كلية العلوم

قسم الانظمة الطبية الذكية

Lec 5 & 6

Edge Detection

Subject: Computer Vision

Level: third

Lecturer: Asst. Lecturer Qusai AL-Durrah

Learning Outcomes

By the end of this lecture, you will be equipped with both the mathematical foundations and practical tools to apply edge detection in real imaging pipelines.

01

Define & Motivate

Define **edge detection** and explain its role in computer vision and medical imaging pipelines.

02

Classify Methods

Distinguish between **gradient-based (first-order)** and **second-order derivative** edge detection approaches.

03

Compute Gradients

Calculate and interpret **Gx, Gy**, gradient magnitude, and gradient direction from given patches.

04

Apply Operators

Identify and apply **Sobel, Prewitt, Roberts, Laplacian, and Canny** operators with their kernels.

05

Implement in Python

Connect theory to **OpenCV practice** and apply edge detection to real images programmatically.

What Is Edge Detection?

Edge detection is a fundamental image processing technique that locates the **boundaries of objects** within an image by identifying locations where pixel intensity changes sharply. Rather than analyzing every pixel's raw color or brightness value, edge detection distills the image to its most structurally informative content.

Edge Core Definition

An **edge** is a location in an image where there is a **strong, abrupt change in intensity** between neighboring pixels — typically signaling the boundary between two distinct regions or objects.

What It Preserves

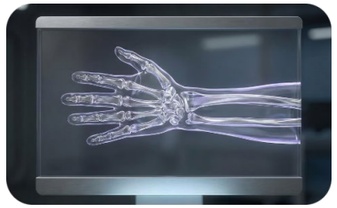
Edge detection simplifies an image by retaining its **structural skeleton**: object outlines, contours, and shape-defining boundaries, while discarding smooth interior regions.

Why It Matters

Edges are the input to higher-level vision tasks: object recognition, segmentation, feature extraction, measurement, and enhancement all become easier when structural boundaries are cleanly identified.

Why Edges Matter in Intelligent Medical Systems

In medical and biomedical imaging, edges carry direct clinical significance. Detecting them accurately is often the first step in automated diagnosis, measurement, and surgical planning.

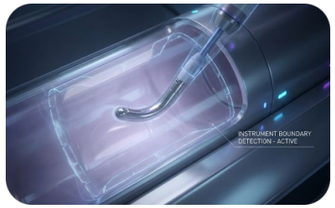


Radiology (X-ray / CT)

Detect **bone outlines**, fracture lines, and organ boundaries. Edges allow precise area and volume measurements of structures such as tumors or lesions.

MRI / Vascular Imaging

Identify **vessel borders**, tissue layer transitions, and lesion margins for segmentation and tracking of pathological changes over time.



Endoscopy / Surgery

Locate **instrument boundaries** and tissue regions in real-time surgical video, enabling robotic guidance and motion tracking.

Edges are nearly always the **first processing step** in a medical vision pipeline — before segmentation, classification, or measurement.

Basic Concept: Intensity Change

The Image as a Mathematical Function

A grayscale image is treated as a 2D function of spatial coordinates:

$$I(x, y)$$

where the output is the intensity (brightness) at position **(x, y)**. The key insight for edge detection is that edges correspond to locations where this function changes **rapidly**.

- **Smooth area** → intensity values between neighboring pixels are nearly equal → small change
- **Boundary area** → intensity values change abruptly between neighbors → large change

Intuition: Flat vs. Steep

Think of the intensity surface as a terrain landscape. A **flat plateau** means no edge. A **steep cliff** means a strong edge. Edge detection is mathematically equivalent to finding these steep slopes — which is exactly what derivatives measure.

Key rule: Large derivative value → strong edge. Near-zero derivative value → smooth region.

Two Categories of Edge Detection

All classical edge detection methods fall into one of two mathematical families, each exploiting a different property of how intensity changes at boundaries.

Category 1 — Gradient-Based (First Derivative)

These methods detect edges by computing the **first derivative** of image intensity. The underlying idea is that edges occur where the first derivative is **large in magnitude**.

Operators: Sobel, Prewitt, Roberts Cross

Strengths: intuitive, efficient, widely used. Weakness: sensitive to noise without additional smoothing.

