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

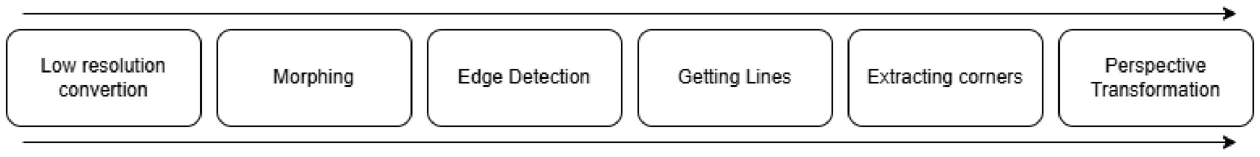
& **Shekhar Borile**

Topic: Image Perspective Transformer and Cropper

Aim: To design and implement an automatic image processing system that detects, crops, and corrects the perspective distortion (unskewing) of document images captured using a camera, thereby producing a clean and properly aligned output similar to the functionality provided by Microsoft Lens.

Theory :

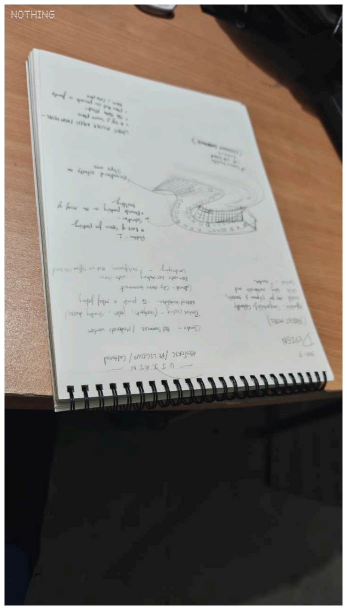
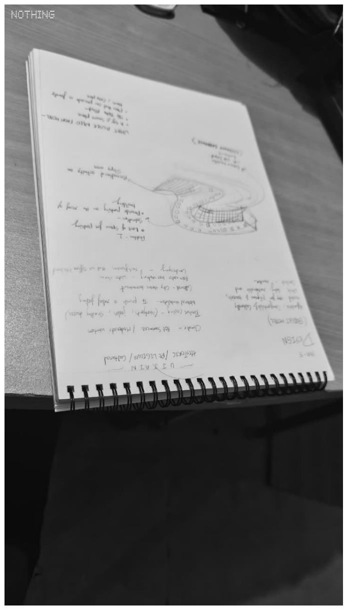
The image transformation will be done through this pipeline:



Low-Resolution and Greyscale Conversion

Large image files (e.g., > *4 MB*) take significantly more time to process due to the high number of pixels. To enhance speed and efficiency, the input image is resized while maintaining aspect ratio. In this project, the width is reduced to **300 pixels** for faster computation. Also converting it into rgb into bw as it simplify further preprocessing and edge detection

- **Reason for Downscaling**:
    - Faster edge detection and line extraction
    - Reduced memory usage during Hough Transform
    - Overall decrease in processing time from ~20–30 seconds to ~2–3 seconds
    - Grayscale conversion enhances contrast and simplifies further processing

Morphological Operations :

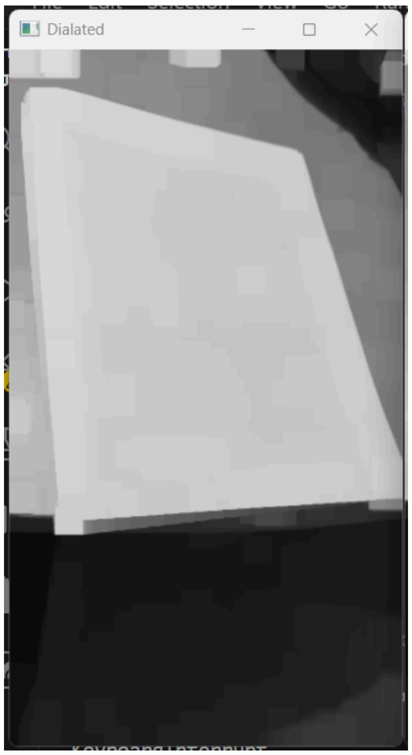
Morphological operations help in smoothing the image and enhancing its structural elements by removing noise and filling small gaps. These operations are applied after converting the image to grayscale.

