MSAI 495 Introduction to Computer Vision - Assignment 6
Sayantani Bhattacharya

Hough Transform for Line Detection

Algorithms Used:

I. For Canny Edge Detection: (explained in detail in MP5)
   A. Load training image and convert it to a grayscale floating-point image.
   B. Apply Gaussian smoothening.
   C. Compute gradient magnitude and directions(Sobel)
   D. For each pixel, check if its magnitude is greater than its neighbors along the gradient direction. If not, suppress it to zero.
   E. Flatten the magnitude array and sort it.
   F. Separate pixels into weak and strong edges, based on thresholding (T_low, T_high).
   G. Weak edges connected to strong edges are kept; others are discarded.

II. For Hough Transform:
   A. Non-zero pixels in the edge image from Canny detection are extracted as possible points on lines.
   B. Accumulator Matrix is initialised and calculated:
      1. θ (theta) is varied from 0° to 180° in theta_res steps.
      2. ρ (rho) is computed as: rho = x*cos(theta) + y*sin(theta) for each edge pixel (x, y).
      3. For each (ρ, θ), the corresponding cell in the accumulator matrix is incremented. If multiple edge pixels vote for the same (ρ, θ), the accumulator cell for that line will get a **high vote count** — indicating a strong likelihood of a line there. So, after all edge pixels have voted, we just look for peaks (cells with high values) in the accumulator.
      4. Each peak corresponds to a detected line in the image.
   C. Return the Hough accumulator matrix, along with the list of all possible ρ and θ values..

III. For Calculating peak lines [`get_peak_lines()`]:
   A. Iterate through the Hough accumulator matrix.
   B. If any (ρ, θ) combination has votes greater than a defined threshold (e.g., 50), it is considered a strong line.
   C. These (ρ, θ) pairs are stored as detected lines.
   D. The draw_lines() function uses each (ρ, θ) pair to draw corresponding lines on the original image using the polar to Cartesian conversion.

Results:

1. Input Test Images

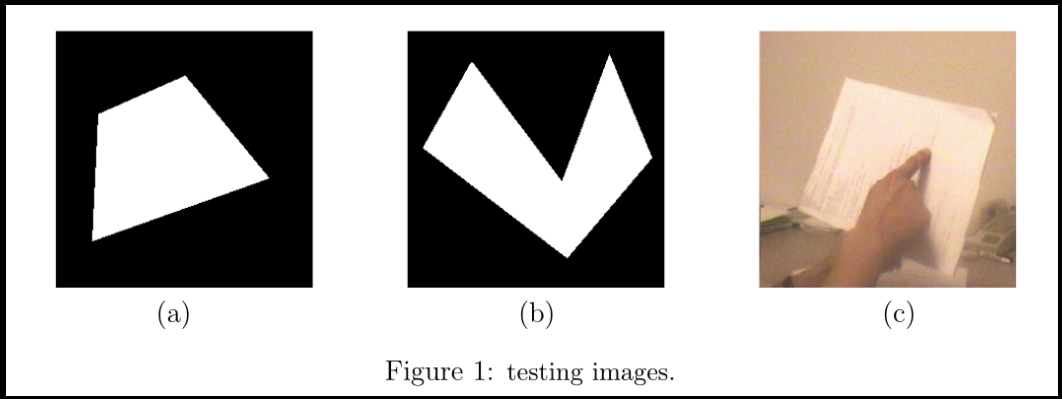
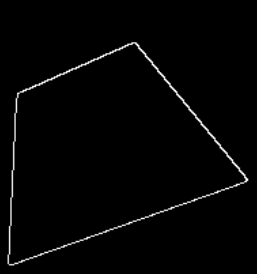
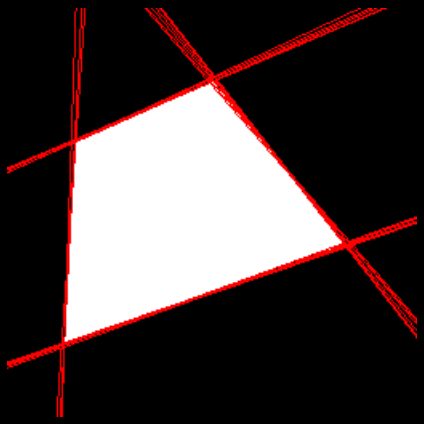
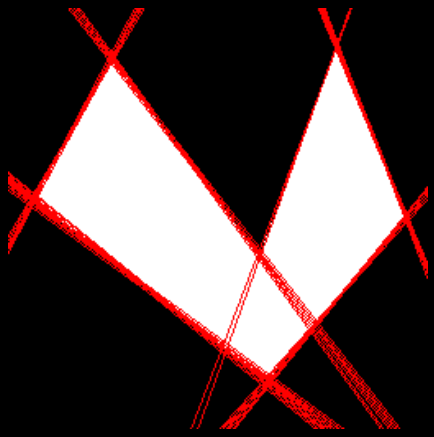
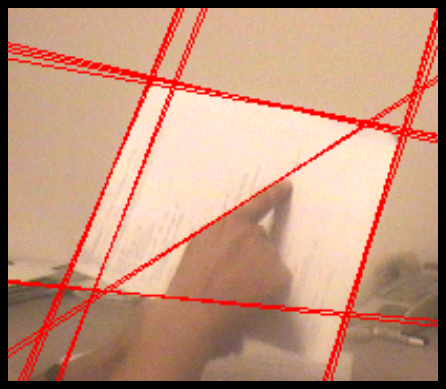


