**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: Detecting Major Edges of an Object (Specifically Documents)

**Aim:** To detect and highlight the major edges of a document in an image using preprocessing techniques such as morphological operations and edge detection algorithms in OpenCV.

### Objectives :

- To study the basics of edge detection in image processing.
- To understand the use of morphological transformations like dilation and erosion.
- To apply the Canny Edge Detection algorithm to extract meaningful edges.
- To develop a Python program using OpenCV that processes an input document image and outputs prominent edges.

### Theory :

#### Edge Detection

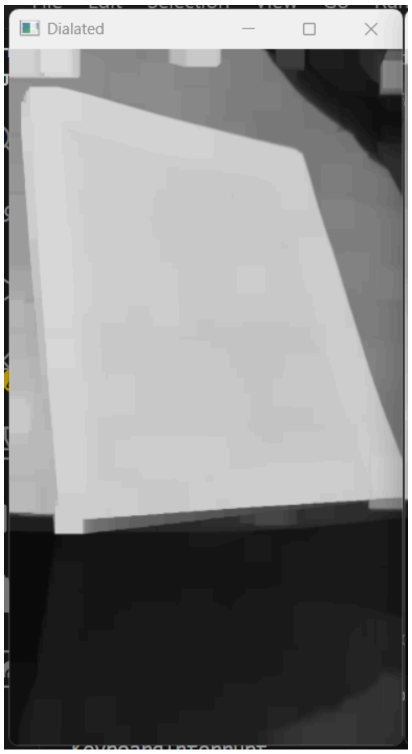
Edge detection is a technique used to identify points in an image where the brightness changes sharply—typically corresponding to object boundaries. It's widely used in applications such as object detection, image segmentation, and computer vision pipelines.

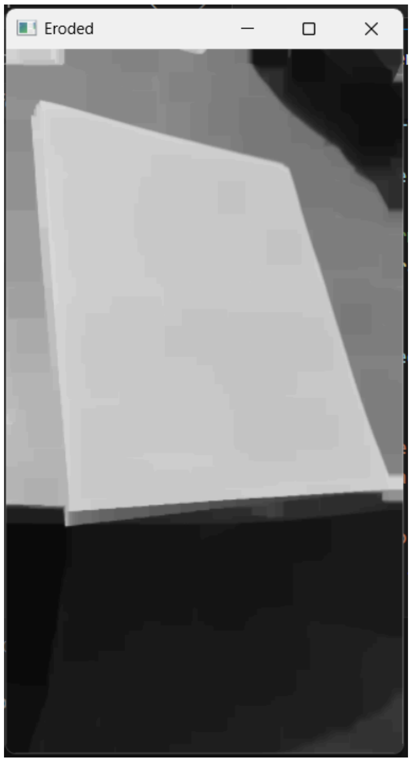
#### Morphological Operations

Morphological transformations are operations based on image shape used to simplify, remove noise, and extract essential structures. Two important operations are:

- **Dilation:**
  - **Purpose**: Expands the white (foreground) regions of an image.
  - **How it works**: For each pixel in the image, dilation takes the **maximum value** within the neighborhood defined by the structuring element.
  - **Application in this project**: Helps to **connect broken components** of document edges and emphasize the boundary by thickening lines.

- **Erosion:**
    - **Purpose**: Shrinks the white (foreground) regions of an image.
    - **How it works**: Erosion takes the **minimum value** within the structuring element's neighborhood.
    - **Application in this project**: Helps **remove small noise** and isolates the most significant parts of the image after dilation.





These operations help strengthen the document boundary and remove small irregularities.

### Canny Edge Detector

Canny is a multi-stage edge detection algorithm that provides good detection, localization, and noise robustness. It works in six main steps: Noise Reduction, Gradient Calculation, Non-Maximum Suppression, Double Thresholding, and Edge Tracking by Hysteresis.

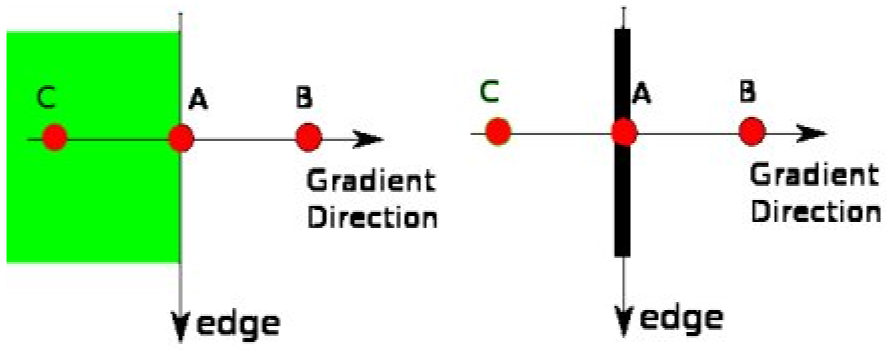
The Canny algorithm follows these main steps:

#### Noise Reduction

Edge detection is sensitive to noise, so the input image is first smoothed using a **Gaussian filter**. This reduces small variations and avoids false edge detection.

