BBM 418 - Computer Vision Laboratory

Assignment 1 Report

Bora Dere
2220765021

Line Detection

Starting to this assignment, we first should detect the lines on the images, so that we can work on them on image de-warping processes. The process of detecting lines in an image involves multiple preprocessing steps to enhance the quality of edge detection. The steps are as follows:

1. **Image Loading and Resizing**: The original digital and distorted images are read using OpenCV and resized to a fixed width while maintaining aspect ratio. The main reason behind this approach was that the images were too large and contained too much details.

$$\text{scale} = \frac{\text{TARGET\_WIDTH}}{\text{dist\_image.shape}[1]},$$
$$\text{new\_size} = (\text{TARGET\_WIDTH}, \text{int}(\text{dist\_image.shape}[0] \times \text{scale})).$$

2. **Grayscale Conversion**: Since color information is not needed for edge detection, the images are converted to grayscale:

$$\text{dig\_gray} = \text{cv2.cvtColor}(\text{dig\_image}, \text{cv2.COLOR\_BGR2GRAY}),$$
$$\text{dist\_gray} = \text{cv2.cvtColor}(\text{dist\_image}, \text{cv2.COLOR\_BGR2GRAY}).$$

3. **Noise Reduction Using Gaussian Blur**: A Gaussian filter is applied to smooth the grayscale image, reducing noise while preserving edges:

$$\text{blurred} = \text{cv2.GaussianBlur}(\text{dist\_gray}, (9, 9), 0).$$

4. **Edge Detection Using Canny Algorithm**: The Canny edge detector is used to extract edges from the blurred image:

$$\text{edges} = \text{cv2.Canny}(\text{blurred}, 100, 200).$$

   The two threshold values (100 and 200) define the hysteresis range, where edges stronger than the upper threshold are retained, and those weaker than the lower threshold are discarded.

5. **Edge Enhancement Using Morphological Operations**: To close gaps in detected edges, a dilation operation is performed using a 3×3 kernel:

$$\text{edges} = \text{cv2.dilate}(\text{edges}, \text{kernel}, \text{iterations} = 1),$$

   where

$$\text{kernel} = \mathbf{1}_{3\times3}.$$

6. **Storage and Visualization**: The processed grayscale and edge images are stored for further processing. Additionally, a sample visualization is generated for each category, showing the original distorted image, the detected edges, and the corresponding ground truth. An example visualization can be found in Figure 1

