Cam-Scanner using OpenCV

Soham Bhattacharya*
Ramakrishna Mission Vivekananda Educational and Research Institute

Abstract

This project presents a simple CamScanner-like document scanner using OpenCV. The system detects a document from an input image, extracts its contour, and applies a perspective transformation to generate a top-down, flattened "scanned" view, effectively mimicking traditional document scanning using only image processing techniques.

1 Introduction

In today's digital world, mobile document scanning has become a common necessity. Traditional scanners are being replaced by computer vision-based solutions that can detect and process documents using just a camera. Applications like CamScanner demonstrate the power of image processing to convert photos of documents into clear, readable scans. This project aims to replicate the core functionality of such applications using Python and OpenCV. It enables automatic document detection, perspective correction, and enhancement to produce a clean, top-down view of the input document.

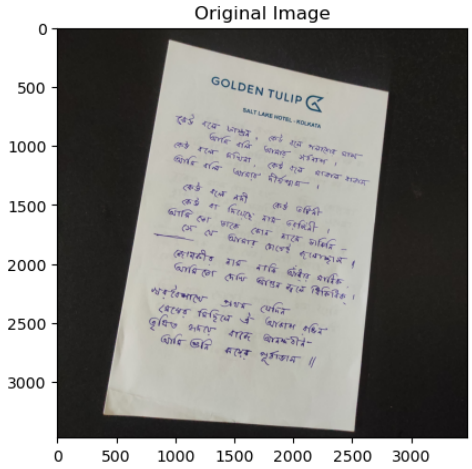

Figure 1: *Reference image [an image of handwritten lyrics on a notepad placed on my study table]*

The implemented method follows a series of well-defined computer vision steps to convert a photograph of a document into a scan-like output:

1. **Preprocessing**: The input image is resized and converted to grayscale, followed by Gaussian blurring to reduce noise.
2. **Edge Detection**: Canny edge detection is applied to highlight boundaries of the document within the image.
3. **Contour Detection**: The largest quadrilateral contour, assumed to be the document, is identified and approximated using contour approximation techniques.
4. **Perspective Transformation**: Once the document corners are found, a four-point perspective transform is applied to warp the document into a top-down view.

*e-mail:bhattacharyasoham026@gmail.com

2 Method Overview

In this section, the final methods that have been implemented are elaborated in short. There are a number of different ways to do this, but this is the usual standard method opted.

1. **Preprocessing**
   Prepare the image for reliable edge and contour detection.
   - **Resizing**: Standardizes the image size to reduce processing time and maintain consistency.
   - **Grayscale Conversion**: Converts the input image from BGR to grayscale using cv2.cvtColor(). This simplifies the image and reduces computational complexity.
   - **Gaussian Blurring**: Applies a Gaussian blur using cv2.GaussianBlur() to reduce noise. This helps in stabilizing edge detection by smoothening sharp variations in pixel intensity.
     K = $\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$ is an example of a gaussian kernel of size 3x3. The 2D Gaussian function is

$$G(x,y) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

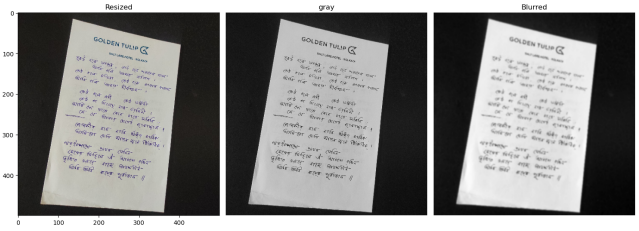

Figure 2: *Preprocessing image results*

2. **Edge Detection** Objective is to detect significant edges in the image, especially the boundaries of the document.
   Process involved is Canny Edge detection with the following steps:
   - **Step 1**: Reduces noise using Gaussian blur.
   - **Step 2**: Gradient calculation using Sobel operator. An example of a Sobel operator for x-directional gradient is $\frac{1}{8}\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix}$ which is applied as a filter on the image Im, that renders a resultant image say I. The image gradient is calculated using this Sobel operator in the x and y directions:

$$G_x = \frac{\partial I}{\partial x}, \quad G_y = \frac{\partial I}{\partial y}$$

The gradient magnitude and direction (angle) at each pixel are computed as:

$$G = \sqrt{G_x^2 + G_y^2}, \quad \theta = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

- **Step 3**: Non maximum Suppression
- **Step 4**: Double Thresholding $(high, low)$
  - Pixels with $grad\_value > high$ are a part of *Strong Edges*
  - Pixels with $grad\_value \in (low, high)$ are a part of *Weak Edges*
  - Pixels with $grad\_value < low$ are a part of *discared Edges*
- **Step 5**: Hysteresis
  Those weak edges that are connected to strong edges are retained.
- **Step 6**: Binary Image Coloring
  Pixels that satisfy the above criteria are given value 255 (white) and remaining are assigned as 0 (black).