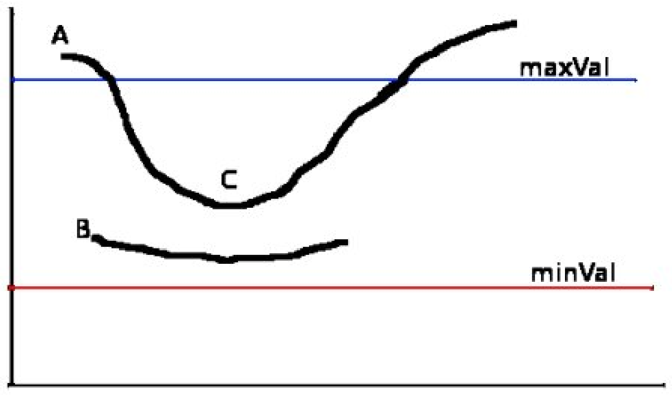
1. **Gradient Calculation**
 The smoothed image is processed using **Sobel operators** to calculate the **gradient magnitude** and **direction** at each pixel. These gradients highlight areas with sharp intensity change—potential edges.

2. **Non-Maximum Suppression**
 To make edges thinner and more precise, pixels that are not part of the local maxima in the gradient direction are removed. This step ensures only the most significant edge points are retained.

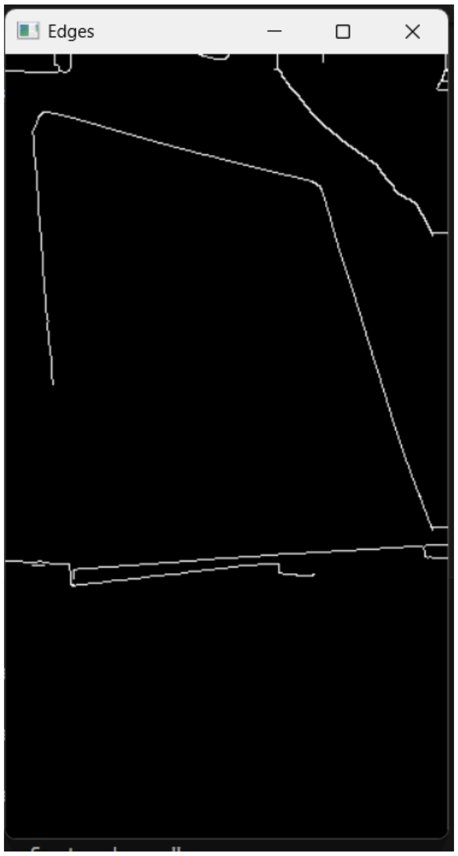


3. **Double Thresholding**
 Two threshold values are used to classify gradient pixels:

   - **Strong edges**: Gradient values above the high threshold.
   - **Weak edges**: Between low and high thresholds.
   - **Discarded**: Below the low threshold.



4. **Edge Tracking by Hysteresis**
   Weak edges are only included in the final edge map if they are connected to strong edges. Isolated weak edges are discarded, reducing noise and ensuring continuity.



**Methodology**

1. Load the input image and convert it to grayscale.
2. Apply dilation to connect weak edges.
3. Apply erosion to remove noise.
4. Use the Canny Edge Detector to extract the main edges.

Display or save the processed image.

**The flow diagram:**

`Input Image → Grayscale → Dilation → Erosion → Canny Edge Detection → Output Edges`

**Code Implementation :**

```python
import cv2 as cv
import numpy as np

# Load the image in grayscale
img = cv.imread('document.jpg', cv.IMREAD_GRAYSCALE)
assert img is not None, "Image not found!"

# Step 1: Create a kernel for morphological operations
kernel = np.ones((5,5), np.uint8)

# Step 2: Dilate the image
dilated = cv.dilate(img, kernel, iterations=5)

# Step 3: Erode the image
eroded = cv.erode(dilated, kernel, iterations=5)

# Step 4: Apply the Canny Edge Detector
edges = cv.Canny(eroded, 60, 180)

# Step 5: Show the result
cv.imshow('Original Image', img)
cv.imshow('Edges Detected', edges)
cv.waitKey(0)
cv.destroyAllWindows()
```

**Results**

- The original image is processed using morphological operations to enhance the document's boundary.
- Using the Canny Edge Detector, clear and sharp edges of the document are detected.
- The output image highlights the border of the document, suitable for further applications like document scanning or perspective correction.

**Learning :**

- Downscaling the image to 300px width reduced processing time from ~25s to ~3s while still preserving key document features.

- Applying 5 iterations of dilation followed by erosion gave the best edge clarity, without keeping unnecessary text.

- Canny Edge Detection worked best because it accurately detects strong edges, suppresses noise, and produces clean, continuous boundaries ideal for document edge extraction.

**Conclusion :**

This project successfully demonstrates the use of morphological operations and Canny Edge Detection for the purpose of document edge detection. The implemented method is efficient and forms the foundation for advanced document processing applications such as scanning, cropping, or OCR systems.

**Applications :**

- Document scanning apps

- Automatic image cropping and perspective correction

- Preprocessing for OCR (Optical Character Recognition)

- Object boundary detection in robotics and computer vision

**References**

- **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**