Automated Document Scanning System

Shayan Shahrabi

March 2025

Abstract

This report presents an automated document scanning system that captures, rectifies, and enhances document images using computer vision techniques. A companion video demonstrates the complete workflow and real-world results. The pipeline includes edge detection, perspective correction, and optional binarization to produce scanner-like results from smartphone photos. The full source code and implementation details are publicly accessible in the project repository.

1 Introduction

Mobile document scanning has emerged as a ubiquitous capability, with commercial applications such as CamScanner, Adobe Scan, and Microsoft Lens providing robust solutions for perspective correction and document enhancement. This project implements a complete document scanning pipeline that replicates the core functionality of these commercial offerings through three primary stages:

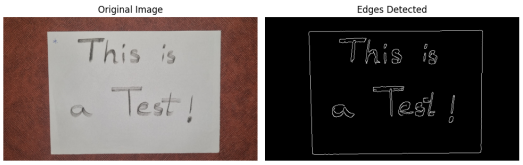
- **Edge Detection**: Equivalent to CamScanner's automatic border detection capability
- **Perspective Correction**: Analogous to Adobe Scan's quadrilateral transformation
- **Image Thresholding**: Comparable to Microsoft Lens's document binarization approach

2 Processing Pipeline

The system implements the following standardized processing stages:

1. **Border Detection**
   The edge detection process employs a multi-stage pipeline to identify document boundaries:
   (a) **Image Preprocessing**:
   - Input image loading and resizing to 20% of original dimensions
   - Color space conversion to grayscale
   - Noise reduction via Gaussian blur (5×5 kernel)

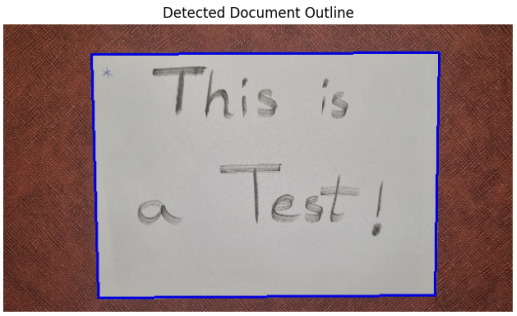
   (b) **Canny Edge Detection**:
   - Gradient calculation using Sobel operators
   - Non-maximum suppression
   - Hysteresis thresholding (50/200 thresholds)



The output demonstrates boundary detection through Canny's algorithm, with white pixels indicating strong edges and black representing background.

2. **Contour Processing**
   The system identifies and visualizes document boundaries through:
   (a) **Contour Rendering**:
   - Creates an image copy to preserve original data
   - Draws detected quadrilateral contour in blue (BGR: (225, 0, 0))
   - Uses 2px line thickness for optimal visibility

   (b) **Result Display**:
   - Converts color space from BGR to RGB for proper visualization
   - Renders output using Matplotlib for inline display
   - Removes axes and adds descriptive title for clarity



The visualization confirms boundary detection, with the blue contour representing the document's quadrilateral shape.

3. **Perspective Correction**
   The perspective transformation process converts detected document corners into a rectified image through:

(a) **Point Organization**:

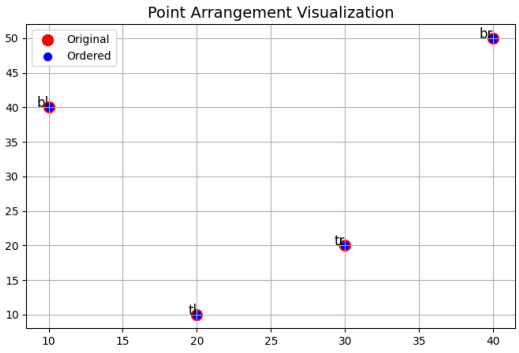
- Orders corners as (top-left, top-right, bottom-right, bottom-left)
- Calculates edge lengths using Euclidean distance

(b) **Dimension Calculation**:

- Determines output height from vertical edges
- Computes width from horizontal edges
- Preserves original aspect ratio

(c) **Homography Transformation**:

- Computes 3×3 perspective matrix
- Solves plane-to-plane projection
- Applies warp transformation



The output demonstrates successful rectification, transforming the detected quadrilateral into a rectangular document while maintaining original proportions.

3 Results and Output

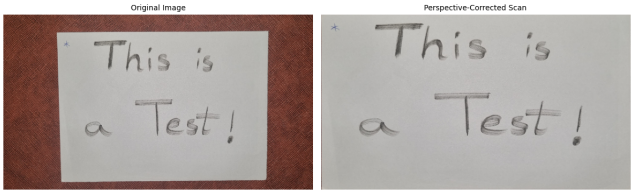


Figure 1: Comparative visualization of original (left) and perspective-corrected (right) document images

The system produces two types of output:

3.1 Standard Output

- Full-color corrected document
- Maintains original image quality
- Suitable for archiving and further processing

3.2 Enhanced Output

- Optional binarization for improved readability
- Adaptive thresholding with:
  - 11×11 pixel neighborhood
  - Gaussian weighting method
  - 10-unit intensity offset
- Ideal for OCR processing

4 Future Work

4.1 Text Recognition Enhancement

Integration of OCR capabilities would bridge the largest gap with commercial solutions:

- **Tesseract OCR**: Open-source engine for offline text extraction
- **Layout analysis**: Paragraph and column detection
- **Searchable PDF**: Generate output with text layers

4.2 Multi-Page Processing

Essential for practical document scanning:

- **Page stitching**: Combine multiple images into single document
- **Batch processing**: Automatic detection of document sets
- **Table of contents**: Generate from detected headings

4.3 User Experience Features

Quality-of-life improvements matching commercial apps:

- **Cloud sync**: Google Drive/Dropbox integration
- **Annotations**: Add text/markup to scanned documents
- **Quality presets**: Optimize for text/photo/mixed content

References

[1] Gonzalez, R. C. and Woods, R. E. (2008). *Digital Image Processing*. 3rd Edition. Prentice Hall.

[2] Olufemi, V. (2022). *Building Your Document Scanner with OpenCV*. Medium. `https://medium.com/@victorolufemi/build-a-document-scanner-with-opencv-ff9f645a4085` (accessed March 2025).

[3] Pourmoradi, N. (2024).
CamScanner with OpenCV.
Kaggle Notebook.
`https://www.kaggle.com/code/nimapourmoradi/camscanner-with-opencv`
(accessed March 2025).

[4] OpenCV Team. (2025).
OpenCV: Open Source Computer Vision Library.
`https://opencv.org/`
(accessed March 2025).

[5] Canny, J. (1986).
A computational approach to edge detection.
IEEE Transactions on Pattern Analysis and Machine Intelligence, 8(6), 679-698.

[6] Douglas, D. H. and Peucker, T. K. (1973).
Algorithms for the reduction of the number of points required to represent a digitized line or its caricature.
Cartographica: The International Journal for Geographic Information and Geovisualization, 10(2), 112-122.