Lecture 5

Edge Detection, Morphological Operations, Corners & Lines

Dr. Hugo Armando GUILLEN RAMIREZ

Department of Visceral Surgery and Medicine
Department of Digital Medicine
University of Bern

Spring 2026

Outline

The Road So Far...

- Linear filtering (convolution/correlation)
- Box / Gaussian filters
- Smoothing to avoid aliasing
- Correlation is a form of “matching” (e.g., dot product)
- Normalized cross-correlation
- Pyramid representations for scale variations

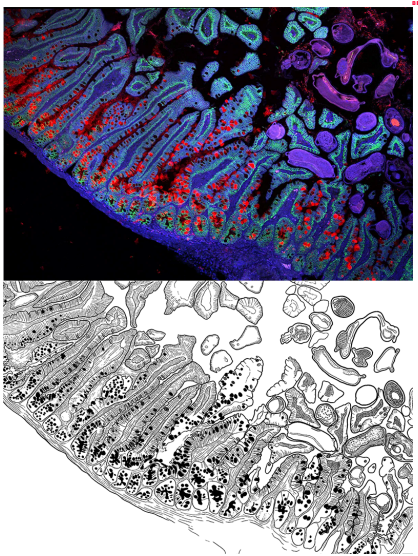
Learning Objectives

By the end of this lecture you will be able to:

1. **Detect edges** in images using gradient filters (Sobel) and the Canny pipeline.
2. **Clean up** binary masks using morphological operations (dilation, erosion, opening, closing).
3. **Find corners** with the Harris detector.
4. **Find lines** with the Hough transform.

Main idea

These are the building blocks you will chain together in real pipelines: *detect edges* → *clean them up* → *find features* (corners, lines) → feed into higher-level tasks (segmentation, tracking, 3-D reconstruction).

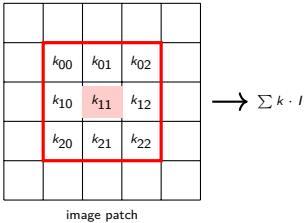


Warm-Up: What Is Convolution Again?

Convolution "slides" a small kernel K over the image I and at every pixel computes a weighted sum of the neighbourhood.

Step by step (for a 3×3 kernel):

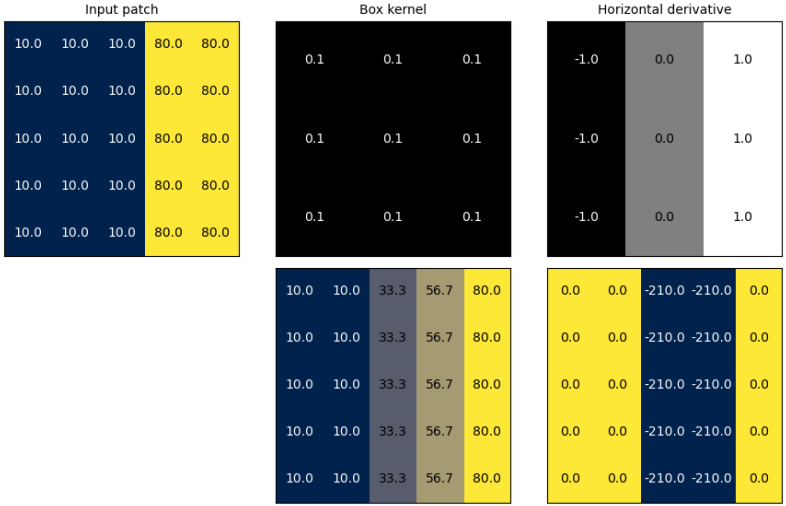
1. Place the kernel centred on pixel (i, j).
2. Multiply each kernel weight by the pixel underneath it.
3. Sum all products $\rightarrow$ that sum becomes the output at (i, j).
4. Slide to the next pixel and repeat.



Quick Recall

A **smoothing** kernel (e.g. Gaussian) has all-positive weights $\rightarrow$ output is a *blurred* version.
A **derivative** kernel has positive *and* negative weights $\rightarrow$ output highlights *change*.

Convolution: smoothing vs. derivative kernel



Gradient-Based Edge Detection

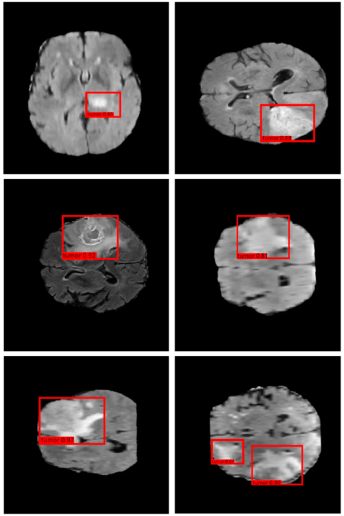
Why Do We Care About Edges?

Edges mark **boundaries** in the image. They arise from:

- **Object boundaries** — where one thing ends and another begins (depth discontinuity).
- **Surface creases** — folds or ridges on a surface.
- **Markings / texture borders** — e.g. stripes, labels.
- **Shadows** — illumination changes.

In **medical imaging**, edges help delineate organs, tumours, vessels, and bone boundaries.

We can find edges on an image by computing its derivative.



doi: 10.3390/bioengineering12010062

What Is a Derivative of an Image?

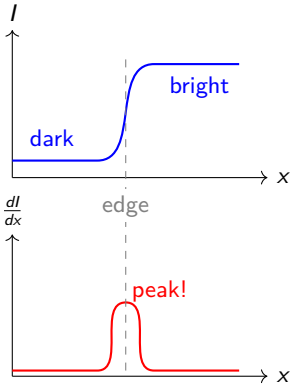
In 1-D, the derivative measures *how fast* a function changes.
For a discrete signal (a row of pixels):

$$\left.\frac{dI}{dx}\right|_i \approx I[i+1] - I[i]$$

This is called a **finite difference**.

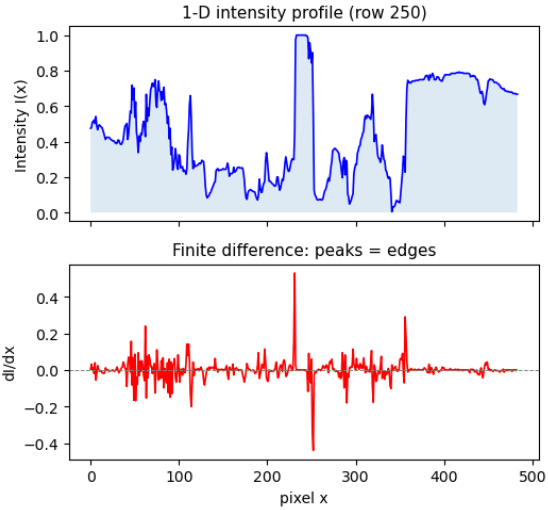
Edge detection = finding where the intensity changes sharply (the derivative is large).
Where the intensity is constant, the derivative is *zero*.

The rest of the first part of the lecture is just different ways to compute derivatives robustly in 2-D.



Johnny Silvercat (484x484)


Original Image


1-D intensity profile (row 250)
Intensity I(x)
1.0
0.8
0.6
0.4
0.2
0.0
Finite difference: peaks = edges
dI/dx
0.4
0.2
0.0
−0.2
−0.4
0
100
200
300
400
500
pixel x

The Image Gradient: Extending Derivatives to 2-D

An image has two directions (x and y), so we compute a **gradient vector** at every pixel:

$$\nabla I = \begin{pmatrix} \partial I/\partial x \\ \partial I/\partial y \end{pmatrix}$$

Two useful quantities we derive from this vector:

Gradient magnitude

"how strong is the edge?":

$$|\nabla I| = \sqrt{I_x^2 + I_y^2}$$

Gradient direction

"which way does the edge face?":

$$\theta = \arctan\big(I_y \,/\, I_x\big)$$

Think of the gradient as an arrow at each pixel.
The arrow *length* tells you edge strength; the arrow *direction* points from dark to bright (perpendicular to the edge line itself).