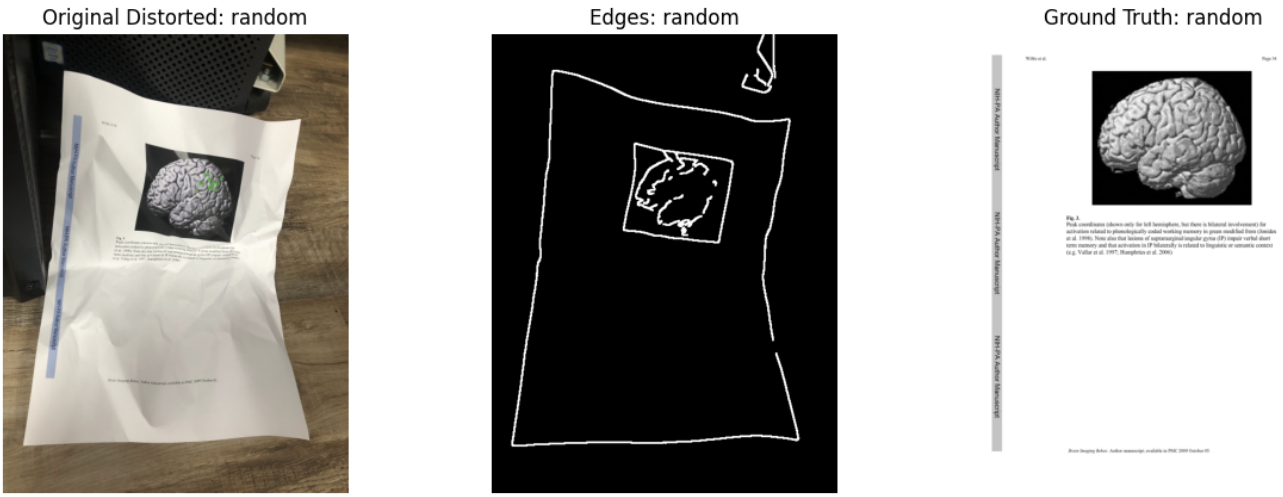

Figure 1: Edge detection result of the first image of random class

Hough Transform for Line Detection

Introduction

The Hough Transform is a feature extraction technique used to detect geometric shapes such as lines, circles, and ellipses in images. It is particularly useful in edge detection applications where the goal is to find straight-line segments from a set of edge pixels. In this project, we implement the Hough Transform for line detection to identify the edges of a document in an image.

Concept of Hough Transform

A straight line in Cartesian coordinates can be expressed as:

$$y = mx + b$$

where m is the slope and b is the y-intercept. However, this representation has difficulties when handling vertical lines, where m approaches infinity. Instead, we use the normal (polar) form of a line:

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

where:

- ρ is the perpendicular distance from the origin to the line.
- θ is the angle between the x-axis and the perpendicular line.
- (x, y) are the coordinates of edge pixels.

This transformation maps a single point (x, y) in Cartesian space to a sinusoidal curve in Hough space (ρ, θ).

Implementation

Hough Transform implementation consists of three key functions:

Accumulator Calculation (`hough_transform` function)

The function `hough_transform(edges, theta_res=1, rho_res=1)` creates a voting accumulator array, where each edge pixel contributes to potential line parameters $(\rho, \theta)2$

- We define a range of angles (θ) from $-90°$ to $90°$.
- We compute the possible values of ρ, considering the diagonal length of the image as the maximum possible distance.

- For each edge pixel, we calculate its corresponding (ρ, θ) values and increment the accumulator at these positions.

