BBM 418 - Computer Vision Laboratory

Programming Assignment 1 Report

Emirhan Utku
2210765029

1 Introduction

Digital images of documents often suffer from various types of geometric distortions when captured under real-world conditions. For instance, scanning a book page with a phone camera can result in curved text lines near the spine, while scanning a folded or crumpled receipt might produce multiple creases or partial occlusions. Additionally, taking a photograph from an angled perspective causes the document to appear trapezoidal or skewed. Correcting these distortions—often referred to as document dewarping or perspective correction—is crucial to ensure that downstream tasks (e.g., Optical Character Recognition, archiving, or information extraction) operate on clear, rectangular, and fronto-parallel images.

In this assignment, we focus on detecting and correcting these distortions by leveraging classical computer vision techniques. The primary goal is to isolate the region of interest (i.e., the document) from each input image, determine its four corners, and apply a geometric transformation (homography) that re-maps the document onto a proper rectangular plane. We specifically tackle this challenge using the following pipeline:

- **Preprocessing and Edge Detection:** Images are first resized and denoised to improve performance. Canny edge detection is then used to highlight the most salient edges, which typically correspond to document boundaries, folds, or other structural lines.
- **Line Detection via Hough Transform:** Using the edge map, we apply the Hough Transform to detect lines. This process involves accumulating votes for all possible lines passing through the detected edge pixels. Lines that accumulate the highest votes are deemed the most prominent line candidates.
- **RANSAC-Based Refinement:** While Hough Transform can detect many lines in cluttered environments (some of which may be outliers), we apply RANSAC (Random Sample Consensus) to further refine these detections. RANSAC helps in discarding lines that do not genuinely represent document edges by requiring a high inlier consensus among edge points.
- **Quadrilateral (Document) Corner Detection:** After refining lines, we look for intersections to pinpoint potential corners. By identifying the topmost, bottommost, leftmost, and rightmost boundaries, we form a quadrilateral that is assumed to be the document's boundary.
- **Geometric Transformation / Unwarping:** Once the four corners are found, we compute a homography mapping the original document shape onto a straight, rectangular output. We use bilinear interpolation to produce a smooth, distortion-corrected image.
- **Evaluation using Structural Similarity (SSIM):** Finally, we compare the corrected images to their corresponding ground-truth versions using the SSIM metric. This measures how visually similar our unwarped images are to the ideal (or "digital") copies.

To gain insight into the robustness of this method, we evaluate our pipeline on a dataset ([1]) containing six classes of distortions: curved, fold, incomplete, perspective, random, and rotate. Each class poses unique challenges—curvature, large folds, partial document visibility, extreme perspective angles, random clutter, and rotation, respectively. By testing multiple RANSAC thresholds, we highlight how tuning parameters can influence the system's ability to detect correct boundary lines and thus improve the final SSIM scores.

In the following sections, we will explain the implementation of each step in detail, describe the challenges we encountered and how we solved them, what we did for better results, discuss the findings and observations from the experiments, and finally provide recommendations to further improve this pipeline for real-world document redaction tasks.

2 Methodology

2.1 Edge Detection

To reliably isolate document boundaries and other structural edges, we apply a custom edge detection strategy that leverages a combination of Gaussian blurring and Canny thresholds determined from the image's median intensity. Below is a summary of the steps and the rationale behind each design choice. An example of the original and edge-detected images can be seen in Figure 1.

2.1.1 Large Gaussian Blur Window ((31, 31))

We intentionally use a large kernel size for the Gaussian blur to heavily smooth the input image. Document images often contain high-frequency textures (e.g., text, slight wrinkling) that can produce spurious edges when running Canny. By applying a large blur, we reduce such noise and make prominent structures (like the outer boundary) more distinct.

2.1.2 Adaptive Thresholds from the Median

After blurring, we compute the median intensity of the blurred image. We then set the Canny lower and upper thresholds in proportion to this median (approximately $0.66 \times$ median and $1.33 \times$ median, each shifted by a small constant of $+20$).

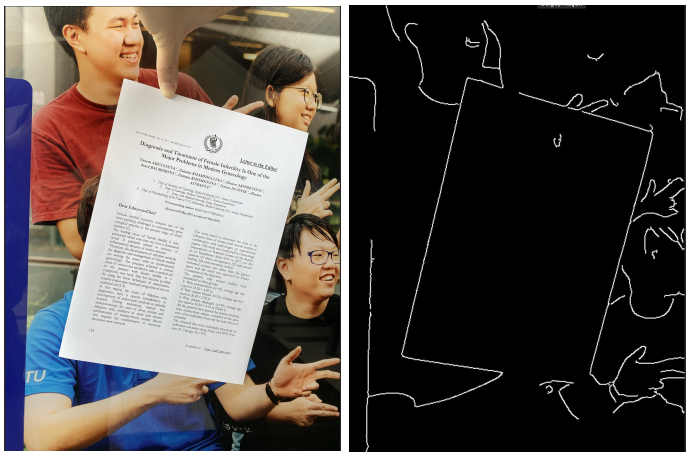
- **Why the median?** The median is more robust to outliers than the mean, making it a good proxy for the "typical" intensity in the blurred image. This leads to more adaptive thresholds that respond to varying lighting conditions or global brightness levels.

2.1.3 Iterative Blur Adjustment (the while loop)

Inside a while loop, we monitor the number of detected edge pixels. If the edge count is below a certain threshold (4000 in this code), we slightly decrease the blur parameter and re-run Canny. This ensures we do not oversmooth the image and lose crucial edges:

- If there are too few edges, the Hough transform might not detect enough lines, harming the final perspective correction.
- Reducing the blur effect reintroduces some higher-frequency details, allowing us to reach a balance where sufficient edges are present to capture the document boundaries without being overwhelmed by noise.