The Sobel Operator

We approximate $\partial I/\partial x$ and $\partial I/\partial y$ by convolving with small kernels. **Sobel is the most commonly used**:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \qquad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Why does this work?

- Positive on one side, negative on the other; measures *change*.
- The centre row/column gets weight ± 2; a mini Gaussian blur that suppresses noise.
- G_y is simply G_x transposed.

Other operators (FYI):

- **Roberts** (2×2): fast but very noisy.
- **Prewitt** (3×3): equal weights, less smoothing.

Sobel: A Worked Pixel Example

Suppose we have this 3×3 neighbourhood around pixel (i, j):

$$\text{Patch} = \begin{bmatrix} 10 & 10 & 80 \\ 10 & 10 & 80 \\ 10 & 10 & 80 \end{bmatrix} \qquad G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

Element-wise multiply, then sum:

$$\begin{aligned} I_x &= (-1)(10) + 0(10) + (+1)(80) \\ &\quad + (-2)(10) + 0(10) + (+2)(80) \\ &\quad + (-1)(10) + 0(10) + (+1)(80) \\ &= -10 + 80 - 20 + 160 - 10 + 80 = \mathbf{280} \end{aligned}$$

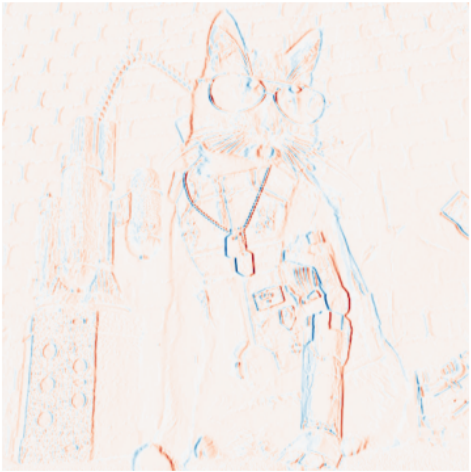
A large positive I_x means there is a strong **vertical edge** here (intensity increases from left to right).

Applying G_y to the same patch gives $I_y = 0$ (no change vertically). So:

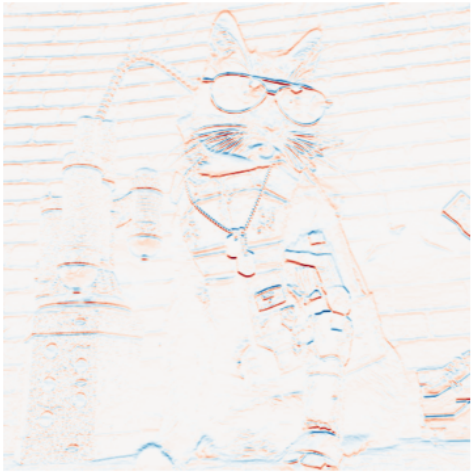
$$|\nabla I| = \sqrt{280^2 + 0^2} = 280$$

Strong edge, pointing horizontally.

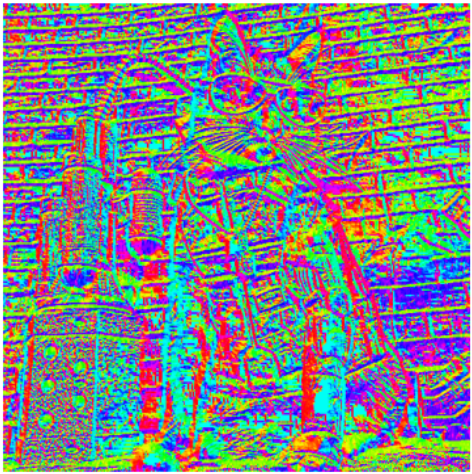
Gx (horizontal)



Gy (vertical)



Direction Θ



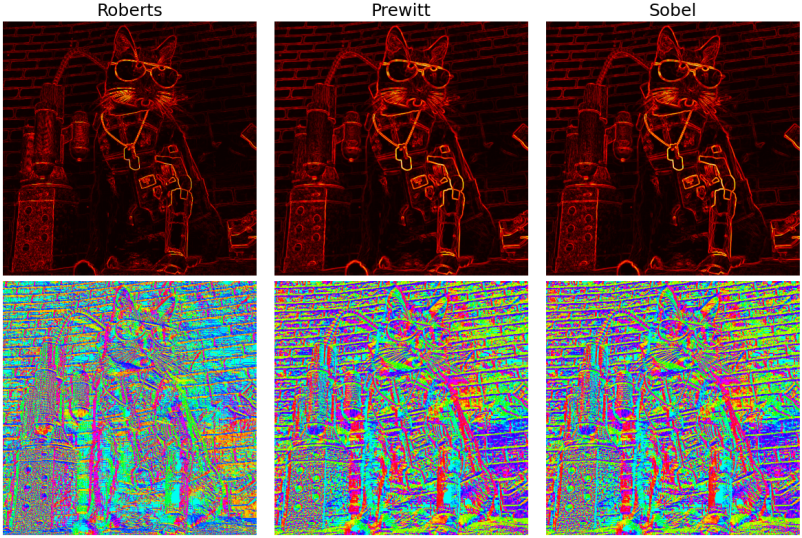
Gradient magnitude |∇I|

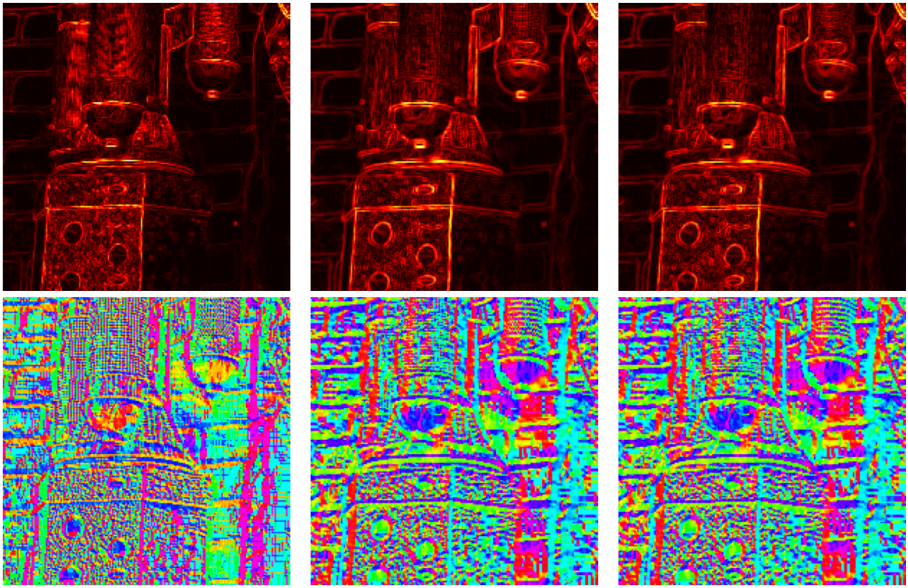
Finite-Difference Kernels: Roberts, Prewitt, Sobel

We approximate $\partial I/\partial x$ and $\partial I/\partial y$ by convolving with small kernels:

Operator	G_x kernel (horizontal edges)	G_y kernel (vertical edges)
Roberts	$\begin{bmatrix} +1 & 0 \\ 0 & -1 \end{bmatrix}$	$\begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix}$
Prewitt	$\begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix}$	$\begin{bmatrix} +1 & +1 & +1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$
Sobel	$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$	$\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$

All three kernels are **positive on one side, negative on the other.**
Sobel adds extra weight (± 2) to the centre row/column for mild smoothing perpendicular to the derivative direction.