Category 2 — Second-Order Derivative

These methods compute the **second derivative** of image intensity. The key idea is that edges occur at **zero-crossings** of the second derivative — where the slope transitions from positive to negative or vice versa.

Operator: Laplacian

Strengths: precise edge localization. Weakness: highly sensitive to noise — must smooth first.

1D Edge (First Difference)

Before moving to 2D images, it is essential to understand edge detection in one dimension. This lays the foundation for the gradient operators that follow.

Definition: First Difference

For a 1D discrete signal **I[x]**, the simplest approximation of the derivative (rate of change) at position x is:

$$\Delta I = I[x+1] - I[x]$$

If **|ΔI| is large** → strong edge at position x.
If **|ΔI| ≈ 0** → smooth region, no edge.

Worked Example

Given signal: **I = [10, 12, 11, 50, 52]**

Compute differences at each step:

Position	Difference	Edge?
x=0 → x=1	12 − 10 = 2	No
x=1 → x=2	11 − 12 = −1	No
x=2 → x=3	50 − 11 = **39**	✅ Edge
x=3 → x=4	52 − 50 = 2	No

At position x=2→3, the difference of **39** is far larger than the others, correctly identifying a boundary.

Image Gradient (2D)

For a 2D image, edges can run in any direction — horizontal, vertical, or diagonal. We therefore compute the rate of intensity change in **both spatial directions** independently.

Horizontal Gradient Gx

$$G_x = \frac{\partial I}{\partial x}$$

Measures how intensity changes **left-to-right** across columns. A large positive Gx means intensity increases sharply going right; a large negative value means it decreases. Gx is large at **vertical edges**.

Vertical Gradient Gy

$$G_y = \frac{\partial I}{\partial y}$$

Measures how intensity changes **top-to-bottom** across rows. Gy is large at **horizontal edges**. Together, Gx and Gy form the **gradient vector** that fully characterizes the local edge.

The Gradient Vector

$$\nabla I = (G_x, G_y)$$

The gradient is a vector pointing in the direction of **greatest intensity increase**. Its magnitude tells us edge strength; its angle tells us edge orientation. Both are extracted from Gx and Gy.

Gradient Magnitude and Direction

Edge Strength — Magnitude

The **magnitude** of the gradient combines Gx and Gy into a single scalar that represents **how strong the edge is** at each pixel:

$$| \nabla I | \approx \sqrt{G_x^2 + G_y^2}$$

A large magnitude (close to 255 in an 8-bit image) → **strong, sharp edge**.
A small magnitude (near 0) → **smooth region, no edge**.

> A common fast approximation used in real-time systems: **|Gx| + |Gy|** (avoids the square root).

Edge Orientation — Direction

The **direction** (angle) of the gradient tells you the orientation of the edge:

$$\theta = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

Important interpretation:

- θ ≈ 0° or 180° → the edge runs **vertically** (intensity changes left-to-right)
- θ ≈ 90° → the edge runs **horizontally** (intensity changes top-to-bottom)
- θ ≈ 45° or 135° → the edge is **diagonal**

Direction is used in the **Canny non-maximum suppression** step to thin edges.

Convolution and Kernels

All the gradient-based operators (Sobel, Prewitt, Roberts, Laplacian) compute Gx and Gy by **convolving** the image with small weight matrices called **kernels**.

The kernel encodes which pixels to compare and how to weight them. Different kernels emphasize different directions of change. This is why Sobel, Prewitt, and Laplacian each have their own characteristic kernels — they embody different mathematical approximations of the derivative.

GRADIENT-BASED OPERATORS

Sobel Operator — Concept

The **Sobel operator** is the most widely used gradient-based edge detector. It estimates the first-order derivative in both spatial directions using a **3×3 kernel** that incorporates a mild **smoothing effect**, making it less sensitive to noise than a simple finite difference.

Why Sobel Is Better Than Simple Differences

A raw pixel difference (I[x+1] − I[x]) is too sensitive to local noise. Sobel adds a **Gaussian-like weighting** in the perpendicular direction — effectively averaging over three rows/columns — which smooths out small noise fluctuations while still detecting real edges.