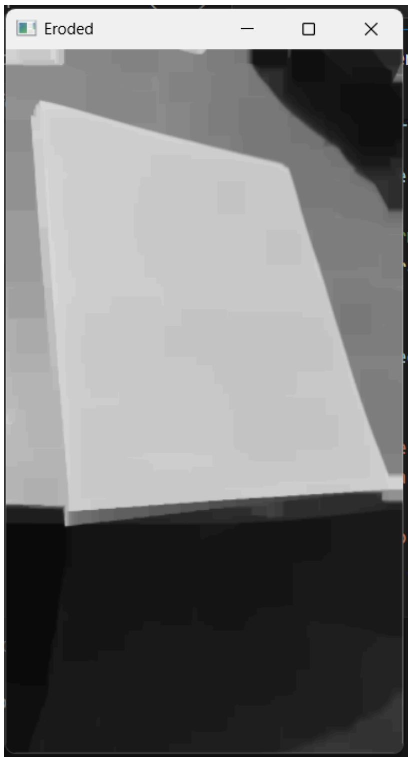
- **Dilation**:
  Expands the boundaries of objects (white pixels), which helps connect broken parts of the document's edges.

- **Erosion**:
  Shrinks the boundaries, effectively removing small noise blobs and ensuring only strong edges remain after dilation.

Together, dilation followed by erosion forms **Morphological Closing**, which helps in:

- Filling small holes inside the document border
- Bridging small gaps in edges
- Making contour and line detection more accurate



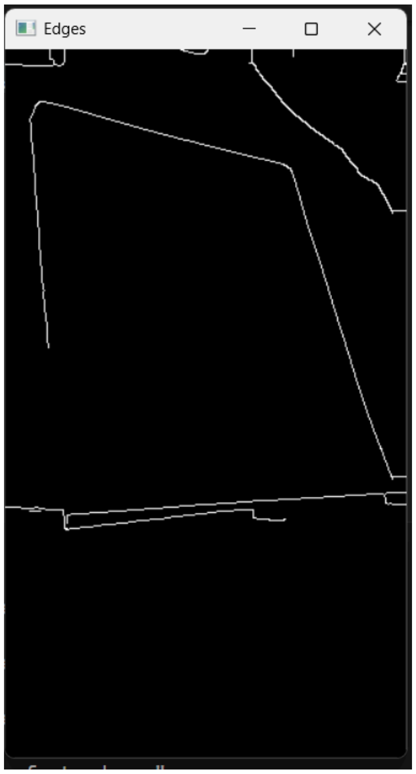


Edge Detection:

Once the morphological operations are applied, the image is processed using the Canny Edge Detection algorithm. This method detects sharp changes in image intensity and isolates strong edges — ideal for detecting document borders.

Key steps in Canny:

- Noise reduction (using Gaussian blur)
- Intensity gradient calculation
- Edge tracking with thresholding



Hough Line Transform :

Based on the detected edges, the Hough Transform is used to find straight line segments that could correspond to the document's boundary.

- This project uses a vectorized Hough Transform for speed.

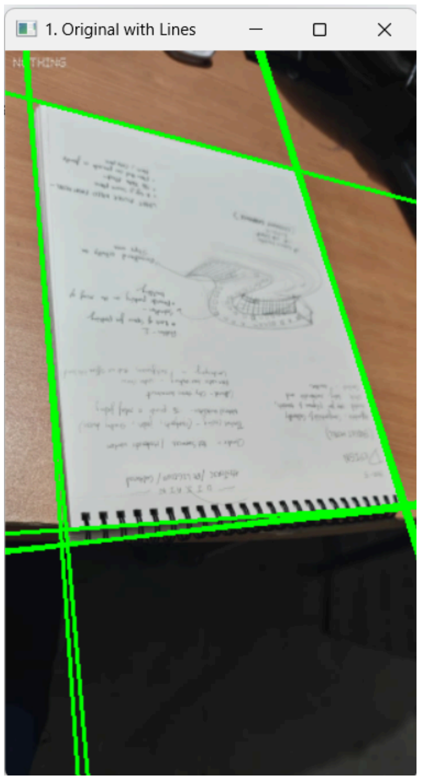
- The algorithm converts points in Cartesian space into sinusoidal curves in Hough space.