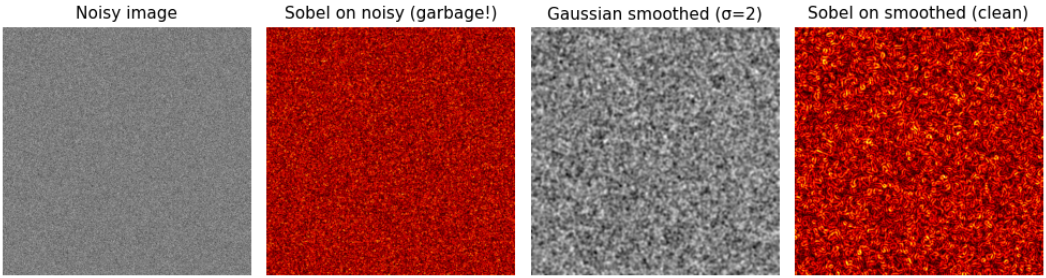

Roberts
Prewitt
Sobel


Roberts
Prewitt
Sobel

Problem: Noise Ruins Simple Derivatives

Derivatives amplify high-frequency content including **noise**.

Naive approach: smooth first with a Gaussian, *then* differentiate.



Solution: Combine Smoothing + Derivative in One Kernel

Because convolution is **associative**, we can merge smoothing and differentiation into a *single* kernel:

$$\frac{\partial}{\partial x}\big(G_\sigma * I\big) = \underbrace{\frac{\partial G_\sigma}{\partial x}}_{\text{DoG kernel}} * I$$

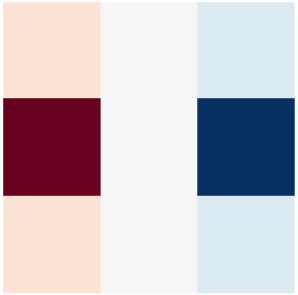
In practice:

- One convolution instead of two — faster.
- Larger $\sigma \rightarrow$ coarser edges, less noise.
- Smaller $\sigma \rightarrow$ finer detail, more noise.

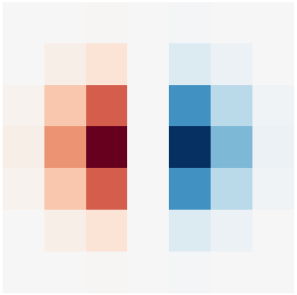
Key takeaway

The DoG kernel is just a "smoothed derivative". Instead of memorising the formula, remember: **blur + differentiate = one combined kernel.**

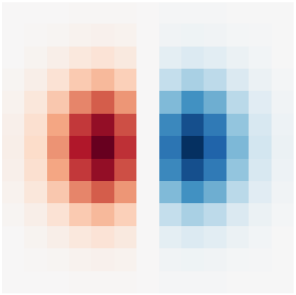
DoG kernel σ=0.5



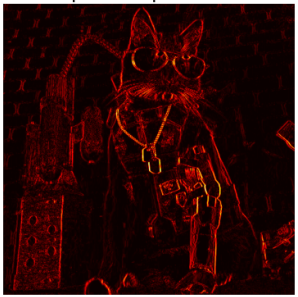
DoG kernel σ=1.0



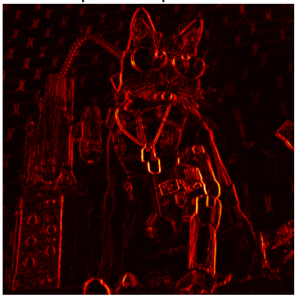
DoG kernel σ=2.0



|DoG * I| σ=0.5



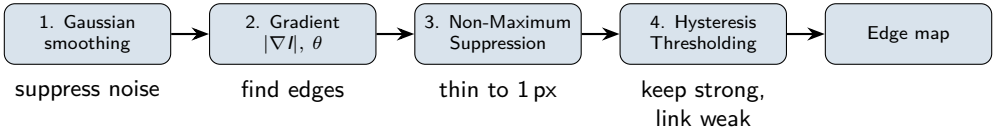
|DoG * I| σ=1



|DoG * I| σ=2

The Canny Edge Detector

The Canny detector (1986) is the **gold standard**. It chains four steps:

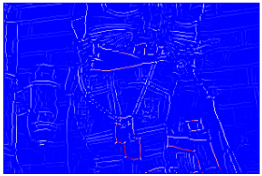


- Steps 1–2 are what we already know (smooth + derivative). Kernels can be combined into one using convolution.
- Steps 3–4 are the *clever part*: they clean up the raw gradient to produce thin, well-connected edges instead of thick blurry blobs.

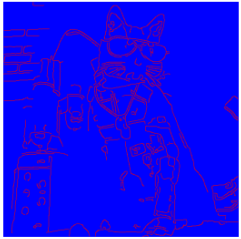
1. Gaussian smoothing

2. Gradient magnitude

3. After NMS (crop)



4. Final Canny output



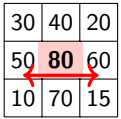
Step 3: Non-Maximum Suppression (NMS)

Problem: the gradient magnitude map has *thick ridges* around edges. We want edges that are exactly **one pixel wide**.

Idea: at each pixel, look at the two neighbours along the gradient direction. If this pixel is not the local maximum, suppress it (set to 0).

Algorithm (quantised NMS):

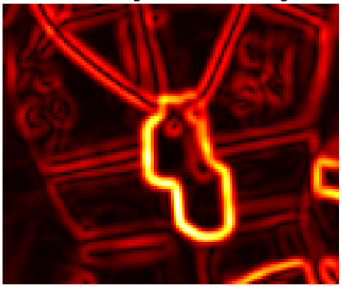
1. Round the gradient direction θ to the nearest of $0°$, $45°$, $90°$, $135°$.
2. Compare $|\nabla I|(i,j)$ with the two neighbours along that direction.
3. If the current pixel is **not** the largest of the three, set it to zero.



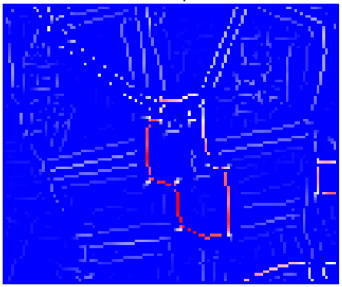
$\theta \approx 0° \Rightarrow$ compare left & right
$80 > 50$ and $80 > 60 \rightarrow$ **keep**

Original (crop)

Gradient magnitude (thick ridges)



After NMS (1-pixel wide)

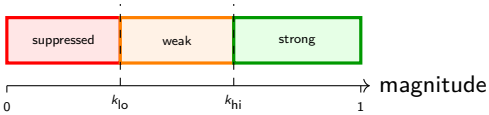


Step 4: Hysteresis Thresholding

After NMS we still have many pixels. A single threshold would either miss weak edges or keep too much noise.

Solution: use two thresholds $k_{\text{lo}} < k_{\text{hi}}$:

- Pixels $\geq k_{\text{hi}}$: **strong edges** (always kept).
- Pixels $\in [k_{\text{lo}}, k_{\text{hi}})$: **weak edges** (kept **only if** 8-connected to a strong edge pixel).
- Pixels $< k_{\text{lo}}$: definitely noise (suppressed).

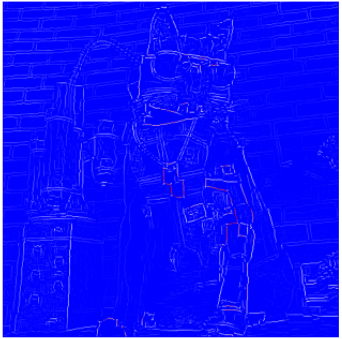


Rule of thumb (Canny's own suggestion): $k_{\text{hi}}/k_{\text{lo}} \approx 2$.