Two Kernels, Two Directions

Sobel uses **two separate 3×3 kernels**: Sx to detect horizontal change (vertical edges) and Sy to detect vertical change (horizontal edges). Both are applied to the same image, and their outputs are combined to find all edges.

Sobel Kernels — Exact Values

Memorize these two kernels. The sign pattern encodes the subtraction (right − left or bottom − top), and the weight of 2 in the center row/column provides the smoothing effect.

Sx — Detects Vertical Edges (Horizontal Change)

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

The left column has **negative weights** (−1, −2, −1) and the right column has **positive weights** (+1, +2, +1). When convolved with the image, this subtracts left intensities from right intensities → measures horizontal change → detects **vertical edges**.

Sy — Detects Horizontal Edges (Vertical Change)

$$S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

The top row has **negative weights** and the bottom row has **positive weights**. This subtracts top intensities from bottom intensities → measures vertical change → detects **horizontal edges**. Note the center row is all zeros — the middle row contributes nothing.

Sobel Workflow — Step by Step

Step 1

Convolve the grayscale image with **Sx** at every pixel position to obtain the horizontal gradient map **Gx**.

Step 2

Convolve the same image with **Sy** at every pixel position to obtain the vertical gradient map **Gy**.

Step 3

Compute the **gradient magnitude** per pixel:

$$|\nabla I| \approx \sqrt{G_x^2 + G_y^2}$$

Step 4 (Optional)

Compute the **gradient direction** per pixel:

$$\theta = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

Steps 1–3 are mandatory for edge detection. Step 4 is used in Canny for non-maximum suppression and in orientation-sensitive applications.

Sobel — Worked Example with Numbers

This example walks through the full Sobel computation on a toy 3×3 patch to build concrete numerical intuition.

The Input Patch P

$$P = \begin{bmatrix} 10 & 10 & 10 \\ 10 & 10 & 10 \\ 50 & 50 & 50 \end{bmatrix}$$

This patch represents a clear **horizontal edge**: the top two rows are dark (intensity = 10) and the bottom row is bright (intensity = 50). We expect Gy to be large and Gx to be near zero.

Computing Gy Using Sy

Top-row contribution (negative weights: −1, −2, −1):

$$(-1)(10) + (-2)(10) + (-1)(10) = -40$$

Middle row contribution (all zeros): **0**

Bottom-row contribution (positive weights: +1, +2, +1):

$$(1)(50) + (2)(50) + (1)(50) = 200$$

Total Gy = −40 + 0 + 200 = **160**

Since Gx ≈ 0 (all columns identical):

$$|\nabla f| \approx \sqrt{0^2 + 160^2} = 160$$

✅ A magnitude of 160 on a 0–255 scale is a **strong edge**, exactly as expected from the 40-unit intensity jump.

GRADIENT-BASED OPERATORS

Prewitt Operator — Concept & Kernels

The **Prewitt operator** is structurally identical to Sobel in its philosophy: it estimates Gx and Gy using two 3×3 kernels and computes magnitude and direction the same way. The key difference is in the kernel weights.

Px — Horizontal Gradient

$$P_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

All three rows are weighted **equally (±1, 0)**. Unlike Sobel, there is no center-row emphasis. This results in **uniform averaging** across all three rows.

Py — Vertical Gradient

$$P_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Again, all weights are ±1 — no center row emphasis. Magnitude and direction are computed identically to Sobel.

Sobel vs. Prewitt: Sobel weights the center row/column by 2, giving slightly better noise resistance. Prewitt uses uniform weights — simpler, slightly faster, slightly noisier.

GRADIENT-BASED OPERATORS

Roberts Cross Operator — Concept & Kernels

What Makes Roberts Different?

The **Roberts Cross** operator uses **2×2 kernels** instead of 3×3. Rather than measuring changes along horizontal and vertical axes, it measures changes along the **two diagonal directions** (45° and 135°).