Together, these steps help stabilize the edge detection phase—yielding enough edges for robust line detection while preventing an excessive number of noise-induced edges that would confuse subsequent Hough and RANSAC processes.



(a) Original Image (b) After Edge Detection

Figure 1: Comparison of Original and Edge Detected Images.

2.2 Hough Transform for Line Detection

After obtaining a stable edge map, we detect candidate lines via a custom implementation of the Hough Transform. Below are the key points about our approach and why the Hough Transform is particularly suitable in this context (see Figure 3).

2.2.1 Implementation Details

Algorithm Outline

The Hough Transform is particularly useful for detecting straight lines in an image. It works by transforming each point (x, y) in the Cartesian space into a sinusoidal curve in the Hough space. The intersection point of these sinusoidal curves indicates the presence of a line (see Figure 2).

- We define a ρ range spanning $-$diag_len to $+$diag_len, where diag_len is the image diagonal length (the maximum possible ρ in the image space).
- We divide θ from 0 to π using a $\theta_{\text{resolution}}$ (defaulted around one degree increments in radians).
- For each edge pixel, we compute the corresponding ρ values for each θ and increment an accumulator array at (ρ, θ).
- The accumulator thus tallies "votes" for all possible lines passing through the edge points.

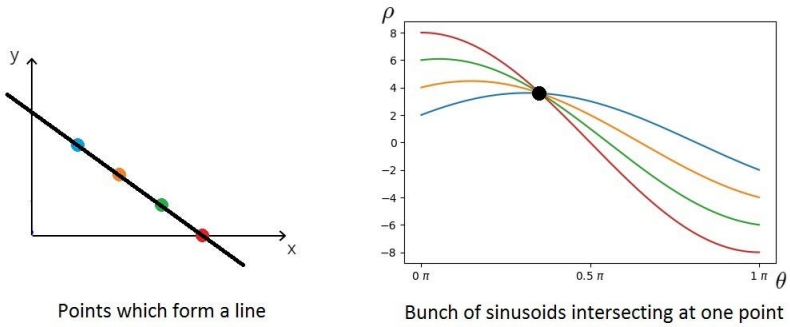


Figure 2: Hough Transform representation

Importance of $\theta_{\text{resolution}}$ and $\rho_{\text{resolution}}$

- $\theta_{\text{resolution}}$ (e.g., $\pi/180$ for 1-degree increments) determines how finely we sample possible line orientations. A finer resolution can detect lines at more precise angles but requires greater computational cost. A coarser resolution may merge near-horizontal or near-vertical lines into a single bin.
- $\rho_{\text{resolution}}$ (often set to 1) specifies the spacing between consecutive ρ values. A larger spacing might cause lines that are close together in ρ to be grouped, whereas a smaller spacing can help distinguish lines that differ only slightly in distance from the origin.
- Balancing these resolutions is key to accurately detecting lines without overwhelming the algorithm with excessive parameter bins.

Selecting the Top Candidate Lines

Once we fill the accumulator, we sort all (ρ, θ) pairs by their vote counts and keep a fraction of the most prominent candidates. We then ensure that the final number of lines is between min_lines and max_lines, preventing two extremes:

- Too few lines might omit important document boundaries, leading to incomplete or no quadrilateral detection.
- Too many lines can cause excessive noise in subsequent RANSAC steps and risk merging or confusing multiple edges.

By bounding the number of lines, we strike a practical balance between missing crucial lines and overburdening the system with spurious ones.

2.2.2 Why Hough Transform?

- **Robustness to Noise and Occlusion:** Even if an edge is interrupted or partially occluded, the Hough Transform can still accumulate sufficient votes to detect the overall line.
- **Simplicity for Line Detection:** Other methods like gradient-based or segment-based line detection might require complex heuristics, while Hough relies on a straightforward voting procedure that naturally handles cluttered scenes.
- **Ideal for Document Boundaries:** Document edges tend to be highly contrasted straight lines, which the Hough Transform reliably pinpoints despite folds or partial shadows.

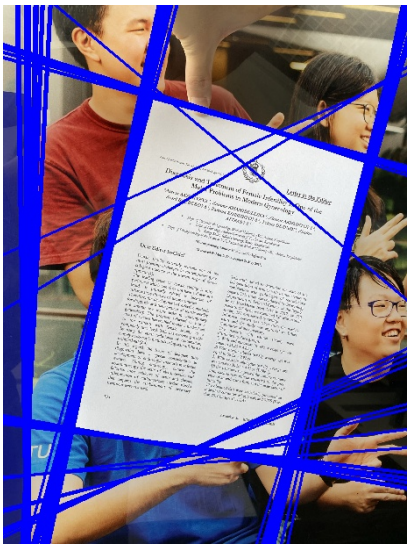


Figure 3: Detected lines using the Hough Transform.

2.3 RANSAC for Line Refinement

After using the Hough Transform to obtain a set of potential lines, we leverage RANSAC (Random Sample Consensus) to refine these detections and discard outliers. The main idea is to repeatedly sample subsets of points, fit a line to those subsets, and count how many other points (inliers) are within a certain distance threshold of that fitted line. The line model with the highest inlier count is the final estimate (see Figure 4).

In the code, we pass a dynamically adjustable RANSAC distance threshold (e.g., 125, 150, or 200), alongside hyperparameters such as the number of iterations (up to 2500) and per-point distance checks. These hyperparameters were tuned empirically for stable results, and we will analyze their impact further in our Findings section.