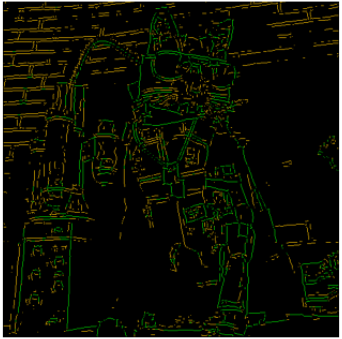
Hysteresis: k_{lo}=0.2, k_{hi}=0.4

Green=strong; Yellow=weak;

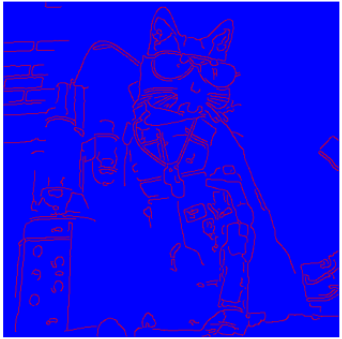
Non-maximum suppression



Black=suppressed



Final Canny (after linking)



Canny in Practice: Tuning Guide

Two knobs to turn:

- σ: controls *scale*. Stronger smoothing $\rightarrow$ fewer, coarser edges.
- $k_{\text{lo}}, k_{\text{hi}}$: control edge sensitivity.
  Low thresholds $\rightarrow$ more edges (including some noise). High thresholds $\rightarrow$ only very strong edges.

Common Pitfall

Change one parameter at a time when tuning! Start with defaults, then adjust σ first, then thresholds.

Quick Look: Laplacian-of-Gaussian (LoG)

An alternative to Canny: instead of peaks in the first derivative, look for **zero crossings** in the *second* derivative.

The Laplacian sums the second derivatives in x and y:

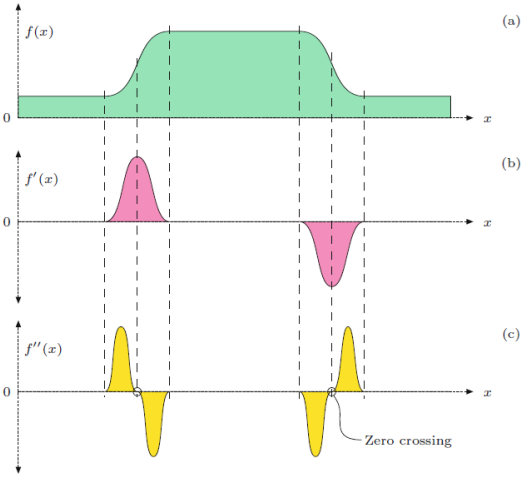
$$\nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} \qquad \text{Kernel: } \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Applied alone it is very noisy, so we combine it with Gaussian smoothing $\rightarrow$ the **LoG** ("Mexican hat" kernel).

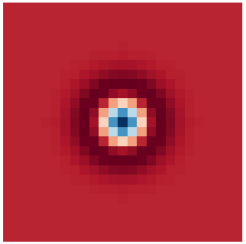
Edges = zero crossings of $(\text{LoG} * I)$.

When to use?

LoG gives closed contours (which Canny doesn't), but is harder to tune. You'll see it again in SIFT feature descriptors.

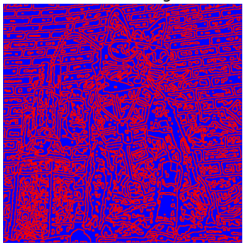


LoG kernel (σ=2,k=25)
"Mexican hat"

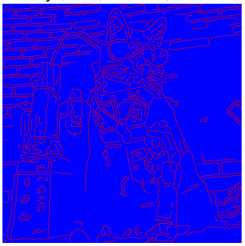


LoG * I

Zero crossings



Canny (σ=2,lo=0.08,hi=0.16)



Exercise 1: Canny Pipeline Exploration (∼10 min)

Tasks

1. Apply Canny to an image of your choice with default parameters.
2. Sweep $\sigma \in \{0.5, 1, 2, 3\}$ and $k_{\text{hi}} \in \{0.05, 0.10, 0.20\}$; display the resulting edge maps in a 4×3 grid.

Morphological Operations

Motivation: Why Do We Need Morphology?

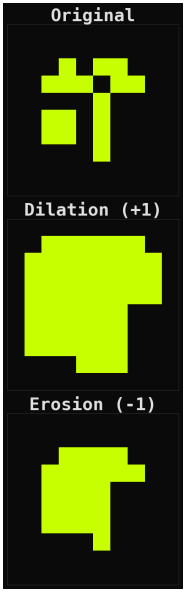
After edge detection (or thresholding), you typically have a (messy) **binary mask**:

- Small noise blobs that should not be there.
- Tiny holes inside regions that should be solid.
- Objects that touch or overlap when they should be separate.

Morphological operations are a toolkit for *cleaning up binary (and grey-scale) images* by probing them with a small shape called a **structuring element** (SE).

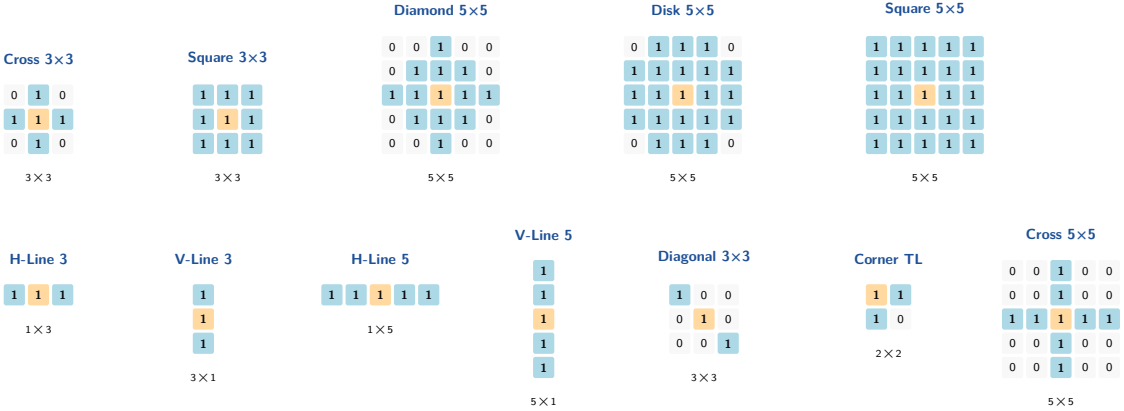
Notation

1 = *foreground*
0 = *background*



Structuring Elements Catalog

Common SE shapes: **disk**, **square**, **cross**. The *size* determines the scale of features affected.



The Four Basic Operations

Operation	Effect	
Dilation ($\oplus$)	Grow foreground	Any pixel the SE *touches* becomes white.
Erosion ($\ominus$)	Shrink foreground	Only anchor pixels where the SE *fits entirely* survive.
Opening	Remove small bright noise	Erosion *then* dilation.
Closing	Fill small dark holes	Dilation *then* erosion.

Intuition

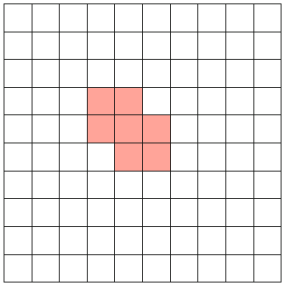
Opening removes noise outside objects.
Closing removes noise inside objects.
When in doubt, apply *both* in sequence.

Cross 3×3 Dilation Example

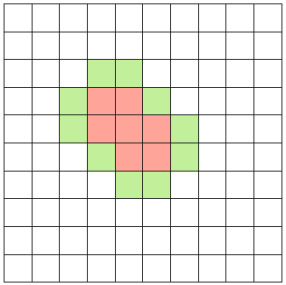
Dilation ($\oplus$):

- Fills small gaps and holes.
- Connects nearby objects.
- Objects get *bigger*.

