Document Scanner and Perspective Correction
ECE501 - Digital Image Processing
Group-1

Members:

Name	Enrolment No.
Dhruvik Shah	AU2340113
Ayush Makwana	AU2340138
Divyam Thakkar	AU2340227
Jainik Patel	AU2340042

1. Objective for the Week (11/10/25 - 17/10/25 and 25/10/2025 - 1/11/2025)

- Detect and isolate the main document region from input images.
- Implement edge detection and contour analysis using OpenCV.
- Identify the largest quadrilateral contour representing the document.
- Save intermediate outputs for further processing in the next phase.

2. Tasks Completed:

- Images were loaded and preprocessed using grayscale and gaussian blur.
- Applied Canny edge detection to extract document boundaries.
- Used **cv2.findContours()** to identify all contours in the image.
- Filtered and arranged contours by area to identify the largest quadrilateral (document area).
- Drew identified edges on the original image for visual confirmation.
- Output images were saved, displaying identified document edges

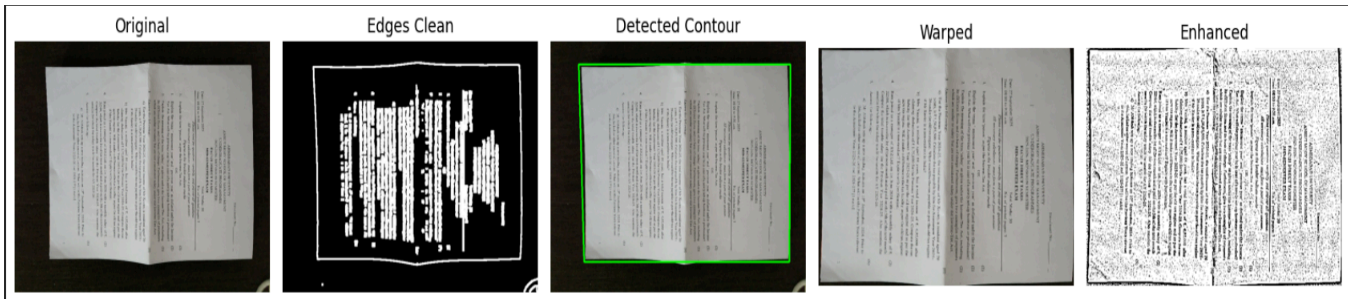
Python
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
import os # Import os to check for file existence

# --- Step 1: Load originals and Week 2 outputs ---
try:
    output_prefix = globals().get('output_prefix', None)
    if output_prefix is None:
         raise NameError("output_prefix not found. Run Week 2 cell first.")

    img_filename = None
    uploaded_files = globals().get('uploaded', {})
    if uploaded_files:
        img_filename = next(iter(uploaded_files))
        if img_filename.rsplit('.', 1)[0] != output_prefix:
             print(f"Warning: Uploaded filename prefix '{img_filename.rsplit('.',
1)[0]}' does not match output_prefix '{output_prefix}'. Using uploaded filename.")
             output_prefix = img_filename.rsplit('.', 1)[0]


    if img_filename is None:
         potential_img_files = [f for f in os.listdir('.') if
f.startswith(output_prefix)]
         if potential_img_files:
             img_filename = potential_img_files[0]
         else:
              raise FileNotFoundError(f"Could not find original image file starting
with '{output_prefix}'.")


    img = cv2.imread(img_filename)

    if img is None:
        raise FileNotFoundError(f"Could not read image from '{img_filename}'. Make
sure it's a valid image file.")

except NameError as e:
    print(f"Error: {e}")
    print("Please run the 'Week 2: Image Preprocessing & Edge Detection' cell first to
generate the necessary outputs.")
    raise # Re-raise the exception to stop execution

except FileNotFoundError as e:
    print(f"Error: {e}")
    print(f"Could not find the necessary output files or the original image file.
Please ensure '{output_prefix}_output_gray_clahe.jpg',
'{output_prefix}_output_edges_clean.jpg' (or fallbacks) and the original image file
'{img_filename}' exist after running Week 2.")
    raise # Re-raise the exception to stop execution


# Use the exact files Week 2 saved
gray = cv2.imread(f'{output_prefix}_output_gray_clahe.jpg', cv2.IMREAD_GRAYSCALE)
edges_clean = cv2.imread(f'{output_prefix}_output_edges_clean.jpg',
cv2.IMREAD_GRAYSCALE)

# fallback if CLAHE file not found
if gray is None:
    gray = cv2.imread(f'{output_prefix}_output_gray.jpg', cv2.IMREAD_GRAYSCALE)
if edges_clean is None:
    edges_clean = cv2.imread(f'{output_prefix}_output_edges.jpg',
cv2.IMREAD_GRAYSCALE)

if gray is None or edges_clean is None:
    raise FileNotFoundError(f"Week 2 outputs not found
({output_prefix}_output_gray_clahe.jpg or {output_prefix}_output_edges_clean.jpg). Run
Week 2 cell first.")