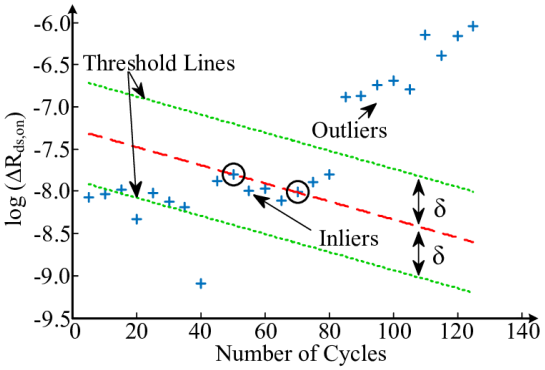


Figure 4: RANSAC algorithm illustration

We employ three key "control" functions to converge on the final set of lines:

- **is_similar_line(...)** compares angles and distances of two lines and merges them if they are nearly identical.
- **double_check_refined_lines(...)** samples points along each line to check if two lines effectively overlap, retaining only the one with the higher inlier count.
- **pick_rectangle_lines(...)** filters the refined lines to identify the topmost, bottommost, leftmost, and rightmost boundaries of the document.

We will go into detail about these in the Findings section.

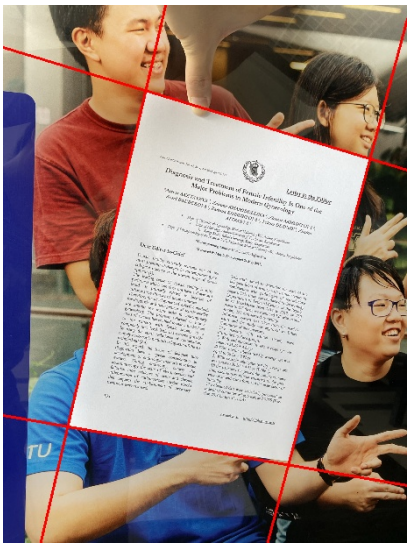


Figure 5: Refined Lines obtained after all controls

Why RANSAC?
RANSAC is well-suited for scenarios containing noise or outliers—which is common in folded, curved, or partially occluded paper edges. It iteratively "tests" line models on random subsets of points, ensuring that only those models that achieve a high inlier consensus get retained. Compared to deterministic least-squares or purely voting-based methods, RANSAC is:

Pros:

- Highly robust to outliers; it can find a valid line even when a significant portion of points are spurious.
- Straightforward to implement and tune for line-fitting tasks.

Cons:

- The iterative approach can be slower than direct analytic methods, especially if a large number of iterations or high-resolution sampling is required.
- RANSAC might fail if the proportion of outliers is extremely high or if hyperparameters are poorly tuned.

By dynamically adjusting the distance threshold and refining lines through these three function calls, we obtain a small and accurate set of edges that represent the true boundaries of the document.

2.4 Quadrilateral Detection and Perspective Transformation

To correctly isolate and unwarp the document region, we rely on a two-stage process: (1) finding the corners of a quadrilateral that encloses the document; and (2) applying a custom perspective transform to map this quadrilateral onto a proper rectangular output (see Figure 6 and Figure 7).

2.4.1 Quadrilateral Detection

Once the line refinement and filtering steps yield a set of lines (most of the time there are 4 lines), we compute their intersection points. We check each pair of lines for intersection, ensuring that any calculated point is within image boundaries. We then aggregate these intersections and compute the convex hull of the resulting point cloud:

Intersection Computation

Each pair of lines (A_1, B_1, C_1) and (A_2, B_2, C_2) is intersected by solving a small system of linear equations. We keep only valid intersection points that lie inside the image dimensions.

Convex Hull

We pass the intersection points to `convexHull`, which returns the smallest convex polygon enclosing all these points—commonly called the hull. The hull essentially "wraps" the set of intersections in a minimal bounding contour.

Polygon Approximation

After computing the hull, we use `approxPolyDP` to simplify it. If this approximation yields exactly four points, we consider them as the document corners. Otherwise, we revert to using the hull's points as a fallback.

These four points define the corners of a quadrilateral we assume to be the boundary of the document.

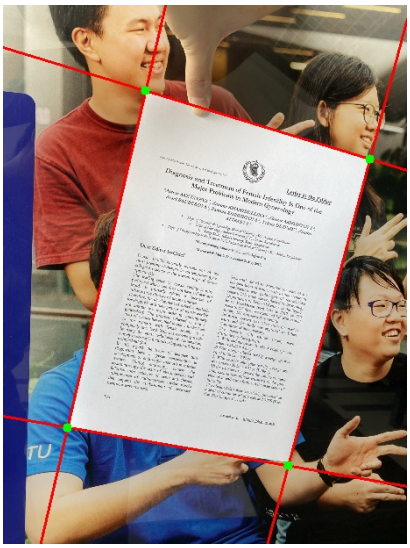


Figure 6: Detected corners of the document

2.4.2 Perspective Transformation

To map this quadrilateral onto a rectangle that matches the original document's shape, we use a homography-based perspective transform. In particular, we implement four helper functions to fully control the process:

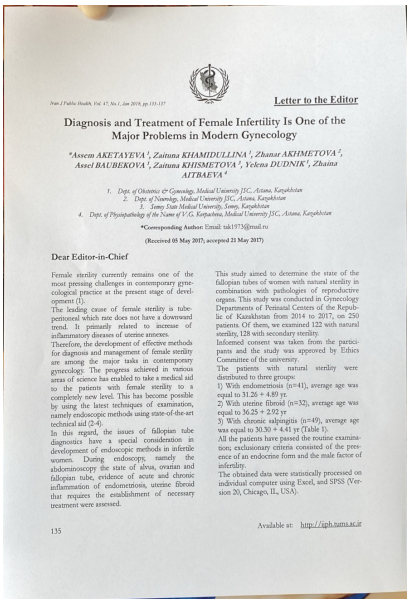
1. **order_points(pts)** :
   - Given four corner points, we reorder them in a consistent manner: top-left, top-right, bottom-right, and bottom-left. This order is crucial for applying the correct mapping to a rectangular output.

