Indian Institute of Information Technology Bhopal

Digital Image Processing

Branch - Information Technology (IT section-2)

Semester- **5th Semester**

Scholar number – **23U03122 & 23U03117**

Submitted to -:
Dr. Supriya Aggarwal

Submitted by -
Rounit Kashyap
& Shekhar Borile

Topic: Extracting Straight Lines from a Binary Image using Hough Transformation

Aim: To detect and extract straight lines from a binary (black and white) image using the Hough Line Transformation technique implemented in Python with NumPy.

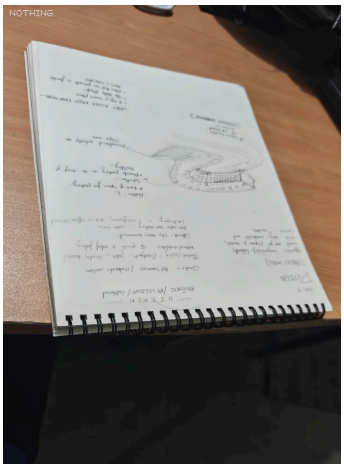
Objectives：

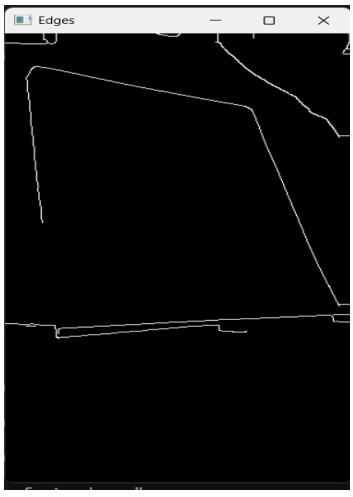
- To study the Hough Transform and its application in line detection.
- To implement a fast, vectorized Hough Line Transform using NumPy.
- To extract and visualize lines from edge-detected images.

Theory :

Edge Image

Original Image **Edge Image**





The edge **image** is the result of an edge detection algorithm, where the pixels corresponding to sharp intensity changes (i.e., edges) are marked. Edge detection is essential as it helps simplify the input before applying more complex algorithms like Hough Transform.

In image processing, edge detection algorithms such as **Canny**, **Sobel**, and **Laplacian** are used to compute these edges. For the **Hough Line Transform**, the input must ideally be a **binary edge image**, where white pixels indicate edges and black pixels represent the background.

Example:The Canny edge detection algorithm is commonly used due to its effectiveness in noise removal and strong edge detection.

Representing Lines in the Hough Space

The most basic representation of a straight line in Cartesian coordinates is:

$$y=mx+by = mx + by=mx+b$$

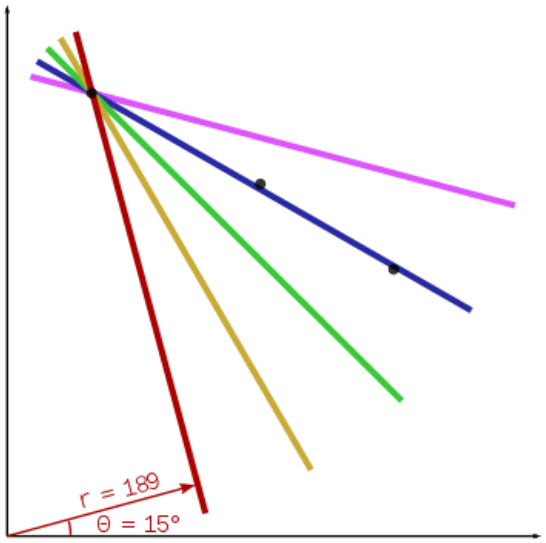
where **m** is the slope and **b** is the y-intercept.

However, this representation presents challenges:

- **Vertical lines** have infinite slope, leading to numerical instability.
- It becomes difficult to uniquely represent and detect all possible lines, especially in computer-based processing.

To overcome this, we use an alternative representation in **polar coordinates**:

$$\rho= x\cos(\theta)+y\sin(\theta)$$



where:

- ρ = perpendicular distance from the origin to the line
- θ = angle between the normal (perpendicular) to the line and the x-axis

This representation ensures:

- Every straight line can be described using a unique (ρ,θ) pair
- No infinite values → perfect for computational purposes

This representation forms the basis of the **Hough Transform**.

Hough Line Transform: How It Works

The **Hough Line Transform** is a feature extraction technique used to detect straight lines in an image. It works on a **voting-based system** in a parameter space rather than directly in the image space.

Steps Involved:

1. **Select Edge Points**
   Take each pixel that is part of an edge (usually obtained from an edge detector like Canny).