0	1	0
1	1	1
0	1	0

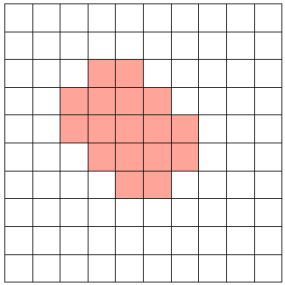


(a)



(b)

3×3



(c)

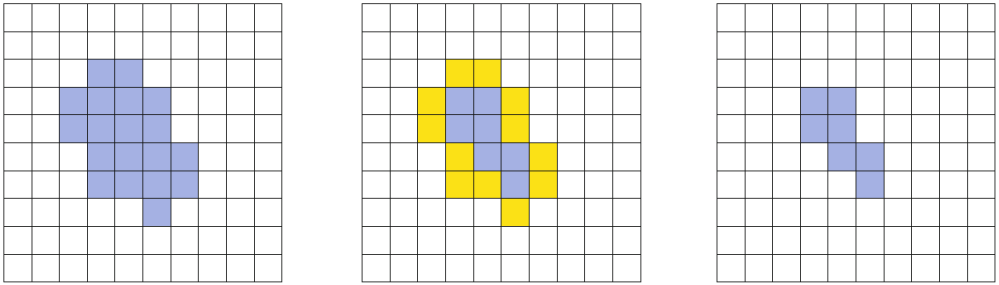
Cross 3×3 Erosion Example

Erosion ($\ominus$):

- Removes thin protrusions.
- Separates touching objects.
- Objects get *smaller*.

0	1	0
1	1	1
0	1	0

3×3



(a) (b) (c)

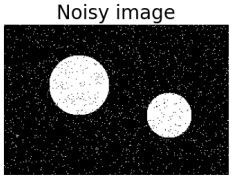
Opening

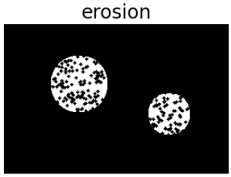
Opening removes noise outside objects.

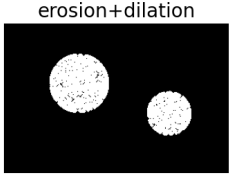
diamond(2)

0	0	1	0	0
0	1	1	1	0
1	1	1	1	1
0	1	1	1	0
0	0	1	0	0

5×5







Closing

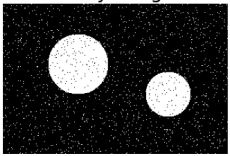
Closing removes noise inside objects.

diamond(2)

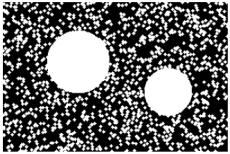
0	0	1	0	0
0	1	1	1	0
1	1	1	1	1
0	1	1	1	0
0	0	1	0	0

5 × 5

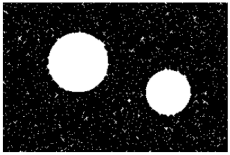
Noisy image



dilation



dilation+erosion

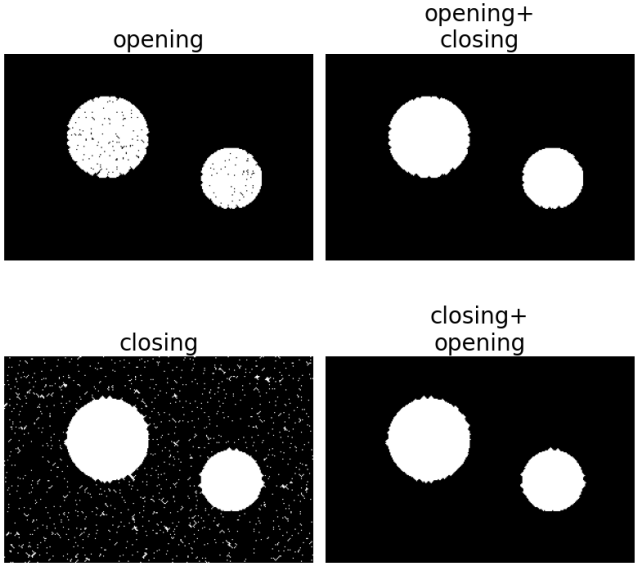


Combining operations

diamond(2)

0	0	1	0	0
0	1	1	1	0
1	1	1	1	1
0	1	1	1	0
0	0	1	0	0

5 × 5



Morphological Gradient (Edge Detector)

A simple edge detector built from morphology:

$$\text{grad}(I) = (I \oplus B) - (I \ominus B)$$

Dilated minus eroded = thick edges at boundaries. Works on both binary *and* grey-scale images.

```python
from skimage.morphology import (
    dilation, erosion, disk)

se = disk(3)
grad = (dilation(img, se)
      - erosion(img, se))
```

Intuition

Where the boundary is, dilation "pushes out" and erosion "pulls in" — the difference is large only at edges.

Disk 7×7

0	0	0	1	0	0	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
1	1	1	1	1	1	1
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	0	0	1	0	0	0

7×7

Original

dilation − erosion

Sobel mag on original

Exercise 2 — Morphological Clean-Up (∼10 min)

UNIVERSITÄT BERN

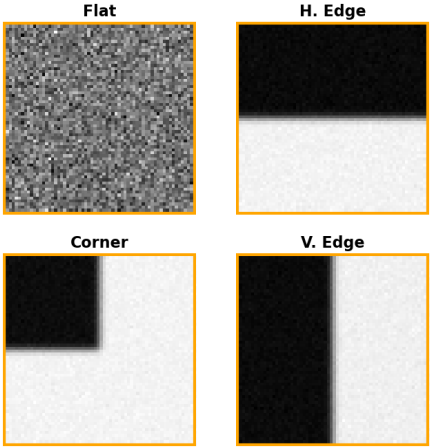
Tasks

1. Apply morphological operations to clean the image on the cell.

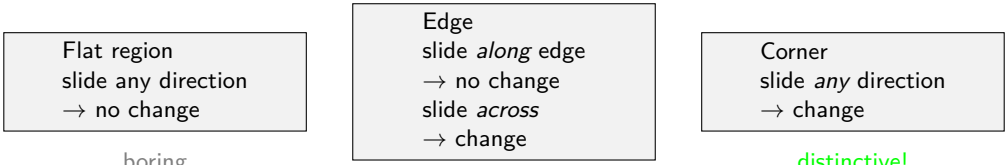
Corners & Lines

Corners: The Aperture Problem

Imagine you look at a small patch of an image through a tiny window(aperture).
If the image shifts, can you tell which way it moved?



Corners: The Aperture Problem



Why corners?

Corners are special because they are **locally unique**: if you shift the window in *any* direction, the content changes.
That makes them ideal anchor points for **matching**, **tracking**, and **stitching** images.

Harris Corner Detector

Goal: find pixels where the gradient is strong in *multiple* directions (not just one direction like an edge).

How it works:

1. Compute I_x, I_y with Sobel.
2. In a small Gaussian-weighted window around each pixel, build the **structure tensor**:

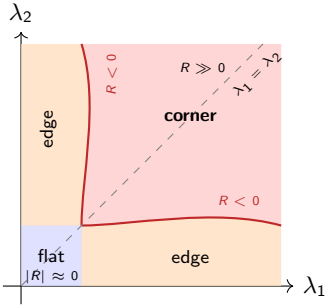
$$M = \begin{pmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{pmatrix}$$

3. The eigenvalues λ_1, λ_2 of M describe the gradient distribution in that window.

Harris computes a **response score** (avoiding explicit eigenvalues):

$$R = \det(M) - k\,(\operatorname{tr} M)^2$$

- $R \gg 0$: **corner** (both λ_1, λ_2 large).
- $R \ll 0$: edge (one λ dominates).
- $|R| \approx 0$: flat region (both λ small).