# Resize original to same size as Week 2
# Get dimensions from one of the processed images as they should be consistent
if gray is not None:
    scale_height, scale_width = gray.shape[:2]
    img = cv2.resize(img, (scale_width, scale_height))
else: # Fallback if gray is None, although the check above should prevent this
    scale_width = 800 # Default size if processed images not found
    scale_height = int(img.shape[0] * scale_width / img.shape[1])
    img = cv2.resize(img, (scale_width, scale_height))


edges_clean = cv2.resize(edges_clean, (scale_width, scale_height))
gray = cv2.resize(gray, (scale_width, scale_height))

# --- Step 2: Contour detection (robust) ---
contours, _ = cv2.findContours(edges_clean, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
contours = sorted(contours, key=cv2.contourArea, reverse=True)

doc_contour = None
img_h, img_w = img.shape[:2]
img_area = img_h * img_w

for c in contours:
    peri = cv2.arcLength(c, True)
```

```
        approx = cv2.approxPolyDP(c, 0.02 * peri, True)
        area = cv2.contourArea(approx)

        # skip tiny shapes
        if area < 0.05 * img_area:
            continue

        # Skip if contour touches image border (likely frame)
        x,y,w,h = cv2.boundingRect(approx)
        if x <= 5 or y <= 5 or x + w >= img_w - 5 or y + h >= img_h - 5:
            continue

        # prefer quads that occupy reasonable area
        if len(approx) == 4 and 0.15 * img_area < area < 0.95 * img_area:
            # optional aspect filter
            ar = w / float(h) if h>0 else 1.0
            if 0.4 < ar < 2.5:
                doc_contour = approx
                break

    # --- Step 3: Fallback strategies if not found ---
    if doc_contour is None:
        # try relaxing border rule: accept largest quad (even if near border)
        for c in contours:
            peri = cv2.arcLength(c, True)
            approx = cv2.approxPolyDP(c, 0.02 * peri, True)
            area = cv2.contourArea(approx)
            if len(approx) == 4 and area > 0.15 * img_area:
                doc_contour = approx
                print("Fallback: using largest quad contour (near border).")
                break

    if doc_contour is None and len(contours) > 0:
        # last resort: use bounding rect of largest contour
        x,y,w,h = cv2.boundingRect(contours[0])
        doc_contour = np.array([[[x,y]],[[x+w,y]],[[x+w,y+h]],[[x,y+h]]], dtype=np.int32)
        print("Final fallback: using bounding box of largest contour.")

    # --- Step 4: Visualize selected contour on original ---
    contour_vis = img.copy()
    if doc_contour is not None:
        cv2.drawContours(contour_vis, [doc_contour], -1, (0,255,0), 3)
        print("Document contour selected.")
    else:
        print("No meaningful contour could be selected.")

    # --- Step 5: Perspective transform (if doc_contour exists) ---
    if doc_contour is not None:
        pts = doc_contour.reshape(4,2).astype(np.float32)