The Medium part of this project has gone through this part's internal implementation in detail.

In summary it does the following steps internally :

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.

In implementation of code, we have taken threshold limit to be **90** as before it detects to many background unnecessary line



Line Classification and Corner Detection :

The lines detected via Hough Transform are classified into horizontal and vertical lines based on their angle:

- Horizontal lines have angles close to 0°.
- Vertical lines have angles close to ±90°.

By selecting two top/bottom horizontal lines and two left/right vertical lines, the intersections of these lines are computed to obtain four corner points — the vertices of the document.

This is a three step process:

Line Classification (Horizontal vs Vertical) :

To classify the lines based on their orientation:

- If the **angle (theta)** is close to **±90°**, the line is nearly vertical.
- If the angle is close to **0° or 180°**, the line is horizontal.

Selecting Relevant Lines

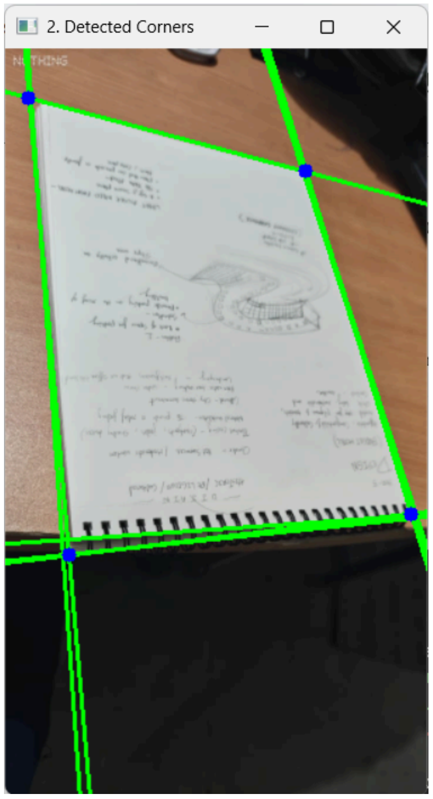
To define the document boundary, we only need:

- The **top-most** and **bottom-most** horizontal lines
- The **left-most** and **right-most** vertical lines

Finding Corner Points

With the 4 boundary lines identified, we compute the intersection points to get the 4 corners using the basic equation of lines in polar form:

- Each line is converted into its parametric form
- Intersection is found by solving a system of linear equations using NumPy's `linalg.solve()`



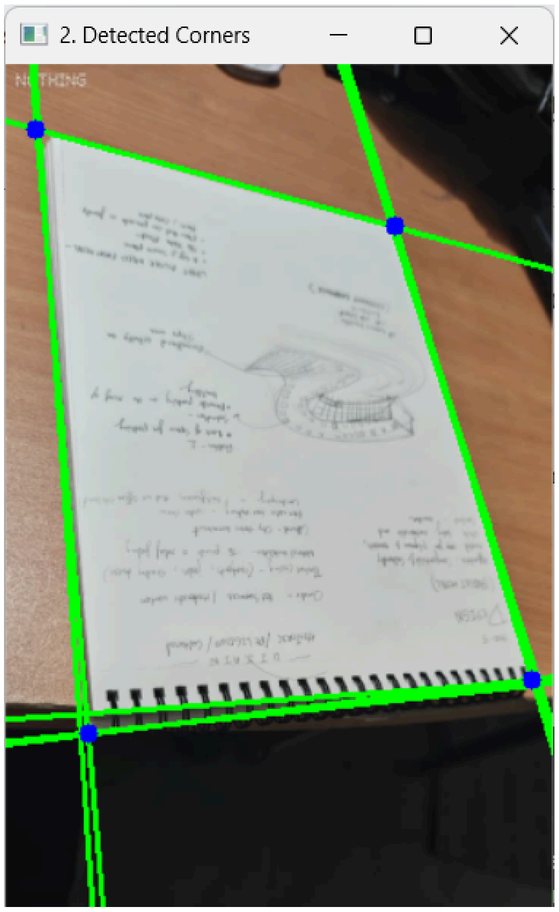
Perspective Transformation (Unskewing)