Harris Corner Detection in skimage

```python
from skimage.feature import corner_harris,
    corner_peaks
import matplotlib.pyplot as plt

# 1. Compute the Harris response map
R = corner_harris(img_gray, k=0.05, sigma=1.0)

# 2. Find corner locations (peaks in R)
corners = corner_peaks(R, min_distance=10,
threshold_rel=0.02)

# 3. Visualize results
fig, ax = plt.subplots()
ax.imshow(img_gray, cmap='gray')
ax.plot(corners[:, 1], corners[:, 0],
'r+', markersize=12)
ax.set_title(f'{len(corners)} corners detected')
plt.show()
```

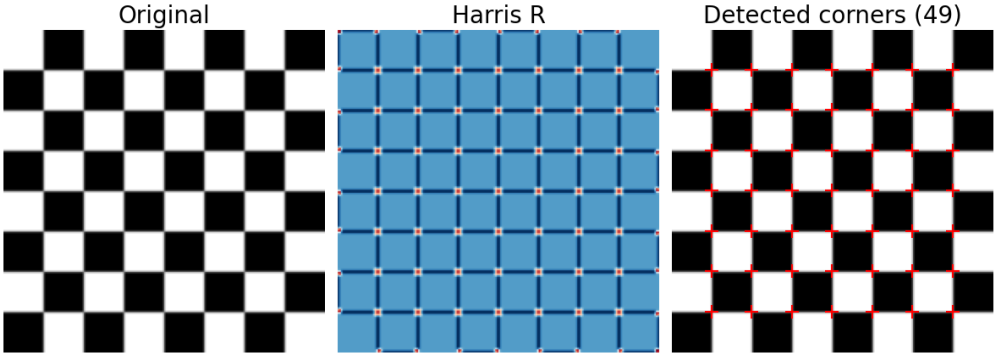
Key Parameters to Tune

`corner_harris`

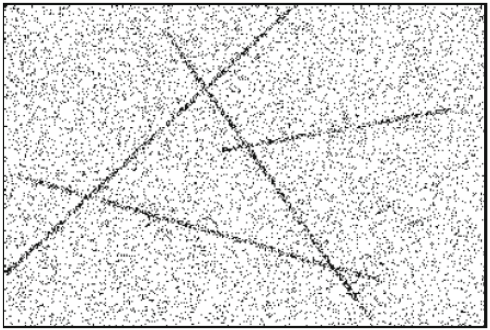
- `k` (≈ 0.04–0.06): Sensitivity. Lower $k \rightarrow$ more corners.
- `sigma`: Gaussian window size. Larger $\rightarrow$ more robust to noise, but less precise.

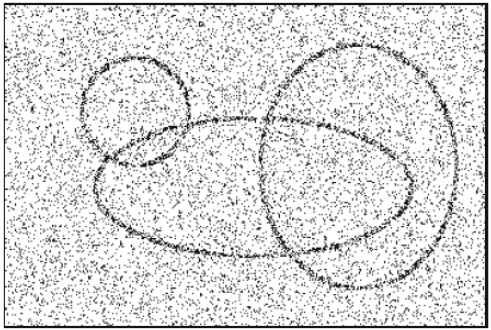
`corner_peaks`

- `min_distance`: Suppresses nearby duplicate detections.
- `threshold_rel`: Minimum response (R) relative to the maximum peak.


Original
Harris R
Detected corners (49)

Line Detection





The Hough Transform: Core Concept

Objective: Identify collinear points within an edge map to extract straight lines.

Methodology (Accumulator Voting):

1. Parameterize lines using polar coordinates to avoid undefined (infinite) slopes:

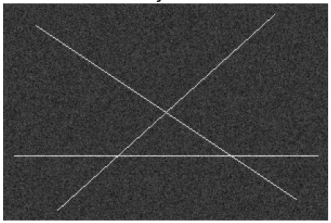
$$x\cos\theta + y\sin\theta = \rho$$

2. Each edge pixel maps to all potential lines passing through it, generating a sinusoidal curve in the (θ, ρ) parameter space.
3. Intersections of multiple curves signify concurrent votes, corresponding to the presence of a line in the spatial domain.

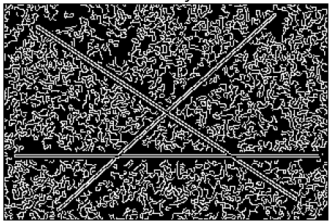
Intuition

The process functions as a voting system. Each edge pixel casts a "vote" for every candidate line that could pass through it. The local maxima (peaks) in the accumulator array represent the lines with the strongest consensus.

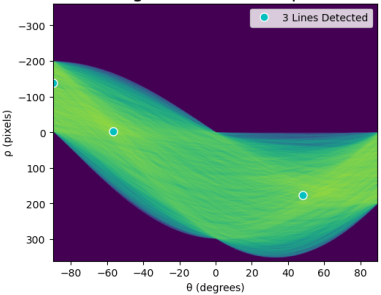
Noisy lines



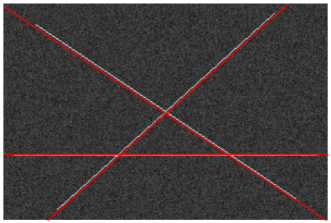
Canny



Hough accumulator space



Found lines



The Hough Transform: Circles and Ellipses

Objective: Detect circular or elliptical structures by extending the accumulator voting principle to higher dimensions.

Methodology (Higher-Dimensional Voting):

1. **Parameterization:**
   - **Circles (3D Space):** Defined by $(x - x_c)^2 + (y - y_c)^2 = r^2$. The parameters are the center coordinates (x_c, y_c) and the radius r. The votes are not over sinusoidal curves but **expanding cones**.
   - **Ellipses (5D Space):** Requires the center (x_c, y_c), major and minor axes (a, b), and orientation angle θ.
2. **Accumulation:** For a known or assumed radius r, every edge pixel casts votes along a circle of radius r centered on itself in the (x_c, y_c) parameter space.
3. **The Dimensionality Challenge:** 3D (and especially 5D) accumulator arrays are highly memory and compute-intensive. Practical algorithms often leverage edge gradient directions to drastically restrict the voting space.

RANSAC: RANdom SAmple Concensus

The Hough transform finds *all* lines at once. **RANSAC** fits *one* model robustly, even when most data points are outliers.

Algorithm (for fitting a line):

1. Pick 2 random points $\rightarrow$ fit a line through them.
2. Count how many other points lie within distance δ of this line (*inliers*).
3. Repeat many times; keep the line with the most inliers.
4. Re-fit using *all* inliers of the best model.

	Hough Transform	**RANSAC**
Finds	multiple lines at once	one model at a time
Deterministic	yes	no (random sampling)
Generalises to	lines, circles	any model (planes, etc.)

Tips

Use **Hough** when you expect *several* lines and want to find them all.
Use **RANSAC** when you want to fit a *single* model robustly despite heavy outliers.

RANSAC: Required Iterations Table ($p = 0.99$)

Number of samples k needed for 99% confidence that at least one all-inlier sample is drawn:

$$k = \frac{\log(1-p)}{\log(1-w^n)} \qquad p = 0.99$$

where:

- p = desired success probability (typically 0.99)
- w = estimated inlier fraction
- n = minimum sample size (e.g. 2 for a line)

Sample size n	Proportion of outliers						
	5%	10%	20%	25%	30%	40%	50%
2	2	3	5	6	7	11	17
3	3	4	7	9	11	19	35
4	3	5	9	13	17	34	72
5	4	6	12	17	26	57	146
6	4	7	16	24	37	97	293
7	4	8	20	33	54	163	588
8	5	9	26	44	78	272	1177

Reading the table: for line fitting ($n = 2$) with 50% outliers, only 17 iterations suffice. For homography estimation ($n = 4$) with 50% outliers, you need 72.