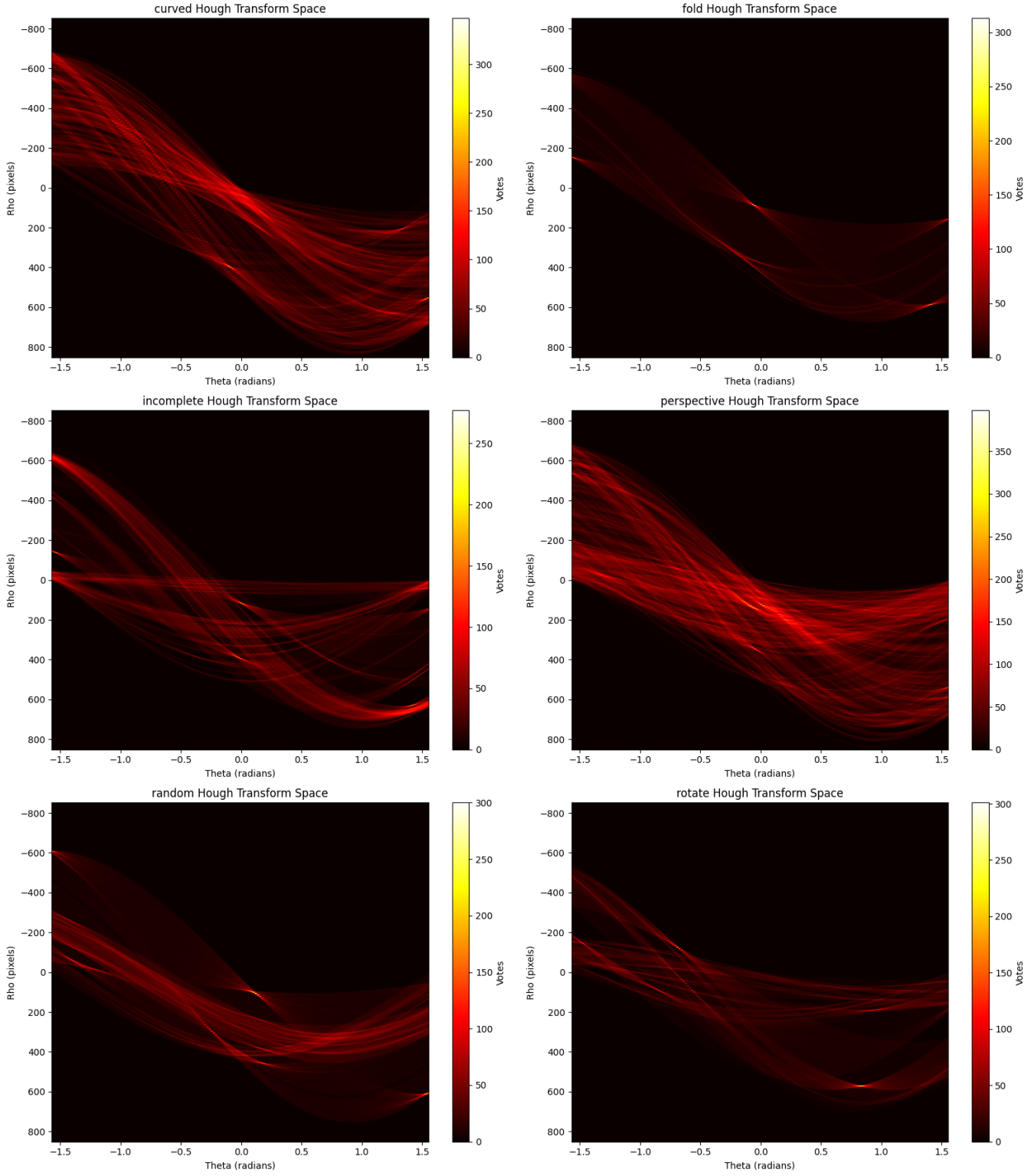

Figure 2: Hough Space

Finding Peaks in Hough Space (`find_hough_peaks` function)

After constructing the accumulator, we identify the most significant lines using `find_hough_peaks(votes, thetas, rhos, threshold_ratio=0.5, max_lines=20)` as in Figure 3

- We define a threshold as a percentage of the highest vote count.

- We check for local maxima in a small window to ensure only the strongest lines are selected.
- The output is a list of detected (ρ, θ) pairs corresponding to the most significant lines.