Figure 1: testing images.

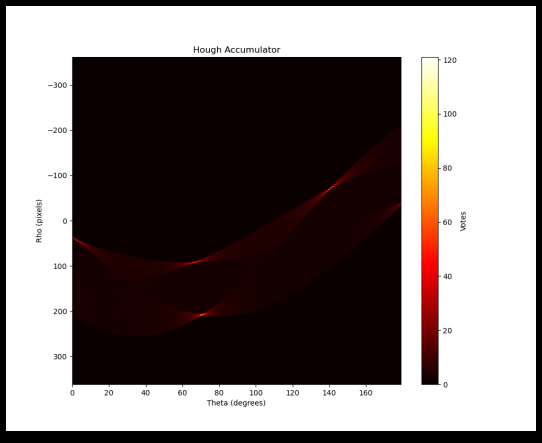
2. Edge detection results:

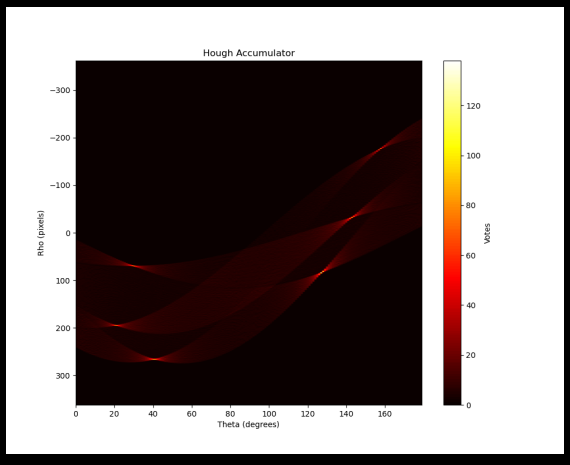


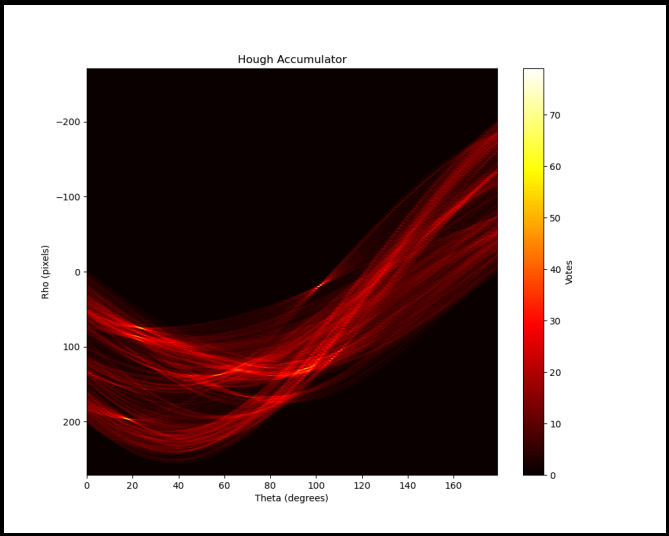
3. Line detection results with param: `rho_res=1, theta_res=1, threshold=50`