Kernel 1 (diagonal ↘):

$$R_1 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Kernel 2 (diagonal ↗):

$$R_2 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

Magnitude and direction follow the same formulas:

$$|\nabla I| = \sqrt{G_1^2 + G_2^2}, \quad \theta = \tan^{-1}\left(\frac{G_2}{G_1}\right)$$

Trade-offs

✅ Fast

2×2 kernel = fewer multiplications per pixel. Computationally the lightest of all classical operators.

✅ Diagonal Edges

Specifically tuned to detect edges running at 45° and 135° — an advantage when diagonal features are important.

⚠️ Noise Sensitive

Smaller kernel = no smoothing at all. Very susceptible to noise. Not recommended for noisy medical images without pre-filtering.

SECOND-ORDER METHODS

The Laplacian Operator — Second Derivative

The **Laplacian** is a second-order differential operator. Instead of finding where the first derivative is large, it finds where the **second derivative crosses zero** — a more precise but noisier edge localization technique.

Mathematical Definition

The continuous Laplacian is the sum of second partial derivatives in both directions:

$$\nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

In discrete form, this is approximated by the following 3×3 kernel:

$$L = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

The center pixel is multiplied by 4 and surrounded by −1 weights. Convolution with this kernel approximates the second derivative at each pixel.

Zero-Crossing Interpretation

At an edge, the intensity profile peaks — its first derivative is maximum and its **second derivative crosses zero**. The Laplacian directly finds this zero-crossing, which in theory gives very precise edge location.

Critical warning: The Laplacian amplifies noise drastically — noise introduces small intensity fluctuations that produce large second-derivative responses. **Always apply Gaussian smoothing before using the Laplacian** (see LoG: Laplacian of Gaussian).

Operator Comparison

Use this table to quickly compare all five operators across the dimensions that matter most for practical application.

Operator	Kernel Size	Order	Noise Robustness	Edge Quality	Best For
Sobel	3×3	1st	Good (weighted avg)	Thick edges	General use, noisy images
Prewitt	3×3	1st	Moderate	Thick edges	Simple, fast detection
Roberts	2×2	1st	Poor (no smoothing)	Thin, diagonal	Fast, diagonal features
Laplacian	3×3	2nd	Poor (amplifies noise)	Very thin (LoG)	Precise localization (with LoG)

Sobel in OpenCV

```
import cv2
import numpy as np

img = cv2.imread("image.jpg", cv2.IMREAD_GRAYSCALE)
blur = cv2.GaussianBlur(img, (5, 5), 1.4)

Gx = cv2.Sobel(blur, cv2.CV_64F, 1, 0, ksize=3)
Gy = cv2.Sobel(blur, cv2.CV_64F, 0, 1, ksize=3)

mag = np.sqrt(Gx**2 + Gy**2)

Gx_disp  = cv2.convertScaleAbs(Gx)   # abs + convert to uint8
Gy_disp  = cv2.convertScaleAbs(Gy)
mag_disp = cv2.convertScaleAbs(mag)
```

```
cv2.imshow("Sobel |Gx|", Gx_disp)
cv2.imshow("Sobel |Gy|", Gy_disp)
cv2.imshow("Sobel Magnitude", mag_disp)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Prewitt in OpenCV

```
import cv2
import numpy as np

img = cv2.imread("image.jpg", cv2.IMREAD_GRAYSCALE)
blur = cv2.GaussianBlur(img, (5, 5), 1.4)

Px = np.array([[-1, 0,  1],
               [-1, 0,  1],
               [-1, 0,  1]], dtype=np.float32)

Py = np.array([[-1, -1, -1],
               [ 0,  0,  0],
               [ 1,  1,  1]], dtype=np.float32)

Gx = cv2.filter2D(blur, cv2.CV_64F, Px)
Gy = cv2.filter2D(blur, cv2.CV_64F, Py)

mag = np.sqrt(Gx**2 + Gy**2)