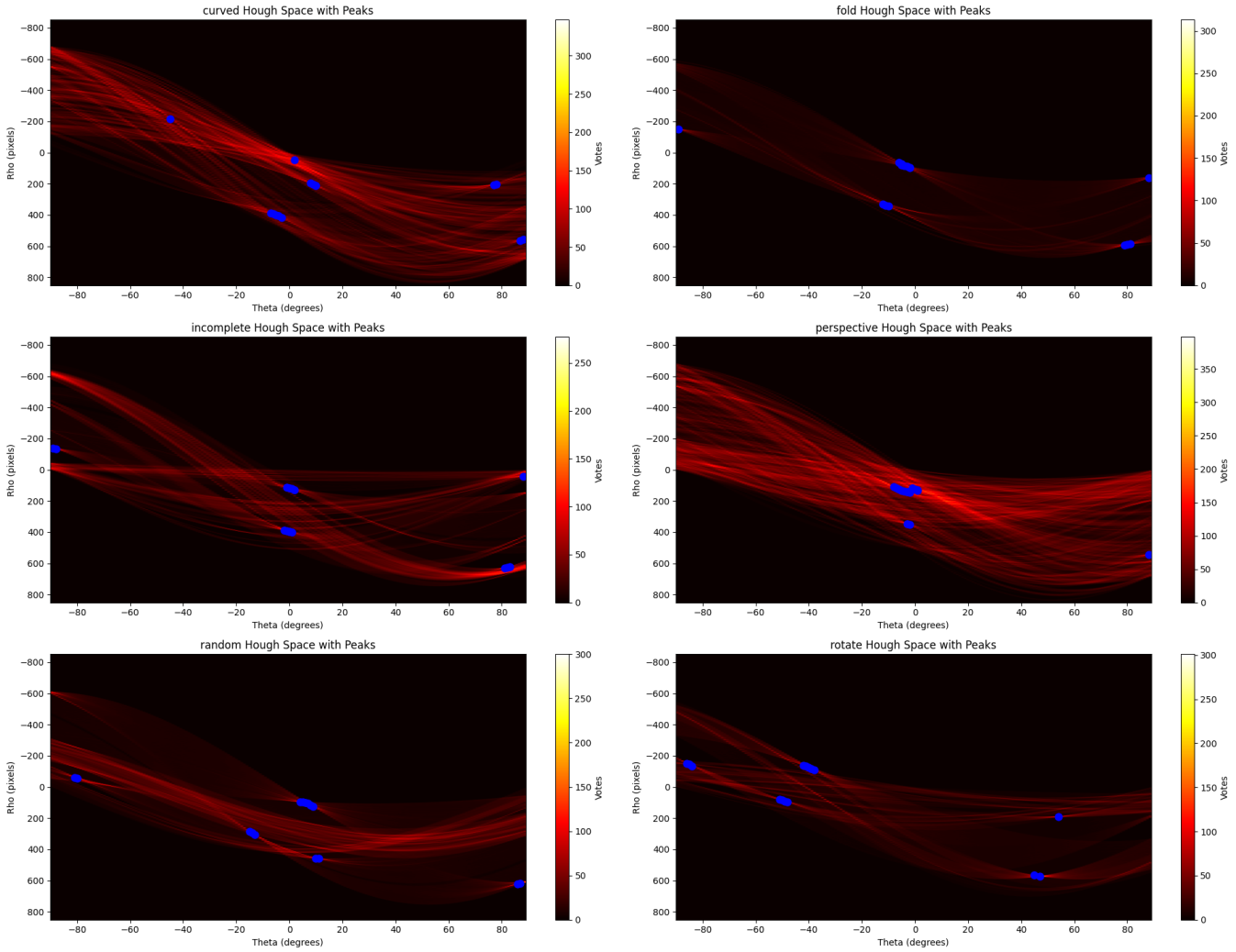

Figure 3: Hough Peaks

Extracting Line Endpoints (`line_points_from_hough` function)

Once we have the parameters (ρ, θ) of a detected line, we compute its endpoints in the image using `line_points_from_hough(rho theta, img_shape)`.

- If the line is nearly vertical ($\cos\theta \approx 0$), we set fixed x-coordinates and compute y-values.
- Otherwise, we solve for the intersection points with the image borders.

Visualization and Interpretation

The detected lines are overlaid on the original image to verify correctness. The expectation is that the document edges are clearly detected. It must be noted that the whole operation does not work well for all the images, creating results far from ideal as in Figure 4 which is actually not the worst result.

Figure 4: 15 Hough Lines for the first image of perspective class

RANSAC for Line Fitting

Overview

Random Sample Consensus (RANSAC) is an iterative algorithm designed to estimate parameters of a mathematical model from a dataset that contains outliers. In the context of line fitting, RANSAC is used to robustly determine the best-fit lines from edge points detected in an image.

Algorithm

The RANSAC algorithm for line fitting follows these steps:

1. **Input:** Extracted edge points from the image.
2. **Initialization:** Set the number of iterations, distance threshold, and minimum required inliers.
3. **Iterative Line Estimation:**
   (a) Randomly select two edge points to define a candidate line.
   (b) Compute the line equation parameters in the form:
   $$ax + by + c = 0.$$
   (c) Normalize the parameters:
   $$\sqrt{a^2 + b^2} = 1.$$
   (d) Compute distances of all edge points to the candidate line.
   (e) Determine inliers as points with distances below the threshold.
   (f) If the number of inliers exceeds the best found so far and meets the minimum threshold, update the best line parameters.
4. **Refinement:** Fit a final line using all inliers.
5. **Output:** Refined set of detected lines.

Implementation

Edge Point Extraction

To extract edge points from an edge-detected image, we locate nonzero pixel coordinates:

$$\text{edge\_points} = (x, y) \mid \text{edges}(x, y) \neq 0.$$

RANSAC Iteration for Line Fitting

For each line detected using the Hough Transform, RANSAC is applied to refine the estimation. The algorithm repeatedly selects two random edge points to form a candidate line. The line equation parameters are calculated based on the selected points:

$$a = -(y_2 - y_1), \quad b = (x_2 - x_1), \quad c = y_1(x_2 - x_1) - x_1(y_2 - y_1).$$

These are normalized for numerical stability:

$$a = \frac{a}{\sqrt{a^2 + b^2}}, \quad b = \frac{b}{\sqrt{a^2 + b^2}}, \quad c = \frac{c}{\sqrt{a^2 + b^2}}.$$

Distance Calculation and Inlier Selection

For each edge point, the perpendicular distance to the candidate line is computed as:

$$d_i = |ax_i + by_i + c| \, .$$

Points satisfying threshold are classified as inliers. If the number of inliers is sufficient, the line is re-estimated using least squares regression over the inliers.