Key takeaway: RANSAC is remarkably efficient for low-n models, but cost grows exponentially with sample size and outlier fraction.

Lecture Summary

1. Edge Detection	2. Morphology	3. Corners	4. Lines & Models
Core Concept Gradients highlight intensity changes (boundaries). Canny cleans raw gradients into thin edges.	**Core Concept** Clean up binary masks using structuring elements to grow/shrink foreground.	**Core Concept** Harris: Find points where the gradient changes in *multiple* directions.	**Core Concept** Vote in parameter space (Hough) or iteratively sample (RANSAC).
Key Math Grad Mag: $\sqrt{I_x^2 + I_y^2}$ Roberts, Prewitt and Sobel kernels	**Key Logic** Dilation, Erosion Opening: Remove outside noise Closing: Fill inside holes Morphological edge detection	**Key Math** $R = \det(M) - k(\operatorname{tr} M)^2$ (Structure Tensor)	**Key Math** $x\cos\theta + y\sin\theta = \rho$ (Polar lines) Hough accumulator space

Learning Goals — Self-Check

Can you now...

1. ✓ **Explain** what an image gradient is and how it relates to edges.
2. ✓ **Write** the Sobel kernels from memory and apply them to a patch.
3. ✓ **Name** the four steps of the Canny pipeline and explain what each one does.
4. ✓ **Distinguish** dilation, erosion, opening, and closing — and know when to use each.
5. ✓ **Clean up** a noisy binary mask using morphological operations in code.
6. ✓ **Explain** why corners are more distinctive than edges, and what the Harris response R tells you.
7. ✓ **Describe** how the Hough transform uses voting to detect lines.

Thank you for attending!

Questions?

hugo.guillenramirez@unibe.ch

Next week:
Lecture 6: Image Segmentation with Prof. Dr. Mauricio Reyes

References

1. Gonzalez, R.C. & Woods, R.E. (2018). Digital Image Processing (4th ed.). Pearson.
2. Burger, W. & Burge, M. (2016). Digital Image Processing: An Algorithmic Introduction Using Java (2nd ed.). Springer.

Appendix

Bonus math!

Continuous Derivative & Discrete Approximation

Continuous partial derivative:

$$\frac{\partial f}{\partial x} = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon, y) - f(x, y)}{\varepsilon}$$

This is **linear and shift-invariant**, so it must be the result of a convolution.

Discrete approximation (finite difference):

$$\frac{\partial f}{\partial x} \approx \frac{f(x_{n+1}, y) - f(x_n, y)}{\Delta x}$$

which corresponds to the convolution kernel $\begin{bmatrix} -1 & 1 \end{bmatrix}$.

Why this matters:

Because differentiation *is* convolution, and convolution is **associative**:

$$D * (G * I) = (D * G) * I$$

So we can pre-compute the combined "smoothing + derivative" kernel once and apply it in a single pass.

All edge filters are finite-difference approximations:

- Roberts: 2×2 diagonal differences
- Prewitt: uniform 3×3 differences
- Sobel: weighted 3×3 differences
- Laplacian: 2nd-order differences (∇^2)

Derivative of Gaussian: Full Kernel Formula

From associativity of convolution:

$$\frac{\partial}{\partial x}\big(G_\sigma * I\big) = \underbrace{\frac{\partial G_\sigma}{\partial x}}_{\text{DoG kernel}} * I$$

DoG kernel (x-direction):

$$\frac{\partial G_\sigma}{\partial x} = -\frac{x}{\sigma^2}\, G_\sigma(x, y)$$

where $G_\sigma(x, y) = \frac{1}{2\pi\sigma^2}\, e^{-(x^2+y^2)/(2\sigma^2)}$.

The y-direction kernel is obtained by replacing x with y:

$$\frac{\partial G_\sigma}{\partial y} = -\frac{y}{\sigma^2}\, G_\sigma(x, y)$$

DC Bias Correction:

In continuous math a derivative kernel sums to exactly zero. Discretisation onto a finite pixel grid introduces a tiny numerical offset.

Fix: subtract the kernel mean so the sum is forced back to zero. This prevents a constant bias on flat regions.

Scale parameter σ:

σ	Effect
Small	fine detail, more noise
Large	coarser edges, less noise

Kernel half-width is typically $\lceil 3\sigma \rceil$ pixels.

Laplacian-of-Gaussian: Full Formula

Laplacian (sum of 2nd partial derivatives):

$$\nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} \qquad \text{Kernel: } \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

LoG kernel (Marr–Hildreth, 1980):

$$\mathrm{LoG}_\sigma(x, y) = -\frac{1}{\pi\sigma^4}\left(1 - \frac{x^2 + y^2}{2\sigma^2}\right) e^{-(x^2+y^2)/2\sigma^2}$$

Edge criterion:

$$\text{Edges} = \text{zero crossings of } (\mathrm{LoG}_\sigma * I)$$

LoG $\approx$ Difference of Gaussians:

For two close scales σ and $k\sigma$ ($k \approx 1.6$):

$$\mathrm{LoG}_\sigma \approx \frac{G_{k\sigma} - G_\sigma}{\sigma^2(k-1)}$$

This is the approximation used in SIFT.

Properties:

- Rotationally symmetric (isotropic)
- Gives **closed contours** (unlike Canny)
- Harder to tune than Canny
- Shape: "Mexican hat" (sombrero)

Morphological Operations: Set-Theoretic Definitions

Dilation (A = image, B = structuring element):

$$A \oplus B = \left\{ z \,\middle|\, (\hat{B})_z \cap A \neq \emptyset \right\}$$

The set of all points where the reflected SE, shifted to z, overlaps at least one foreground pixel.

Erosion:

$$A \ominus B = \left\{ z \,\middle|\, (B)_z \subseteq A \right\}$$

The set of all points where the SE, shifted to z, fits entirely inside the foreground.

Duality: $(A \ominus B)^c = A^c \oplus \hat{B}$

Erosion of foreground = dilation of background (with reflected SE).

Opening:

$$A \circ B = (A \ominus B) \oplus B$$

Erosion then dilation. Removes structures smaller than B.

Closing:

$$A \bullet B = (A \oplus B) \ominus B$$

Dilation then erosion. Fills holes smaller than B.

Morphological Gradient:

$$\text{grad}(A) = (A \oplus B) - (A \ominus B)$$

Key Properties:

- Opening is **anti-extensive**: $A \circ B \subseteq A$
- Closing is **extensive**: $A \subseteq A \bullet B$
- Both are **idempotent**: $(A \circ B) \circ B = A \circ B$

Structure Tensor & Harris Corner Response: Full Math

Structure Tensor (Second-Moment Matrix):

For a Gaussian-weighted window w around each pixel:

$$M = \sum_{(u,v)\in W} w(u,v) \begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix} = \begin{pmatrix} S_{xx} & S_{xy} \\ S_{xy} & S_{yy} \end{pmatrix}$$

Eigenvalue interpretation ($\lambda_1 \geq \lambda_2$):

- Both small $\rightarrow$ flat region
- One large, one small $\rightarrow$ edge
- Both large $\rightarrow$ **corner**

Harris Response (avoids explicit eigendecomposition):

$$R = \underbrace{\det(M)}_{\lambda_1\lambda_2} - k \underbrace{\left(\mathrm{tr}(M)\right)^2}_{\lambda_1+\lambda_2}$$

with $k \approx 0.04$–0.06.

Why this works:

Condition	R value
λ_1, λ_2 small	$\lvert R\rvert \approx 0$ (flat)
$\lambda_1 \gg \lambda_2$	$R \ll 0$ (edge)
λ_1, λ_2 large	$R \gg 0$ (corner)

Detection Pipeline:

1. Compute I_x, I_y (Sobel or DoG).
2. Form M with Gaussian weighting (σ_w).
3. Compute R at every pixel.
4. Threshold R and apply NMS.