2. **compute_homography(src_pts, dst_pts)** :
   - Computes the 3×3 homography matrix $\mathbf{H}$ from four source points to four destination points. The code forms an 8×9 system of equations, then uses Singular Value Decomposition (SVD) to solve for $\mathbf{H}$. We normalize $\mathbf{H}$ so that $\mathbf{H}[2, 2] = 1$.

3. **warp_image_bilinear(src_image, H, out_width, out_height)** :
   - Performs the **actual warping** of the source image onto the output plane. We invert the homography and for each pixel in the output image, back-project it into the source image's coordinate system. We use **bilinear interpolation** to compute the pixel intensity, yielding a smooth result without blocky artifacts.

4. **four_point_transform_manual_bilinear(src_image, src_pts)** :

- A convenience function that chains all of the above steps together:
  - Orders the corner points.
  - Computes the width and height of the new document view.
  - Constructs the target rectangle corners.
  - Invokes **compute_homography** and **warp_image_bilinear** to produce the **unwarped** image.



Letter to the Editor

Diagnosis and Treatment of Female Infertility Is One of the Major Problems in Modern Gynecology

Dear Editor-in-Chief

Figure 7: Warped document after perspective transform

Which Geometric Transform Is Used?

We rely on a **perspective (projective) transform** represented by a 3×3 homography matrix. This type of transform is significantly more flexible than rigid, similarity, or affine transformations:

- A **rigid transform** (combining only rotation and translation) would be too restrictive, since it cannot handle changes in scale or angles that occur when a page is tilted.
- An **affine transform** (allowing translation, rotation, uniform or nonuniform scaling, and shear) still cannot capture **true perspective distortions**—for instance, when one edge of the paper appears smaller or shorter due to foreshortening.
- A **projective (homographic) transform**, on the other hand, enables us to accurately map a **quadrilateral** (the paper in perspective) to a rectangle in the output plane, preserving the straight lines of the document margins while correcting perspective-induced trapezoidal shapes.

By determining four point correspondences (the corners of the document in the image to the corners of a chosen output rectangle), we can solve for the homography. This allows a **complete** perspective correction that recovers the original rectangular shape of the page.

Why This Method?

Perspective transformation via homography provides **precise** unwarping and is well-suited for document images that might be photographed at arbitrary angles. Some key benefits include:

1. **Accurate Modeling of Real Scenes**
   Real documents under typical photographic conditions experience perspective distortions. A homography can correct these, producing a **straightened** view of the text and graphics.
2. **Stable and Well-Understood Mathematics**
   Homography-based warping has been a fundamental tool in computer vision for decades, making it relatively straightforward to **implement, debug, and extend**. The direct use of linear algebra (SVD) to solve for the transform makes it mathematically transparent.

3. **Handles Complex Distortions**
   Because homographies can model **foreshortening**, a rectangle viewed obliquely (appearing as a trapezoid in the image) can be accurately remapped. Simpler transforms like affine or similarity would be unable to recover the full shape.

4. **Minimizes Visible Artifacts**
   Our usage of **bilinear interpolation** during warping helps avoid sudden intensity jumps or "blocky" areas. This ensures a smoother, more natural-looking corrected image, which is especially beneficial for reading text near the edges of the page.

Overall, **perspective transformation** is the natural choice whenever **any** camera tilt or skew is present; it preserves line geometry and effectively turns the document image to a flat, front-facing view.

3 Findings

In this section, we highlight several **practical choices** and **parameter settings** that helped the algorithm handle a variety of document distortions (e.g., folds, perspective tilts, curvature). Below are the main findings:

3.1 Dynamically Setting the RANSAC Distance Threshold (in `is_similar_line`)

One of the first observations was that the **distance threshold** used in our `is_similar_line` function (e.g., 125, 150, or 200) plays a **critical role** in merging line candidates that are essentially duplicates.

Recall that `is_similar_line(line1, line2, angle_thresh_deg, dist_thresh)` checks whether two lines are close in orientation (angle difference below `angle_thresh_deg`) and in their perpendicular distance from the origin (within `dist_thresh`).

- **Lower Threshold (e.g., 125)**
  This makes the merging criterion more **strict**, so lines must be *very close* in distance for us to treat them as duplicates. This can preserve some genuine lines that represent distinct edges. However, it also risks *leaving too many lines*, which can clutter subsequent filtering steps.

- **Higher Threshold (e.g., 200)**
  A more **generous** merging criterion allows lines that deviate significantly in distance to be considered "similar," sometimes *over-merging* distinct edges. This can be helpful when dealing with heavily distorted documents (e.g., deep folds) but risks incorrectly collapsing multiple boundaries into one.

Overall, no single distance threshold **consistently** outperforms the others for every distortion type. For instance, **125** is more effective for *curved* images, achieving the highest SSIM of **0.5345**.Meanwhile, for *fold* images, **150** yields a stronger result (SSIM **0.5164**), indicating that a moderate threshold merges lines more effectively in the presence of folds. In the *incomplete* folder, the highest performance (SSIM **0.4511**) is reached at **200**, demonstrating that a higher threshold is necessary to capture partial boundaries. Likewise, *perspective*-distorted images see their best correction (SSIM **0.5435**) at **150**, while *random* distortions show only slight variations but still peak at **150** (SSIM **0.5132**). Finally, for *rotate*, **200** emerges as the top choice (SSIM **0.4663**), reflecting the need for a more lenient merging criterion to handle pronounced rotation angles.

Looking at the *sample images* confirms that the **complexity of each distortion** dictates which threshold works best. Documents with clean, well-defined edges benefit from lower thresholds that avoid merging distinct lines, whereas **major folds** or *incomplete boundaries* demand a higher threshold to ensure the essential edges are recognized. Consequently, dynamically adjusting the threshold is a **critical** step for robust performance across various classes of real-world document distortions.