        # order points tl, tr, br, bl
        s = pts.sum(axis=1)
        rect = np.zeros((4,2), dtype="float32")
        rect[0] = pts[np.argmin(s)]
        rect[2] = pts[np.argmax(s)]
        diff = np.diff(pts, axis=1)
```

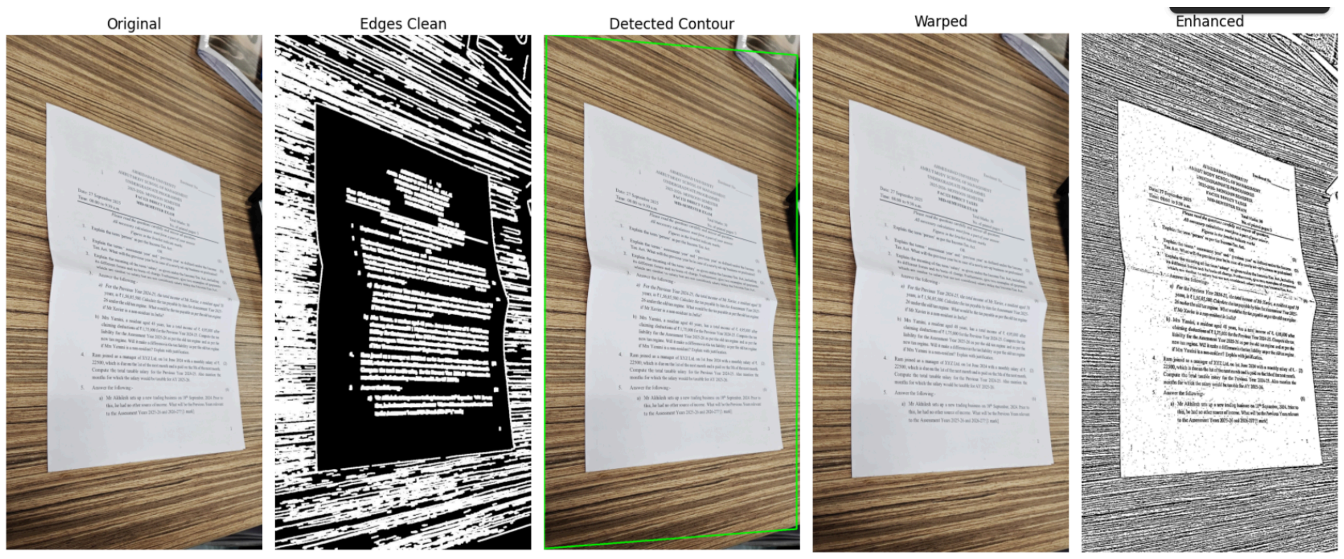
```python
    rect[1] = pts[np.argmin(diff)]
    rect[3] = pts[np.argmax(diff)]

    (tl, tr, br, bl) = rect
    widthA = np.linalg.norm(br - bl)
    widthB = np.linalg.norm(tr - tl)
    heightA = np.linalg.norm(tr - br)
    heightB = np.linalg.norm(tl - bl)
    maxW = int(max(1, round(max(widthA, widthB))))
    maxH = int(max(1, round(max(heightA, heightB))))
    dst = np.array([[0,0],[maxW-1,0],[maxW-1,maxH-1],[0,maxH-1]], dtype="float32")

    # avoid degenerate sizes
    if maxW < 10 or maxH < 10:
        warped = img.copy()
        print("Warp size too small, skipping warp.")
    else:
        M = cv2.getPerspectiveTransform(rect, dst)
        warped = cv2.warpPerspective(img, M, (maxW, maxH))
else:
    warped = img.copy()

# --- Step 6: Enhance final scanned output ---
warped_gray = cv2.cvtColor(warped, cv2.COLOR_BGR2GRAY)
# choose adaptive params robustly (block size must be odd and <= image dim)
block = 11
if block >= warped_gray.shape[0]:
    block = warped_gray.shape[0] - 1
if block % 2 == 0:
    block += 1
enhanced = cv2.adaptiveThreshold(warped_gray, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                 cv2.THRESH_BINARY, block, 2)

# --- Step 7: Display outputs ---
titles = ['Original','Edges Clean','Detected Contour','Warped','Enhanced']
imgs = [img, edges_clean, contour_vis, warped, enhanced]

plt.figure(figsize=(16,8))
for i in range(len(imgs)):
    plt.subplot(1,5,i+1)
    im = imgs[i]
    if im is None:
        plt.text(0.5,0.5,'None',ha='center')
    elif len(im.shape)==2:
        plt.imshow(im, cmap='gray')
    else:
        plt.imshow(cv2.cvtColor(im, cv2.COLOR_BGR2RGB))
    plt.title(titles[i])
    plt.axis('off')
plt.tight_layout()
plt.show()

# --- Step 8: Save Week 3 outputs ---
cv2.imwrite(f"{output_prefix}_output_contour.jpg", contour_vis)
cv2.imwrite(f"{output_prefix}_output_warped.jpg", warped)
cv2.imwrite(f"{output_prefix}_output_enhanced.jpg", enhanced)
```

3. Observations and Results

- Document contours were successfully detected for most test images.
- Detection worked well for images with clear edges and good lighting.
- Some low-contrast or white-background images needed parameter tuning in Canny edge detection.
- Visual results clearly highlighted the detected quadrilateral region around the document
- Low lighting



- Well lit



4. Next Week's Plan

- Use detected contours to continue on **perspective correction**.
- Apply **adaptive thresholding** and **contrast enhancement** to get scanner-like output.
- Save final transformed outputs for evaluation.

5. References

- OpenCV Documentation: https://docs.opencv.org/
- OpenCV-Python Tutorials: Youtube and other sources.
- Youtube: https://youtu.be/q-gWjCgigTg?si=vk71qO7aJkUDmj7O
- Research Paper: Perspective Correction of Document Images – IIIT Hyderabad