Gx_disp  = cv2.convertScaleAbs(Gx)
Gy_disp  = cv2.convertScaleAbs(Gy)
mag_disp = cv2.convertScaleAbs(mag)

cv2.imshow("Prewitt |Gx|", Gx_disp)
cv2.imshow("Prewitt |Gy|", Gy_disp)
cv2.imshow("Prewitt Magnitude", mag_disp)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Roberts Cross Edge Detection in OpenCV

```
import cv2
import numpy as np

img = cv2.imread("image.jpg", cv2.IMREAD_GRAYSCALE)
blur = cv2.GaussianBlur(img, (5, 5), 1.4)

R1 = np.array([[ 1, 0],
               [ 0,-1]], dtype=np.float32)

R2 = np.array([[ 0, 1],
               [-1, 0]], dtype=np.float32)

G1 = cv2.filter2D(blur, cv2.CV_64F, R1)
G2 = cv2.filter2D(blur, cv2.CV_64F, R2)

mag = np.sqrt(G1**2 + G2**2)

G1_disp  = cv2.convertScaleAbs(G1)
G2_disp  = cv2.convertScaleAbs(G2)
mag_disp = cv2.convertScaleAbs(mag)

cv2.imshow("Roberts |G1| (diag1)", G1_disp)
cv2.imshow("Roberts |G2| (diag2)", G2_disp)
cv2.imshow("Roberts Magnitude", mag_disp)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Laplacian in OpenCV

```
import cv2
import numpy as np

img = cv2.imread("image.jpg", cv2.IMREAD_GRAYSCALE)
blur = cv2.GaussianBlur(img, (5, 5), 1.4)

lap = cv2.Laplacian(blur, cv2.CV_64F, ksize=3)
lap_disp = cv2.convertScaleAbs(lap)

cv2.imshow("Laplacian |response|", lap_disp)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Original Image

Edge Detected Image

Applications of Edge Detection

Medical Imaging

Highlight organ boundaries, lesion margins, and vessel borders for measurement, segmentation, and computer-aided diagnosis in radiology and pathology workflows.

Object Detection

Provide structural features for shape-based recognition and classical feature pipelines. Edge maps feed into HOG descriptors and contour-based detection algorithms.

Robotics

Enable real-time navigation and obstacle detection by identifying the boundaries of objects in the robot's visual field, supporting path planning and avoidance.

Automotive Vision

Detect lane markings and road boundaries in autonomous driving systems. Canny is commonly used in lane detection pipelines combined with Hough transform.

Challenges in Edge Detection

Even the best edge detectors encounter fundamental difficulties in real images. Understanding these challenges is essential for selecting the right method and parameters — especially in medical imaging where reliability is critical.

🔴 Noise → False Edges

Problem: Random pixel intensity variations create spurious high-gradient responses that appear as edges.

Solution: Apply Gaussian blur (σ tuned to noise level) before gradient computation. Canny's built-in blur step is designed specifically for this.

🟠 Texture → Edge Overload or Breaks

Problem: Textured surfaces (e.g., trabecular bone, liver parenchyma) produce hundreds of small local edges, flooding the output and potentially breaking meaningful object boundaries.

Solution: Increase Gaussian σ or apply morphological post-processing.

🟡 Illumination Changes → Unstable Edges

Problem: Uneven lighting causes the same boundary to have very different contrast in different regions of the image.

Solution: Normalize illumination (histogram equalization, CLAHE) before applying edge detection, or use adaptive thresholding.

🔵 Scale Differences → Parameter Sensitivity

Problem: Edges at different scales (fine vessel walls vs. broad organ boundaries) require different kernel sizes and threshold values. No single parameter set captures all scales.

Solution: Multi-scale edge detection or scale-space methods.

Lecture Key Takeaways

Review these core ideas before the next session. Each point maps to an exam-ready concept.

Edge = Intensity Change

An edge is defined by a rapid change in I(x,y). Large $|\nabla I|$ → strong edge. Near-zero → smooth region. This is the universal definition underlying all methods.

Two Families

1st derivative (Sobel, Prewitt, Roberts): find large gradient magnitude. **2nd derivative** (Laplacian): find zero-crossings. Know which family each operator belongs to.

Kernels Encode Derivatives

Sobel, Prewitt, and Roberts each use specific kernel weight patterns to approximate partial derivatives. The sign pattern (negative/positive) is what performs the subtraction. Know the exact kernel values.

Noise Is the Main Enemy

Always smooth before edge detection (especially for Laplacian and Roberts). In medical images, pre-processing quality directly determines edge quality downstream.

Thank you

Google Classroom