3.2 Three Key Control Functions in RANSAC

After RANSAC proposes a set of lines, we use three **auxiliary functions** to further refine and prune them:

1. `is_similar_line(line1, line2, angle_thresh_deg, dist_thresh)`
    - Compares the orientation (angle) and the perpendicular distance from the origin for both lines.
    - If their angles differ by less than `angle_thresh_deg` and their distance difference is within `dist_thresh`, the lines are considered duplicates.
    - This helps **immediately merge** lines that are effectively the same boundary but might have been detected separately.
2. `double_check_refined_lines(refined_lines, image_shape, geom_thresh=85, n_samples=10)`
    - Samples each line at `n_samples` positions and calculates the average distance of these sampled points to every other line.
    - We use a `geom_thresh` (85 pixels by default) to decide if two lines are close enough across multiple sample points.
3. `pick_rectangle_lines(refined_lines)`
    - From the final set of lines, selects and returns just the top, bottom, left, and right boundaries of the document by inspecting line intercepts.
    - Ensures that interior folds or smaller boundary fragments do not interfere with the quadrilateral detection.

Why Do We Need These Functions?
The `is_similar_line` function takes the line with the most inlier points instead of selecting all the best lines that are very close to each other in terms of angular and distance, as obtained from the RANSAC process.

Handling Close Lines with Different Angles:
Sometimes, even though two lines are very close to each other, their angle differences cause the distances to exceed the threshold defined in the `is_similar_line` function. As a result, these lines are not classified as similar. To address this, we check the geometric distances in the `double_check_refined_lines` function. This ensures that lines that are spatially close (on average) across their length are treated as the same line.

Handling Excess Lines:
In rare cases, it is possible to end up with more than four lines after the refinement process. In such situations, we use the `pick_rectangle_lines` function to select the four lines that best form a rectangular shape. This way, we maintain a clear and accurate quadrilateral representation of the document boundaries.

These three control functions work together to ensure that the lines produced by RANSAC are consolidated into a **concise set** of true document edges, even when multiple near-duplicate lines exist or folds create confusing overlaps.

3.3 Effect of the Scale Factor = 0.3

We chose to **downscale** images by a factor of 0.3 to **speed up** the algorithm and make parameter tuning more stable. Working with high-resolution images directly can:

- Create **extremely large** edge maps and Hough parameter spaces, slowing down the entire process.
- Introduce too many fine details (e.g., small wrinkles, text edges), overshadowing the main document boundary lines.

At **scale = 0.3**, we found a good balance between **performance** (fewer points to process) and **accuracy** (enough detail to detect the page contour). In practice, if extremely high precision is required (e.g., for an OCR pipeline), one could refine the corners at a higher scale. However, for standard document capture scenarios, the 0.3 factor worked well and was more computationally efficient.

4 Results and Discussion

The SSIM values below provide a quantitative assessment of our dewarping results across six distortion types, using three different distance thresholds in the RANSAC line-merging stage.

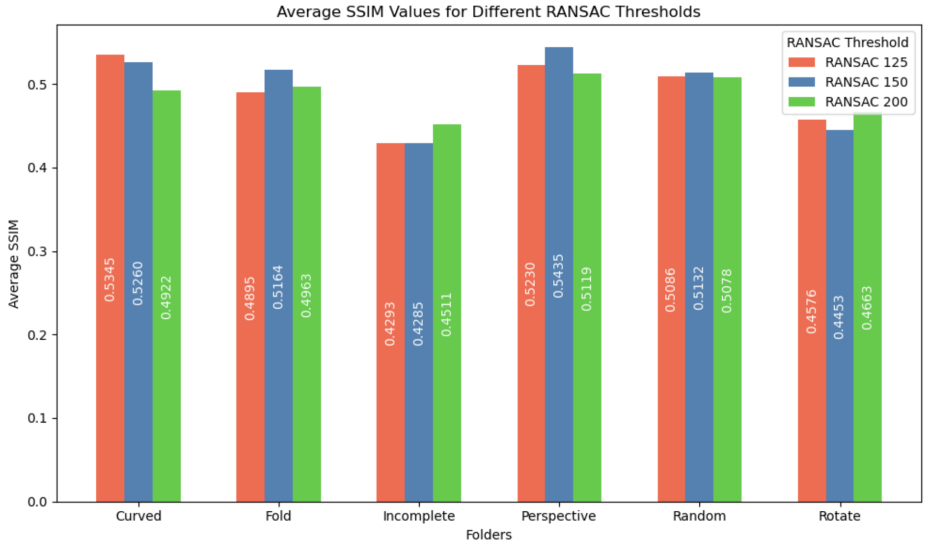
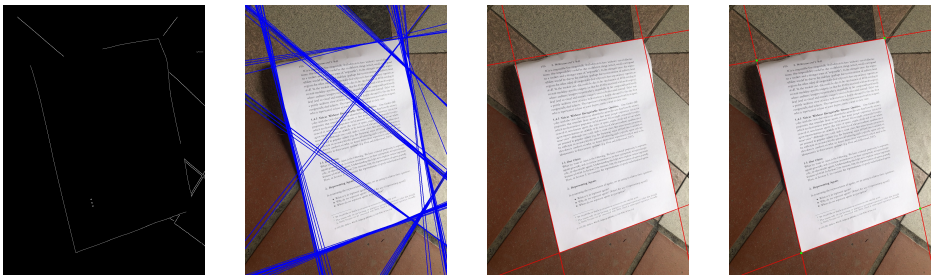


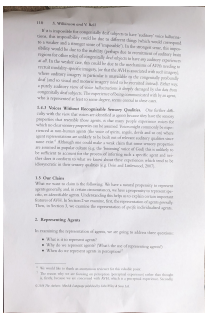
Figure 8: Average SSIM Values for Different RANSAC Thresholds.