4. Accumulator (in same order) plots with param: `rho_res=1, theta_res=1, threshold=50`

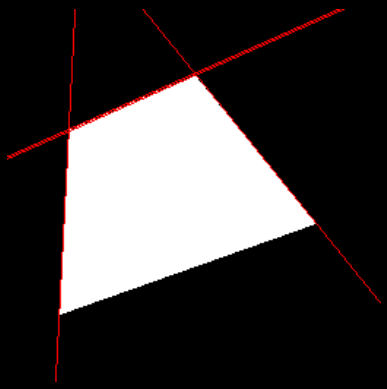


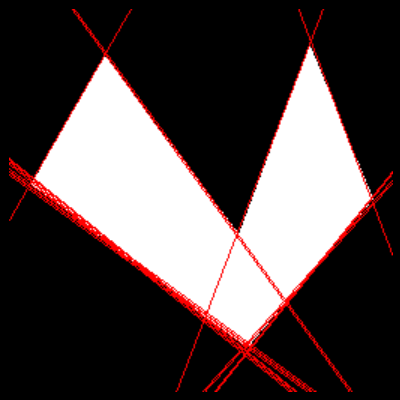


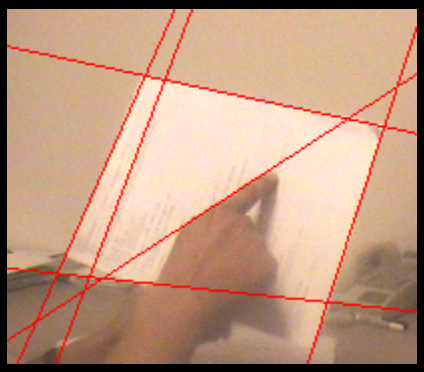


5. Accumulator (in same order) plots with param: `rho_res=1, theta_res=3, threshold=50.` Higher theta_res gives fewer but more confident lines, as votes get combined across a wide angle range.

```
hw6/input.bmp: saved edges and line-detected images. Lines found: 6
hw6/test.bmp: saved edges and line-detected images. Lines found: 4
hw6/test2.bmp: saved edges and line-detected images. Lines found: 17
```

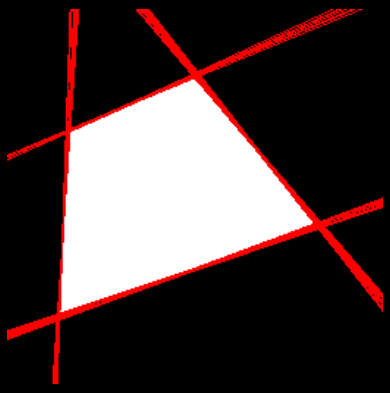
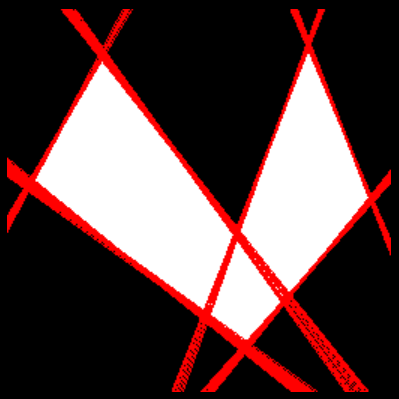
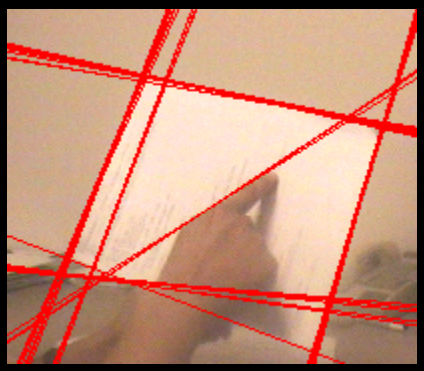






6. Accumulator (in same order) plots with param: rho_res=1, theta_res=0.4, threshold=50. Lower theta_res means more angular precision, thus more lines are detected, they may have less strength (fewer votes).

```
hw6/input.bmp: saved edges and line-detected images. Lines found: 39
hw6/test.bmp: saved edges and line-detected images. Lines found: 39
hw6/test2.bmp: saved edges and line-detected images. Lines found: 94
```

7. Line detection results with param: `rho_res=1, theta_res=1, threshold=80`: As expected fewer lines were detected,and the last case does not detect any lines in the threshold range of 80, which means no edge vote is strong enough to have 80.