$\det(M) = \lambda_1\lambda_2$ *is large only when* ***both*** *eigenvalues are large.* $\mathrm{tr}(M) = \lambda_1 + \lambda_2$ *penalises edge cases where one eigenvalue dominates.*

Harris Corner: Full SSD-to-Matrix Derivation (1/2)

Goal: measure how much a patch changes when shifted by (u, v).

Sum of Squared Differences (SSD):

$$\mathrm{SSD}(u, v) = \sum_{x,y} \big[I(u + x,\ v + y) - I(x, y)\big]^2$$

Taylor expansion (1st order) of $I(u + x,\ v + y)$:

$$I(u + x,\ v + y) \ \approx\ I(x, y) \ +\ u\, I_x(x, y) \ +\ v\, I_y(x, y)$$

Substituting into SSD:

$$\mathrm{SSD}(u, v) \approx \sum_{x,y} \big[u\, I_x(x, y) + v\, I_y(x, y)\big]^2$$

$$= \sum_{x,y} \big[u^2 I_x^2 + v^2 I_y^2 + 2uv\, I_x I_y\big]$$

$$= \sum_{x,y} \begin{pmatrix} u & v \end{pmatrix} \begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}$$

Pulling the constant shift (u, v) out of the sum:

$$\mathrm{SSD}(u,v) \approx \begin{pmatrix} u & v \end{pmatrix} \underbrace{\begin{pmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{pmatrix}}_{C = \text{structure tensor}} \begin{pmatrix} u \\ v \end{pmatrix}$$

Why C is a covariance matrix:

Let $D = \begin{pmatrix} I_{x_1} & I_{y_1} \\ \vdots & \vdots \\ I_{x_n} & I_{y_n} \end{pmatrix}$ be the data matrix of gradients.

Then $C = D^T D$, which is exactly the (un-normalised) covariance of the gradient vectors (assuming zero-mean, since derivative filters are zero-sum).

Spectral decomposition:

Because $C = D^T D$ is symmetric positive semi-definite, by the **spectral theorem** it has real eigenvalues $\lambda_1 \geq \lambda_2 \geq 0$ and orthogonal eigenvectors:

$$C = R^T \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} R$$

The eigenvalues give the axis lengths of the gradient distribution ellipse; the eigenvectors give its orientation.

Key insight: $\mathrm{SSD}(u,v) = \mathbf{u}^T C \mathbf{u}$ is a quadratic form. Its iso-contours are ellipses whose shape is governed by λ_1, λ_2.

Eigenvalue Problem: Solving the 2×2 Case

Eigenvalue equation:

$$C\mathbf{v} = \lambda\mathbf{v} \quad \Longrightarrow \quad (C - \lambda I)\mathbf{v} = \mathbf{0}$$

Non-trivial solutions exist iff the **characteristic equation** holds:

$$\det(C - \lambda I) = 0$$

For the 2×2 structure tensor:

$$\det\begin{pmatrix} S_{xx} - \lambda & S_{xy} \\ S_{xy} & S_{yy} - \lambda \end{pmatrix} = 0$$

$$(S_{xx} - \lambda)(S_{yy} - \lambda) - S_{xy}^2 = 0$$

$$\lambda^2 - (S_{xx} + S_{yy})\lambda + (S_{xx}S_{yy} - S_{xy}^2) = 0$$

Recognise the coefficients:

Quantity	In terms of λ
$\mathrm{tr}(C)$	$= \lambda_1 + \lambda_2$
$\det(C)$	$= \lambda_1 \cdot \lambda_2$

Harris shortcut (avoids solving the quadratic):

$$R = \det(C) - k\,[\mathrm{tr}(C)]^2$$

This uses only det and tr, both computable from the matrix entries directly — no eigendecomposition needed.

Computing det *and* tr *costs* $O(1)$ *per pixel; eigendecomposition costs* $O(n^3)$ *in general.*

Hough Transform for Lines: Parameterisation Math

The Cartesian Problem:

A line is traditionally $y = mx + c$. However, the parameter space (m, c) is unbounded because vertical lines have $m \to \infty$.

Polar Parameterisation:

We represent a line by its normal vector from the origin. The line equation becomes:

$$x\cos\theta + y\sin\theta = \rho$$

- ρ: Perpendicular distance from origin to the line.
- θ: Angle of the normal vector with the x-axis.

Parameter Bounds:

- $\theta \in [-\frac{\pi}{2}, \frac{\pi}{2})$ (or $[0, \pi)$)
- $\rho \in [-D, D]$, where $D = \sqrt{\text{width}^2 + \text{height}^2}$ is the image diagonal.

Point-to-Curve Mapping:

For a single fixed edge pixel (x_i, y_i), there is an infinite family of lines that pass through it. If we treat (x_i, y_i) as constants and (θ, ρ) as variables, we get:

$$\rho(\theta) = x_i\cos\theta + y_i\sin\theta$$

This traces a **sinusoid** in the (θ, ρ) parameter space.

Collinearity Condition:

If two points (x_1, y_1) and (x_2, y_2) lie on the same line defined by (θ^*, ρ^*), their respective sinusoids will intersect exactly at (θ^*, ρ^*):

$$\begin{cases} x_1\cos\theta^* + y_1\sin\theta^* = \rho^* \\ x_2\cos\theta^* + y_2\sin\theta^* = \rho^* \end{cases}$$

Hough Transform for Lines: Accumulator & Pipeline

The Accumulator Array (A):

To compute this computationally, we quantise the continuous (θ, ρ) space into a 2D grid matrix A with step sizes $\Delta\theta$ and $\Delta\rho$.

Voting Mechanism: For each edge pixel (x_i, y_i) and for each discrete angle step θ_j:

$$\rho_k = \text{round}\left(\frac{x_i \cos\theta_j + y_i \sin\theta_j}{\Delta\rho}\right) \cdot \Delta\rho$$

We cast a vote by incrementing the bin:

$$A[\theta_j, \rho_k] \leftarrow A[\theta_j, \rho_k] + 1$$

Resolution Trade-offs:

- **Too fine (Δ small):** Sinusoids might miss each other due to discretisation noise; peaks become fragmented.
- **Too coarse (Δ large):** Distinct nearby lines merge into a single bin; false positives increase.

Detection Pipeline:

1. Compute the **edge map** (e.g., using Canny).
2. Initialize the accumulator $A[\theta, \rho]$ to all zeros.
3. For every pixel (x, y) where $\text{Edge}(x, y) > 0$:
   - Loop over all θ_j.
   - Compute corresponding ρ_k.
   - Increment $A[\theta_j, \rho_k]$.
4. Find **local maxima** in A that exceed a threshold T.
5. Apply **Non-Maximal Suppression** to avoid thick clusters of votes.
6. Map the surviving (θ, ρ) pairs back to the image space.

Optimization Hint: If the gradient angle ϕ is known (e.g., from Sobel), we only vote around $\theta \approx \phi$, massively reducing computation.

RANSAC: Iteration Count & Comparison

How many iterations k?

$$k = \frac{\log(1-p)}{\log(1-w^n)}$$

where:

- p = desired success probability (typically 0.99)
- w = estimated inlier fraction
- n = minimum sample size (e.g. 2 for a line)

Example:

$w = 0.5,\ n = 2$:

$$k = \frac{\log(0.01)}{\log(0.75)} \approx 16 \text{ iterations}$$

Even with only 50% inliers, 16 samples suffice for 99% confidence.

Hough vs. RANSAC:

	Hough	RANSAC
Finds	all lines	one model
Deterministic	yes	no
Outlier handling	moderate	excellent
Generalises to	lines, circles	any model
Speed	$\propto$ resolution	usually fast

Rule of thumb:

- **Hough** when you expect *several* lines/shapes and want to find them all.
- **RANSAC** when fitting a *single* model robustly despite heavy outliers.