Visualization

Detected lines are drawn on the original image by computing their endpoints:

$$x_1 = 0, \qquad y_1 = \frac{-c}{b}, \; x_2 \qquad = W - 1, y_2 \qquad = \frac{-a(W-1) - c}{b}.$$

where W is the image width. The final set of detected lines is visualized by overlaying them on the input image. [5]

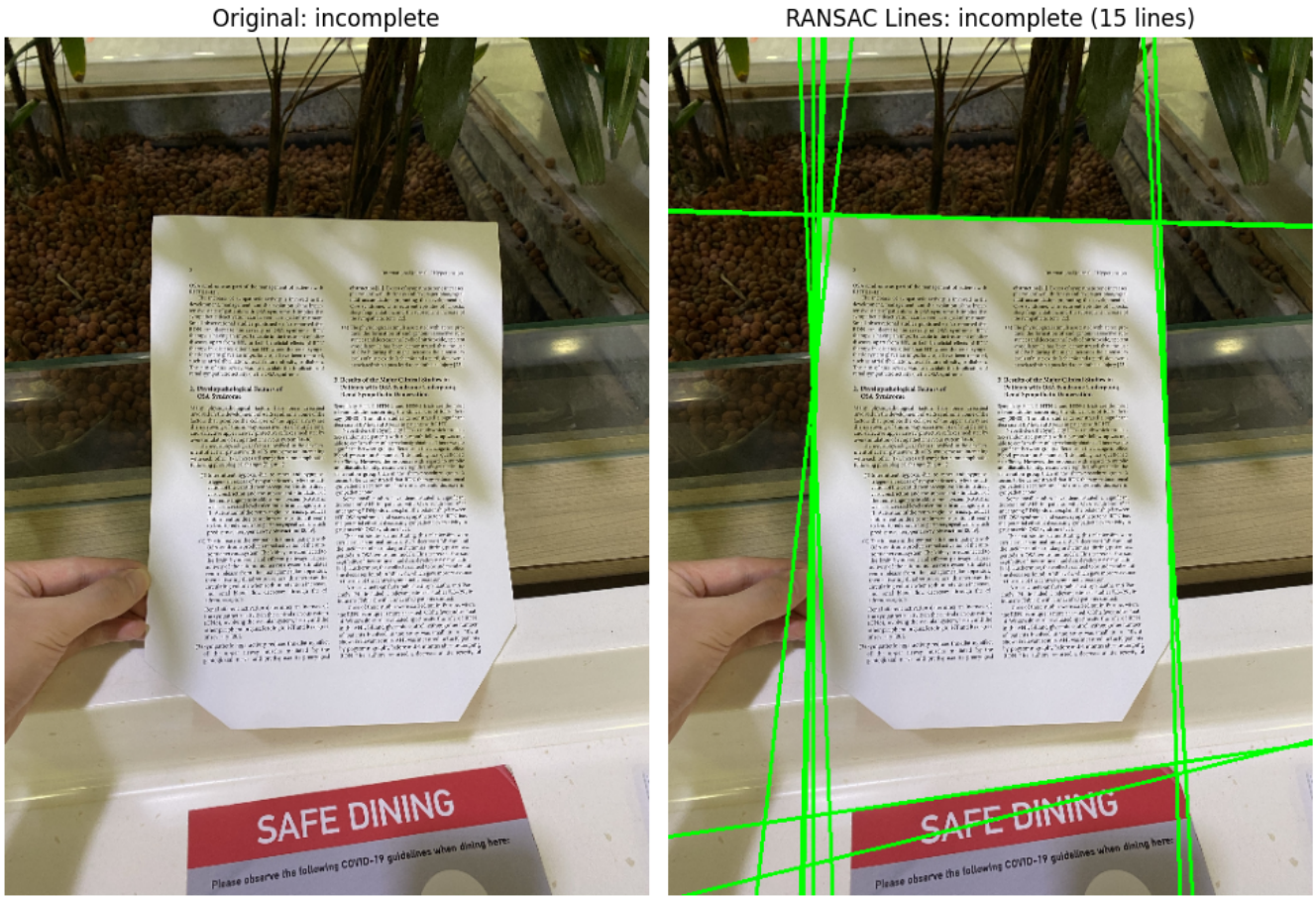

Figure 5: RANSAC lines for the first image of the incomplete class

Geometric Transformations

The process involves detecting quadrilateral boundaries from detected edges, computing a perspective transformation matrix, and applying the transformation to recover a rectified version of the image. The quality of transformation is assessed using Structural Similarity Index (SSIM).

Finding the Quadrilateral

Given a set of detected lines from the Hough Transform and RANSAC, we identify the quadrilateral that best represents the document boundaries. The method follows these steps:

1. Compute the intersection points of detected lines using the line equation form:

$$a_1x + b_1y + c_1 = 0, \quad a_2x + b_2y + c_2 = 0$$

2. Filter intersections that lie within a reasonable margin of the image dimensions.
3. Use a convex hull approach to determine the outer boundary of the detected intersections.
4. Select four representative points by dividing the convex hull points into quadrants relative to the centroid.
5. Order the four selected points to ensure a consistent mapping.

If fewer than four valid intersection points are found, the method falls back to using the full image as a default quadrilateral.

Computing the Perspective Transformation Matrix