4.1 Interpretation of Results

Curved: The highest score corresponds to threshold 125. Because the page edges in these images are clearly visible and exhibit minimal confusing background textures, Hough and RANSAC rarely detect false lines. A stricter threshold effectively merges near-duplicate lines without losing the main document boundary.



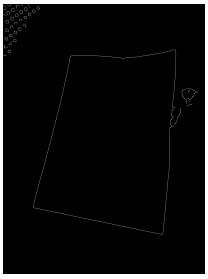
(a) Detect edge (b) Hough Lines (c) Refined Lines (d) Detect corner



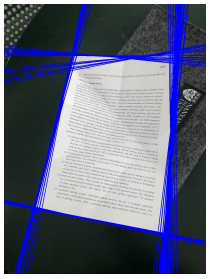
(e) Final Perspective

Figure 9: Good results for Curved folder

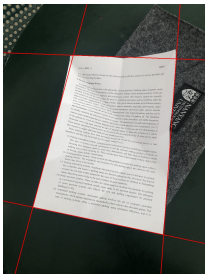
Fold: Folded pages sometimes produce extra edges where creases form. With a more relaxed threshold at 150, spurious lines are merged properly, leaving the true outer boundaries intact. Hence, the score at 150 emerges as the best score here.



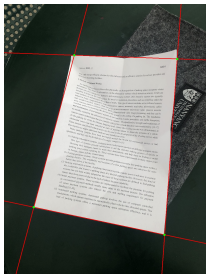
(a) Detect edge



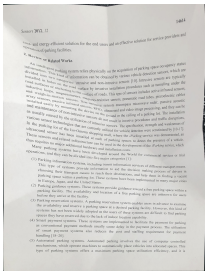
(b) Hough Lines



(c) Refined Lines



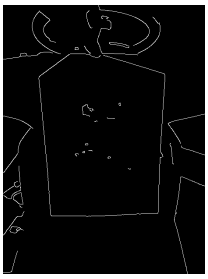
(d)Detect corner



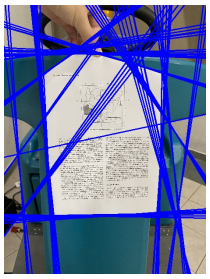
(e) Final Perspective

Figure 10: Good results for Fold folder

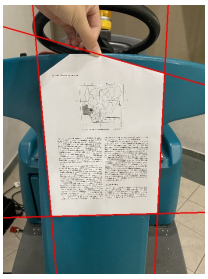
Incomplete: In this category, the page typically does not show four well-defined edges, complicating both Hough detection and RANSAC refinement. A more tolerant threshold (200) yields the highest SSIM, indicating that less strict merging occasionally preserves partial edges better when the true paper boundary is missing or truncated.



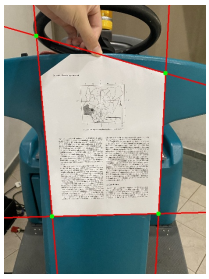
(a) Detect edge



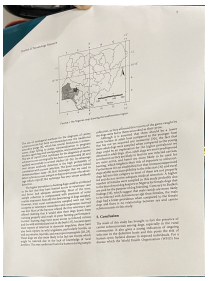
(b) Hough Lines



(c) Refined Lines



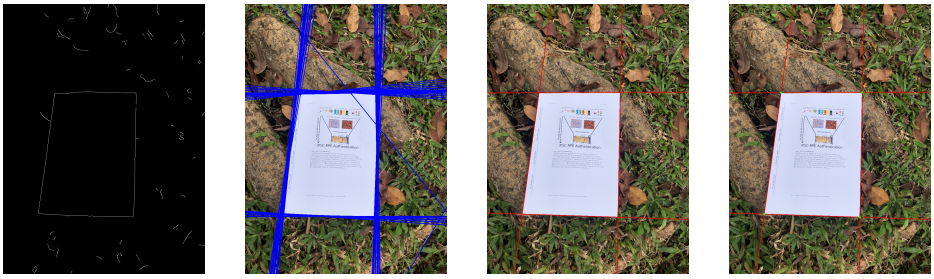
(d) Detect corner



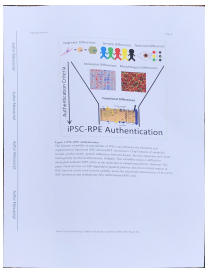
(e) Final Perspective

Figure 11: Good results for Incomplete folder

Perspective: Extreme perspective angles can produce trapezoidal shapes. If Hough occasionally outputs false lines, a threshold of 150 works best to unify slight variations of the same boundary, achieving a higher SSIM.



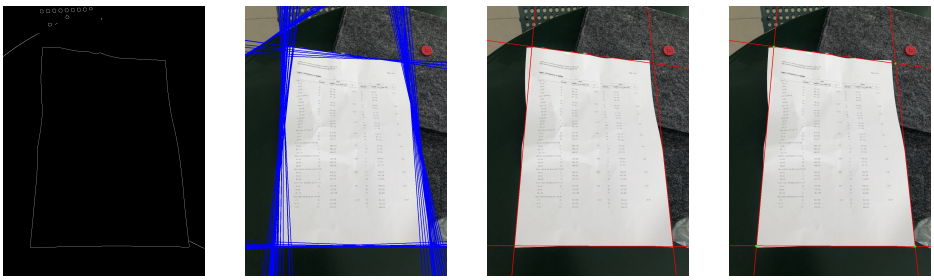
(a) Detect edge (b) Hough Lines (c) Refined Lines (d) Detect corner



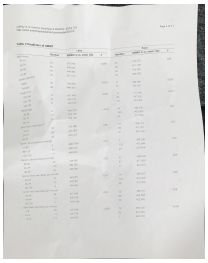
(e) Final Perspective

Figure 12: Good results for Perspective folder

Random: Random distortions (blend of small folds, perspective shifts, or rotations) are moderately addressed by all thresholds. Here, 150 still produces the highest average, balancing keeping true edges separate without merging them prematurely.



