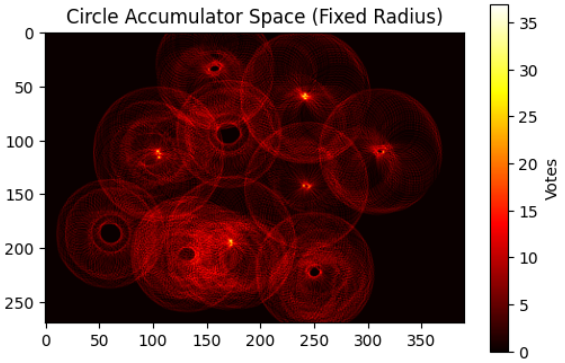

Figure 6: Circle Accumulator Space (Fixed Radius)

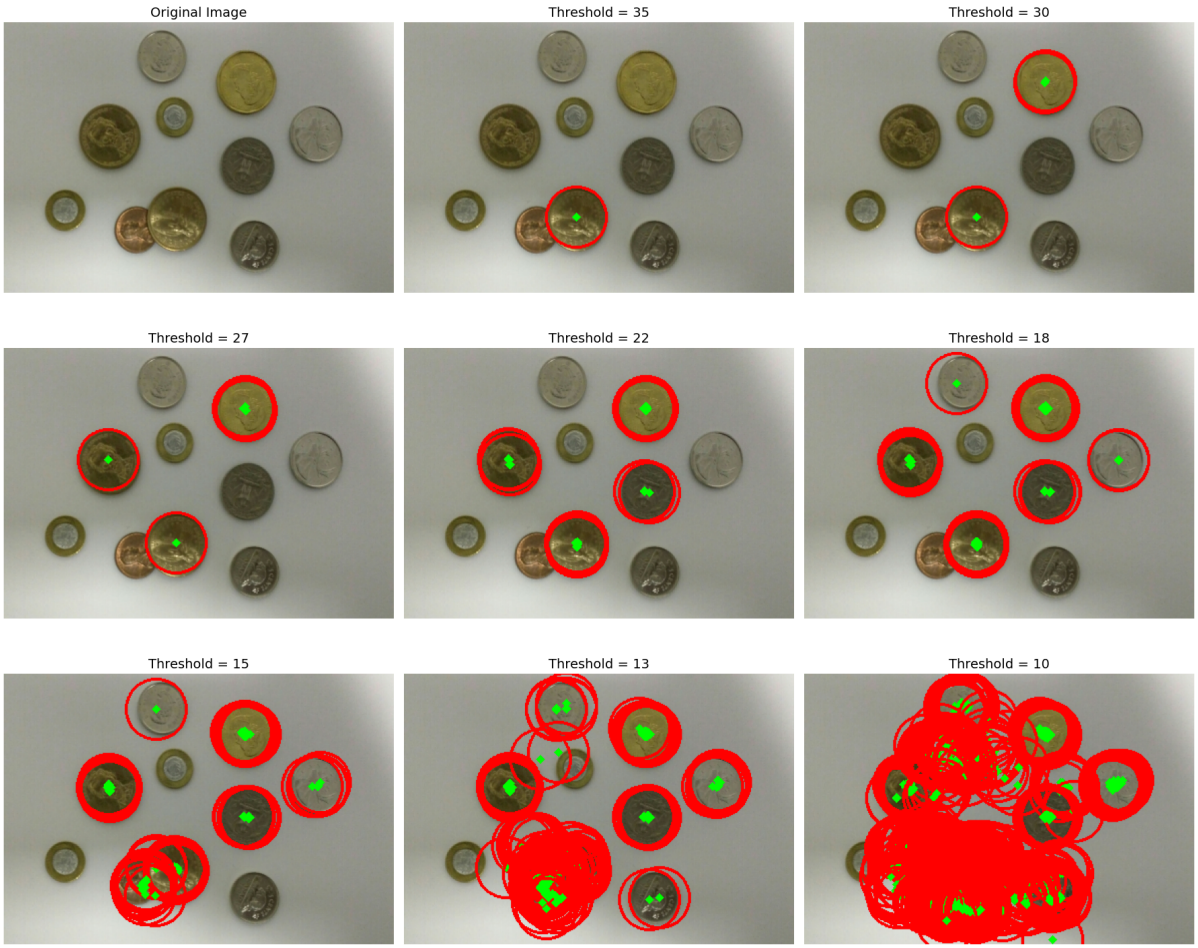

Figure 7: Circle Detections at Different Accumulator Thresholds

4.5 Observations

The circle accumulator visualisation shows bright peaks at locations where many edge pixels from the Canny output collectively vote for the same centre, confirming circular structure in the image. The effect of the threshold is clearly illustrated across the detection panels:

- **High thresholds (35, 30):** Only the most strongly supported centres survive, yielding clean detections aligned with the actual circles in the image with very few false positives.
- **Medium thresholds (27, 22, 18):** More centres are accepted, and detections begin to cluster around the true circle boundaries. Slightly noisy edge regions start contributing spurious centres nearby.
- **Low thresholds (15, 13, 10):** The number of accepted centres grows rapidly. Many spurious detections appear far from any real circle, as background noise in the accumulator also crosses the threshold.

The angular step size of 5° provides a reasonable trade-off between speed and voting coverage — a finer step (e.g., 1°) would yield more precise peaks but at a significant computational cost. A key limitation of the fixed-radius approach is that it will completely miss any circles whose actual radius differs from the assumed value; extending to variable radius requires a full 3D (ρ, θ, r) accumulator at much greater computational expense.