To map the detected quadrilateral to a rectangular target shape, we compute a perspective transformation matrix:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

where H is a 3×3 transformation matrix. Given four corresponding point pairs $(x_i, y_i) \rightarrow (x_i', y_i')$, the transformation equations can be rewritten as a system of equations:

$$\begin{aligned} x' &= h_{11}x + h_{12}y + h_{13} \\ y' &= h_{21}x + h_{22}y + h_{23} \\ w' &= h_{31}x + h_{32}y + h_{33} \end{aligned}$$

We solve for the unknowns h_{ij} using a least-squares approach.

Applying Perspective Transformation

The computed homography matrix is used to apply the perspective transformation using inverse mapping:

1. Compute the inverse transformation matrix H^{-1} to map destination pixels back to source pixels.
2. Perform bilinear interpolation to obtain pixel values from non-integer source coordinates.
3. Construct the rectified image by sampling pixel intensities from the transformed coordinates.

Evaluation using SSIM

The transformed images are evaluated against their corresponding ground truth images using the Structural Similarity Index (SSIM). SSIM is computed as:

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

where μ and σ represent local means and standard deviations of the images.

The SSIM scores are computed for each category and summarized as:

`SSIM Scores by Category:`

- Category 1: 0.XXXX
- Category 2: 0.XXXX
- ...

Visualization

For assessment, we visualize the detected quadrilateral, transformed image, and ground truth. A sample visualization (Figure 6) includes:

1. Original distorted image
2. Detected quadrilateral overlaid on the original image
3. Rectified image after perspective transformation
4. Ground truth image for comparison