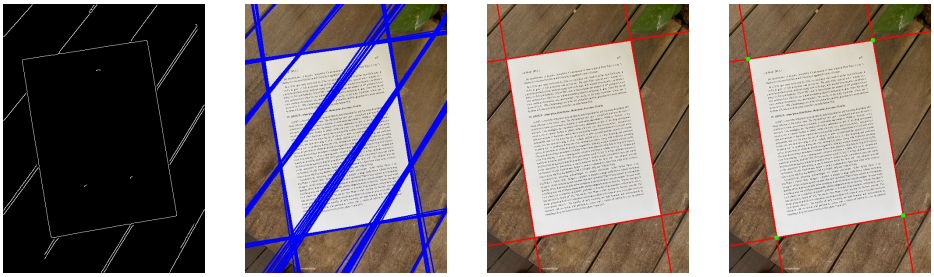
(a) Detect edge (b) Hough Lines (c) Refined Lines (d) Detect corner



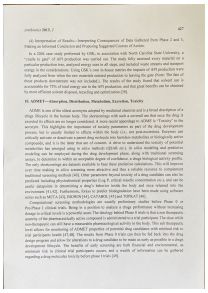
(e) Final Perspective

Figure 13: Good results for Random folder

Rotate: In rotation-heavy images, background pixels can share similar intensity with the page, complicating edge detection. A looser threshold at 200 merges small discrepancies in the perimeter, leading to the best average SSIM.



(a) Detect edge (b) Hough Lines (c) Refined Lines (d) Detect corner

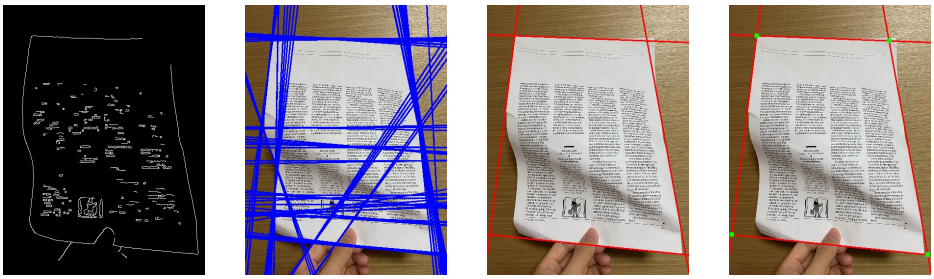


(e) Final Rotation

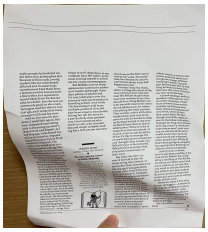
Figure 14: Good results for Rotate folder

4.2 Overall Observations

1. **Clear Boundaries (Curved):** With minimal background interference, a lower threshold (125) efficiently handles edge consolidation without sacrificing real edges.However, due to the paper structure, poor results are also obtained.



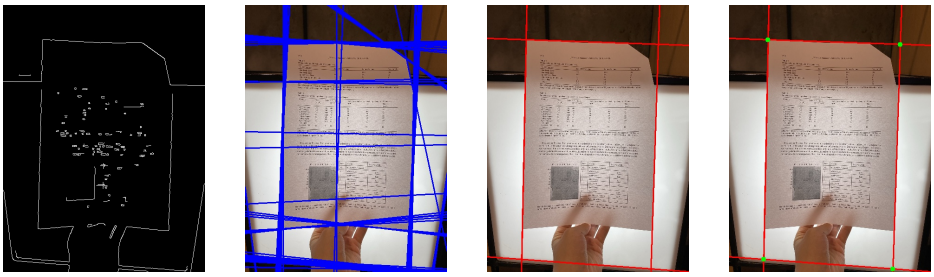
(a) Detect Edge (b) Hough Lines (c) Refined Lines (d)Detect Corner



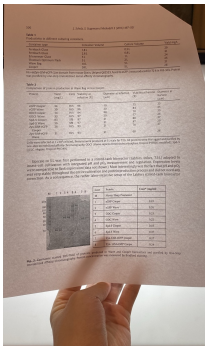
(e) Final Perspective

Figure 15: Bad results for Curved folder

2. **Missing Edges (Incomplete):** When a full outline is not visible, a higher threshold is beneficial, preventing over-pruning partial edges.



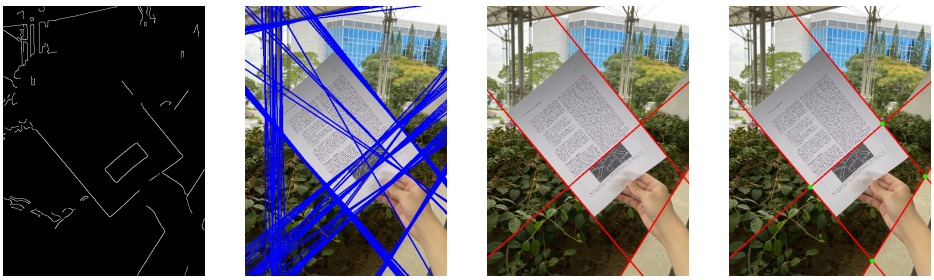
(a) Detect edge (b) Hough Lines (c) Refined Lines (d) Detect corner



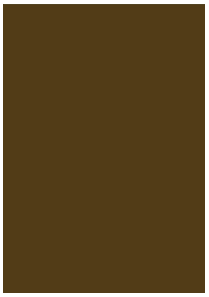
(e) Final Perspective

Figure 16: Bad results for Incomplete folder

3. **Complex Background (Rotate):** Similar intensities cause edge detection to latch onto irrelevant regions, so being more lenient (threshold = 200) helps unify the correct boundary lines.