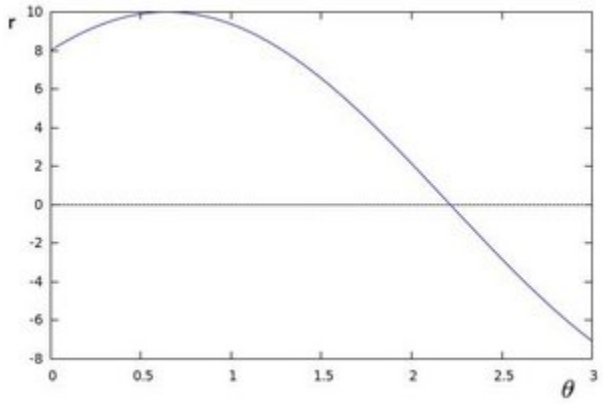
2. **Transform into Parameter Space**
   For each selected point (x,y), represent all possible lines passing through it using the normal form of a line:

   $$\rho = x \cdot \cos(\theta) + y \cdot \sin(\theta)$$

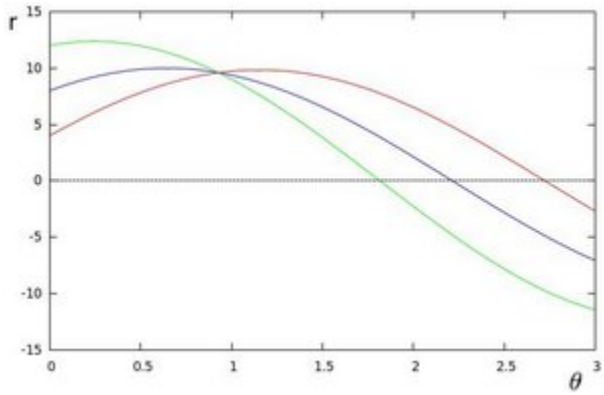
   where

   - ρ = perpendicular distance from origin
   - θ = angle of the normal line to the x-axis

3. **Plot in Hough Space**
   For each point, plot the resulting sinusoidal curves on a 2D graph of ρ\rhoρ vs θ\thetaθ. This space is known as the **Hough space**.



4. **Voting / Accumulation**



   All curves intersect at a point (ρ,θ) corresponding to a possible line in the image. The more intersections or "votes" at a point, the higher the likelihood of a line.

5. **Apply Threshold**
   Set a threshold on the number of votes. Points in the Hough space that exceed this threshold are considered to represent valid lines in the image.

6. **Extract Lines**
   The (ρ,θ) pairs corresponding to the peaks in the Hough space give the parameters of the detected lines.

Why It Works Well for Line Detection

- **Robust to Noise**: Even if the edge data is noisy, the voting-based nature can still extract the strongest line.
- **Detects Hidden Lines**: It can detect lines even if they are fragmented in the image but align globally.
- **Works Well on Binary Edge Images**: Perfect after Canny or Sobel.

Workflow :

1. Input a binary image (output of edge detection like Canny).
2. Compute ρ and θ values using vectorized calculations.
3. Accumulate votes in Hough space.
4. Apply a threshold to extract significant lines.

Methodology

1. Preprocess the image using edge detection (Canny).
2. Apply vectorized Hough Transform to accumulate votes.
3. Extract lines based on a voting threshold.
4. Plot or return detected (ρ,θ)(\rho, \theta)(ρ,θ) pairs for further visualization.

Code Implementation :

```
def getLines(img, threshold, angle_step=1):
    """Hough Line Transform using vectorized numpy operations"""
    thetas = np.deg2rad(np.arange(-90.0, 90.0, angle_step))
    width, height = img.shape
    diag_len = int(np.ceil(np.sqrt(width * width + height * height)))  # 
max_dist
    rhos = np.linspace(-diag_len, diag_len, diag_len * 2)

    # Precompute reusable values
    cos_theta = np.cos(thetas)
    sin_theta = np.sin(thetas)
    num_thetas = len(thetas)

    accumulator = np.zeros((2 * diag_len, num_thetas))
    y_idxs, x_idxs = np.nonzero(img)  # (row, col) indices for edge points

    # Vectorized rho computation
    xcosthetas = np.dot(x_idxs.reshape((-1,1)), cos_theta.reshape((1,-1)))
    ysinthetas = np.dot(y_idxs.reshape((-1,1)), sin_theta.reshape((1,-1)))
    rhosmat = np.round(xcosthetas + ysinthetas) + diag_len
    rhosmat = rhosmat.astype(np.int16)

    # Voting
    for i in range(num_thetas):
        _rhos, counts = np.unique(rhosmat[:, i], return_counts=True)
        accumulator[_rhos, i] = counts

    # Extract lines above threshold
    idxs = np.argwhere(accumulator > threshold)
    rho_idxs, theta_idxs = idxs[:,0], idxs[:,1]

    return np.column_stack((rhos[rho_idxs], thetas[theta_idxs]))
```

Results

- The algorithm accurately detected straight lines from the binary edge image.
- Vectorized implementation significantly reduced computation time compared to nested loops.
- Detected lines can be overlaid on the original image for visualization.

Key Learnings

- Choosing an appropriate **threshold value** is crucial to filtering out insignificant lines.
- From practical trial and error among various images the threshold value of 90 is the most appropriate value.

Applications

- Document scanning and page border detection
- Lane detection in autonomous driving
- Object recognition in robotics
- Industrial line-based defect detection

References

- https://oezeb.github.io/hough-transform/
- https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html