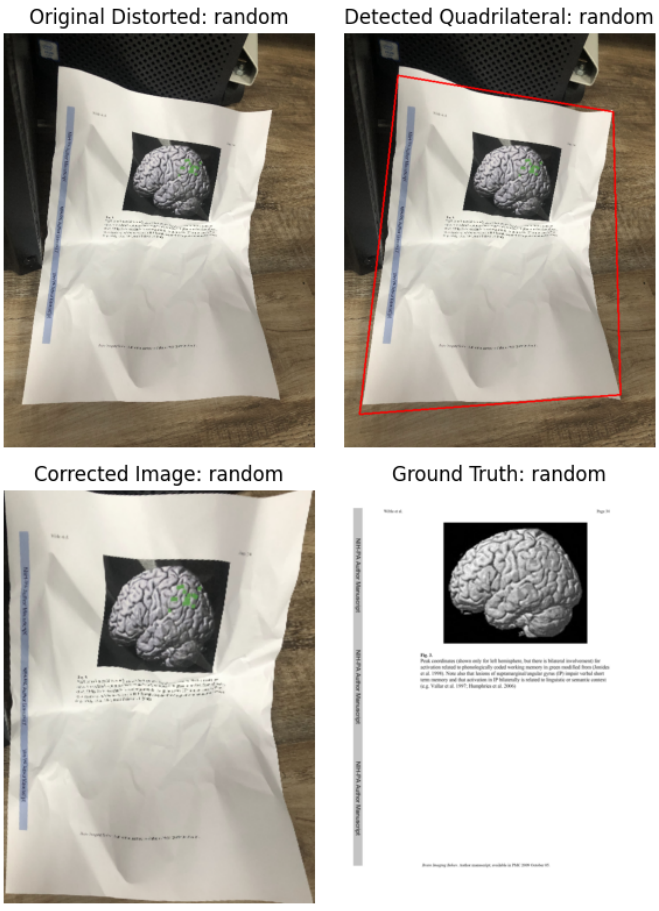

Figure 6: RANSAC lines for the first image of the incomplete class

Results

To refine the geometric transformation results, we conducted a series of experiments with different preprocessing techniques. The primary modifications involved:

- Applying morphological operations such as erosion (`cv2.erode`) to remove weak or spurious edges.
- Adjusting the lower and upper thresholds of the Canny edge detector to optimize edge detection sensitivity.

	cv2.erode	Thresholds	GB Kernel Size	Curved	Fold	Incomplete	Perspective	Random	Rotate
1	No	100-200	-	0.1806	0.1500	0.1854	0.1717	0.1674	0.1671
2	Yes	100-200	3x3	0.1543	0.1582	0.1852	0.1668	0.1650	0.1927
3	Yes	Dynamic	3x3	0.2239	0.2086	**0.2395**	0.2247	0.2328	**0.2278**
4	Yes	Dynamic	9x9	0.2194	0.2081	0.1995	0.2212	0.2059	0.2051
5	Yes	100-200	9x9	**0.2469**	0.2235	0.2095	0.2315	0.2183	0.2234
6	No	Dynamic	9x9	0.2317	0.2219	0.2244	0.2199	0.2289	0.2081
7	No	100-200	9x9	0.2552	**0.2379**	0.2283	**0.2345**	**0.2452**	0.2157
8	No	100-200	5x5	0.2129	0.2199	0.2152	0.2159	0.2165	0.2031
9	No	100-200	7x7	0.2389	0.2316	0.2216	0.2302	0.2159	0.2127

Table 1: Results from experiments.

- Varying the Gaussian blur kernel size to balance noise reduction and edge preservation.
- Implementing dynamic thresholding for the Canny edge detector, where the lower and upper values are computed based on the median pixel intensity of the blurred image:

$$\text{med\_val} = \text{median}(\text{blurred}), \quad \text{lower} = \max(0, 0.66 \times \text{med\_val}), \quad \text{upper} = \min(255, 1.33 \times \text{med\_val})$$

Each combination of these parameters was systematically evaluated, and the resulting transformations were analyzed to determine their impact on edge detection and overall rectification quality. The findings from these experiments can be seen on the Table 1

Further Improvements

Although the experiments conducted have provided valuable insights into the impact of different preprocessing techniques, there are still several ways to further enhance the results. Fine-tuning the parameters and introducing additional methods can potentially improve the accuracy of the detected features. Below are some possible improvements:

- **Optimizing Gaussian Blur Kernel Size:** Experimenting with different kernel sizes (e.g., 5x5, 7x7, 11x11) may help find an optimal balance between noise reduction and edge preservation. Larger kernels provide smoother results, but they may also blur finer details.
- **Refining Canny Edge Detection Thresholds:** Instead of relying on fixed threshold values, a more adaptive approach (e.g., using the median of pixel intensities) can improve edge detection under varying lighting conditions.
- **Adaptive Preprocessing Based on Image Type:** Different types of distortions (curved, folded, rotated, etc.) may require different preprocessing techniques. Implementing a preprocessing pipeline that adjusts dynamically based on the input image characteristics could lead to better results.
- **Experimenting with Different Edge Detection Methods:** While Canny edge detection is widely used, alternative edge detection techniques like Sobel, Laplacian, or structured edge detection can be explored to compare their effectiveness.

By systematically testing different combinations of these techniques and evaluating their effects on various image distortions, a more robust and accurate detection pipeline can be developed.