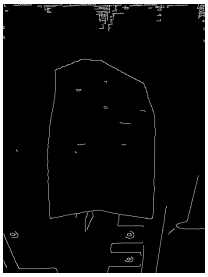
(a) Detect edge (b) Hough Lines (c) Refined Lines (d) Detect corner



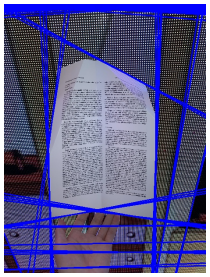
(e) Final Perspective

Figure 17: Bad results for Rotate folder

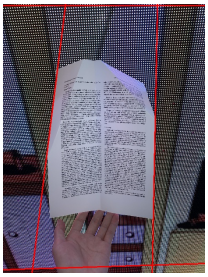
4. **Balanced Threshold (Fold, Perspective, Random):** A mid-range threshold (150) excels at filtering out false lines from folds, slight warps, or moderate perspective transformations. However, when there is too much noise in the background, both folders are badly affected.



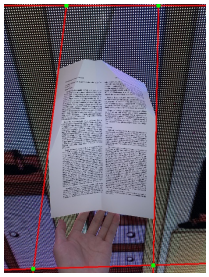
(a) Detect edge



(b) Hough Lines



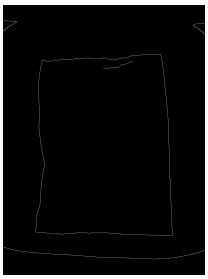
(c) Refined Lines



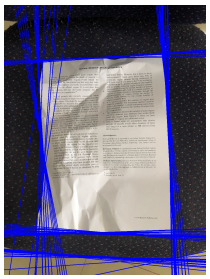
(d) Detect corner

(e) Final Perspective

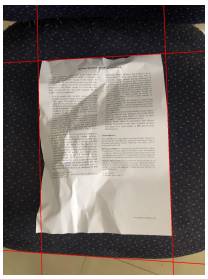
Figure 18: Bad results for Fold folder



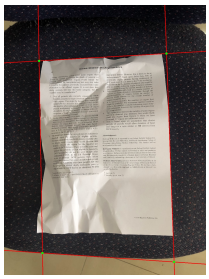
(a) Detect edge



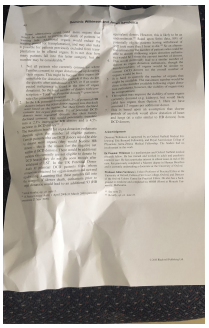
(b) Hough Lines



(c) Refined Lines



(d) Detect corner



(e) Final Perspective

Figure 19: Bad results for Random folder

5 Potential Improvements

1. **Enhanced Edge Detection**

   - **Adaptive Methods:** Instead of using a fixed Gaussian blur window and median-based thresholds, we could adopt auto-parameterized Canny or use guided filtering to preserve stronger edges while smoothing out noise. Additionally, edge-preserving filters (e.g., bilateral filtering) might better retain paper boundaries while eliminating minor text or wrinkles.
   - **Morphological Post-processing:** Once we obtain the raw edge map, morphological operations (closing, dilation) could help fill small gaps in the boundary contour, making the subsequent line detection more robust.

2. **Adaptive RANSAC Hyperparameters**

   - **Dynamic Inlier Threshold:** We used a fixed threshold (e.g., 1.5 pixels internally). An adaptive variant that factors in the current scale of the image or the local density of edges might yield more consistent line-fitting results.
   - **Adaptive Iterations:** Instead of running a fixed number of iterations (2500), we could terminate early if we converge on a stable solution, potentially reducing runtime.

3. **Hierarchical or Multi-scale Approaches**

   - By operating at multiple scales, we can detect prominent global features (e.g., outer edges) at a coarse scale and then refine line placement at a finer resolution. This multi-scale approach can handle heavily folded or curved documents more reliably.

4. **Advanced Quadrilateral Detection**

   - If the final set of lines still includes some spurious or near-duplicate edges, additional checks (e.g., enforcing orthogonality for opposite sides or parallel constraints) might improve our detection of the true document outline.

5. **Better Homography Estimation**

   - Although our current approach works well for four-corner documents, more advanced non-planar corrections could be explored for highly curved pages (e.g., mesh-based warping).
   - Similarly, introducing a bundle adjustment or leastsq refinement step after the initial homography could reduce small alignment errors near corners.

Collectively, these improvements could further increase edge detection accuracy, stabilize line fitting, and enhance the final perspective correction—especially when dealing with challenging real-world documents that feature heavy clutter, folds, or incomplete boundaries.

6 Conclusion

In this assignment, we built and tested a complete pipeline for document dewarping, combining edge detection, Hough-based line detection, RANSAC refinement, corner extraction, and homography. Our experiments show that adjusting thresholds—especially in RANSAC line merging—significantly affects performance for different distortions (e.g., curved, folded, incomplete, or rotated documents).

Key observations:

- Lower thresholds are effective when edges are well-defined (e.g., Curved).
- Moderate thresholds often strike the best balance for folds or perspective distortions.
- Higher thresholds can help with incomplete or heavily rotated pages where only partial edges are visible.

Overall, our pipeline is robust across various real-world document scenarios. Future enhancements, such as more adaptive edge filtering or fine-tuned homography refinement, could further improve accuracy and stability, especially for documents with extreme distortions.