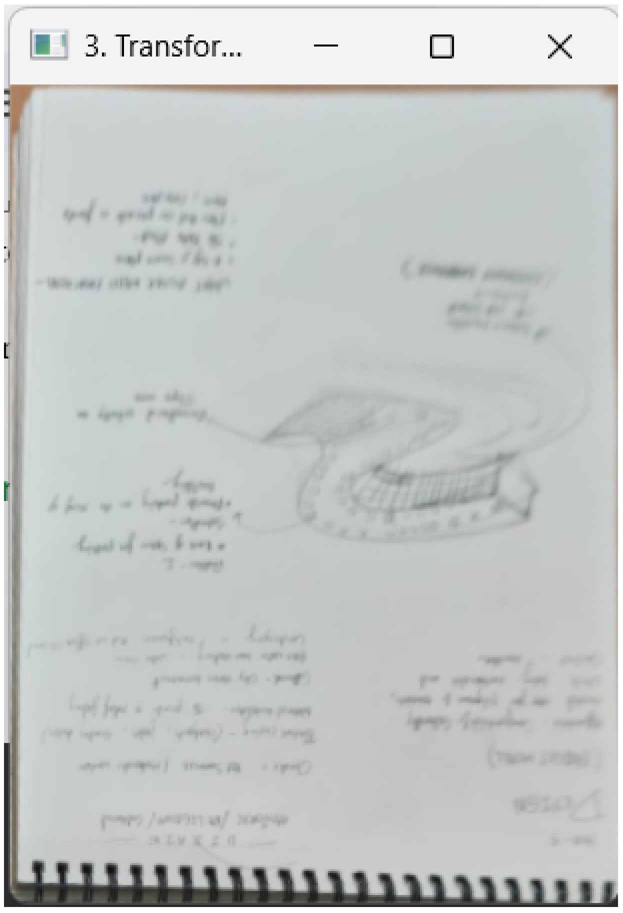
After the four corners are identified, the document is cropped and unskewed using a **Perspective Transform**:

- Maps the quadrilateral region formed by the corners into a rectangle
- Eliminates perspective distortion (like taking a picture of a document at an angle)
- Achieves a clean, top-down "scanned" view of the document

The Medium part of this project has gone through this part in detail.

OpenCV's `getPerspectiveTransform()` and `warpPerspective()` functions are used for this task.