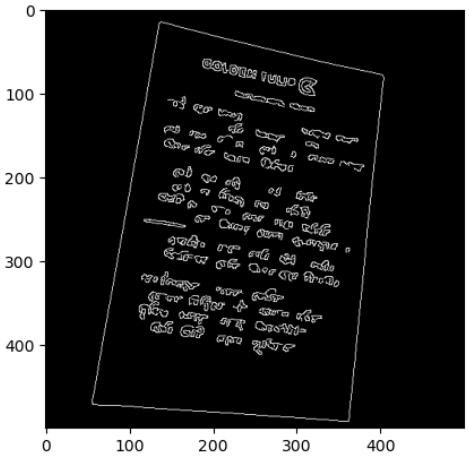

Figure 3: *Canny-Edge Detection image results*

3. **Contour Detection**: Objective is to identify and isolate the boundary of the document.
   Processes involved:
   - **Contour Extraction**: Uses cv2.findContours() to retrieve the contours from the edge-detected image.
   - **Sorting and Filtering**: Sorts the contours based on area, assuming the document is the largest visible quadrilateral.
   - **Contour Approximation**: Uses cv2.approxPolyDP() to approximate each contour to a polygon. The approximation factor (epsilon) is based on the perimeter of the contour.
   - **Quadrilateral Detection**: Selects the first polygon with four points (indicative of a document). This is considered the document boundary.

Given a contour with perimeter P, the approximation accuracy ϵ is defined as:

$$\epsilon = \alpha P, \quad \text{where } \alpha \in [0.01, 0.1]$$

The approximation process finds a polygonal curve such that:

$$\text{distance between contour and approximated polygon} < \epsilon$$

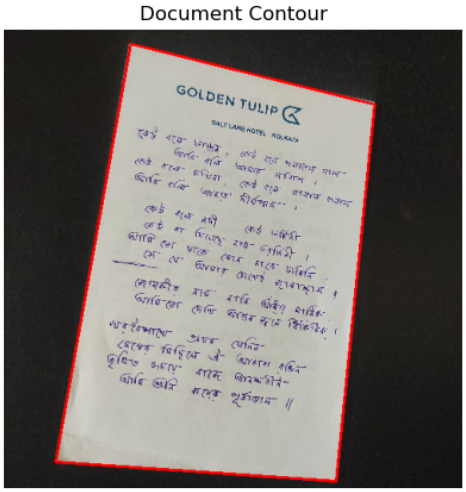

Figure 4: *Contour Detection image results*

4. **Perspective Transformation**: Objective is to warp the detected quadrilateral to produce a top-down "scanned" view of the document. Processes Involved:
   - **Corner Ordering:** The four detected points are ordered as top-left, top-right, bottom-right, and bottom-left. This ordering is determined based on the sum and difference of their (x, y) coordinates.
   - **Destination Mapping:** The document is mapped to a new rectangle of desired width and height. This is done using the OpenCV function `cv2.getPerspectiveTransform()`, which computes a 3×3 transformation matrix.
   - **Warping:** The input image is warped using `cv2.warpPerspective()`, which applies the transformation matrix to project the original quadrilateral onto a flat, rectangular surface. This step corrects for skew, tilt, and perspective distortions.

To map points from the source plane (x, y) to the destination plane (x', y'), a homography matrix $\mathbf{H}$ is used:

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

To retrieve the actual 2D coordinates from homogeneous coordinates:

$$x' = \frac{h_1^T X}{h_3^T X}, \quad y' = \frac{h_2^T X}{h_3^T X}$$

Where:

- h_1^T, h_2^T, h_3^T are the rows of the homography matrix $\mathbf{H}$
- $X = \begin{bmatrix} x & y & 1 \end{bmatrix}^T$ is the source point in homogeneous coordinates

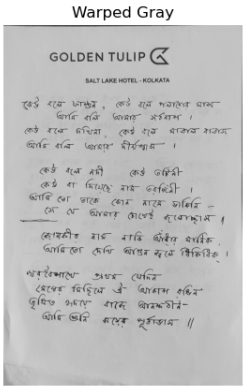

Figure 5: *Perspective Transformation image results*

3 Results

The proposed CamScanner-like system was tested on various images of documents captured under different lighting and perspective conditions. The results indicate successful detection of document boundaries and accurate perspective transformation, producing top-down, flat representations of the documents.

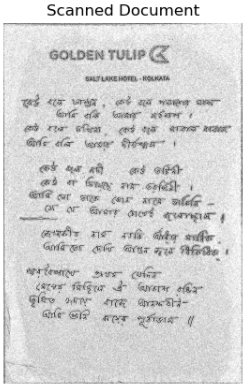

Figure 6: *Scanned output*

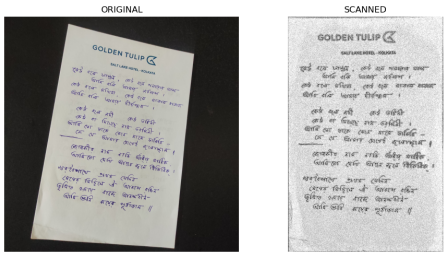

Figure 7: *Comparison between original capture and scanned result.*