Code:

```
import cv2 as cv
import numpy as np

def getLines(img, threshold, angle_step=1):
    """hough line using vectorized numpy operations,
    may take more memory, but takes much less time"""
    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)

    # Cache some resuable values
    cos_theta = np.cos(thetas)
    sin_theta = np.sin(thetas)
    num_thetas = len(thetas)

    # Hough accumulator array of theta vs rho
    accumulator = np.zeros((2 * diag_len, num_thetas))
    y_idxs, x_idxs = np.nonzero(img)  # (row, col) indexes to edges
    # Vote in the hough accumulator
    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)
    for i in range(num_thetas):
        _rhos,counts = np.unique(rhosmat[:,i], return_counts=True)
        accumulator[_rhos,i] = counts
    # Thresholding
    idxs = np.argwhere(accumulator > threshold)
    rho_idxs, theta_idxs = idxs[:,0], idxs[:,1]
    return np.column_stack((rhos[rho_idxs], thetas[theta_idxs]))


def line_intersection(line1, line2):
    """Find intersection point of two lines defined by (rho, theta)"""
    rho1, theta1 = line1
    rho2, theta2 = line2

    A = np.array([
        [np.cos(theta1), np.sin(theta1)],
        [np.cos(theta2), np.sin(theta2)]
```

```python
    ])
    b = np.array([[rho1], [rho2]])

    try:
        x0, y0 = np.linalg.solve(A, b)
        return int(x0[0]), int(y0[0])
    except np.linalg.LinAlgError:
        return None


def get_corners_from_lines(lines, img_shape):
    """Extract 4 corner points from detected lines"""
    if len(lines) < 4:
        return None

    # Separate lines into horizontal and vertical based on angle
    horizontal = []
    vertical = []

    for rho, theta in lines:
        angle_deg = np.rad2deg(theta)
        # Vertical lines: close to -90° or 90°
        if abs(angle_deg) > 45:
            vertical.append((rho, theta))
        else:
            horizontal.append((rho, theta))

    if len(horizontal) < 2 or len(vertical) < 2:
        return None

    # Sort to get top/bottom horizontal and left/right vertical
    horizontal.sort(key=lambda x: x[0])
    vertical.sort(key=lambda x: x[0])

    # Get the two most extreme lines in each direction
    h1, h2 = horizontal[0], horizontal[-1]
    v1, v2 = vertical[0], vertical[-1]

    # Find 4 corner intersections
    corners = []
    for h in [h1, h2]:
        for v in [v1, v2]:
            corner = line_intersection(h, v)
            if corner:
                corners.append(corner)
    if len(corners) != 4:
        return None

    # Sort corners: top-left, top-right, bottom-right, bottom-left
    corners = sorted(corners, key=lambda x: (x[1], x[0]))  # Sort by y, then x

    if len(corners) == 4:
        top_pts = sorted(corners[:2], key=lambda x: x[0])  # Top 2, sort by x
        bottom_pts = sorted(corners[2:], key=lambda x: x[0])  # Bottom 2, sort
by x
        return np.array([top_pts[0], top_pts[1], bottom_pts[1], bottom_pts[0]],
dtype=np.float32)

    return None


def perspective_transform(img, corners):
    """Apply perspective transformation to get bird's eye view"""
    # Calculate width and height of the output image
    width_top = np.linalg.norm(corners[0] - corners[1])
    width_bottom = np.linalg.norm(corners[3] - corners[2])
    width = int(max(width_top, width_bottom))

    height_left = np.linalg.norm(corners[0] - corners[3])
    height_right = np.linalg.norm(corners[1] - corners[2])
    height = int(max(height_left, height_right))

    # Destination points (rectangle)
    dst = np.array([
        [0, 0],
        [width - 1, 0],
        [width - 1, height - 1],
        [0, height - 1]
    ], dtype=np.float32)

    # Get perspective transformation matrix
    M = cv.getPerspectiveTransform(corners, dst)

    # Apply transformation
    warped = cv.warpPerspective(img, M, (width, height))
    return warped


# Main execution
```

Learning:

- Downscaled image processing: Resizing to 300 px width significantly improved performance (20–30s → 2–3s).
- 5x dilation + erosion: Best method to enhance document boundaries without retaining inner handwritten/text details.
- Canny edge detection: Ideal algorithm for detecting clear document edges with strong noise suppression.
- .From practical trial and error among various images the canny edge detection threshold value of 60 to 180 is the most appropriate value.
- From practical trial and error among various images the hough line threshold value of 90 is the most appropriate value.

Improved understanding of:

- Preprocessing and removing uneccesaay details from image
- Hough Transform for line detection
- Perspective correction using OpenCV
- Corner extraction using vectorized computation

Conclusion :

This project successfully demonstrates how to automate document extraction and perspective correction from an image using image processing. The approach replicates functionality similar to commercial document scanning apps and can be extended with contour-based detection and adaptive thresholding for more robust performance.

References:

- “DIP easy.pdf” part of this project
- “DIP medium.pdf” part of this project
- https://oezeb.github.io/hough-transform/
- https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html
- https://en.m.wikipedia.org/wiki/Hough_transform
- https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html
- https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html

Final Working Code:

- **Rounit Kashyap’s github repository :** https://github.com/Rounit-1st/Digital-Image-Processing-Text-Extraction-Minor-Project