MCS4304 · Computer Vision

All notes for open-book use · Topics 1–6 · built 2026-11-27

SUM	Module summary big picture · each topic in four lines · formula sheet · tables · whole-pipeline example · cross-topic questions · mixed MCQs	**2**
LO1	Introduction to Computer Vision summary · deep summary · exam questions · practice (10) · MCQs + key	**12**
LO2	Visual perception & image sensing summary · deep summary · exam questions · practice (10) · MCQs + key	**24**
LO3	Arithmetic, logic & geometric operations summary · deep summary · exam questions · practice (10) · MCQs + key	**37**
LO4	Image noise & filtering summary · deep summary · exam questions · practice (10) · MCQs + key	**51**
LO5	Edge detection summary · deep summary · exam questions · practice (10) · MCQs + key	**65**
LO6	Hough transform summary · deep summary · exam questions · practice (10) · MCQs + key	**79**

Source: the lecture slides (each part cites "Topic N · slide n"); items marked **added** are study aids beyond the slides. Each section: summary → deep summary (step by step) → exam questions with answers → 10 practice questions → MCQs with an answer key.

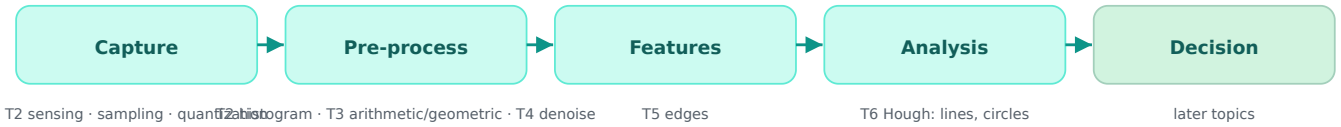
SUM MCS4304 Computer Vision — revision summary

All six lecture topics on one page: where each one sits in the pipeline, the key points of each, a formula sheet, a noise-to-filter table, and the likely exam questions with links to the worked answers.

Download all notes (PDF, A4) summary + contents, then every topic with answers and an MCQ key

The big picture

Where each topic sits in the CV pipeline (Topic 1)



raw light → clean pixels → edges → whole shapes

Topics 2–6 walk along the pipeline from Topic 1, from how light becomes pixels to finding whole shapes in them.

Each topic in four lines

TOPIC 1 · LO1

Introduction to Computer Vision ›

- CV = **image → meaning**; IP = image → image; CG = model → image.
- Pipeline: capture → pre-process → features → analysis → decision.
- Digital image = I(x, y) with x, y and I finite and discrete; pixel; grey level.
- 5 resolutions: pixel · spatial · spectral · temporal · radiometric. RGB (additive) · CMY (subtractive) · HSI/HSV.

TOPIC 2 · LO2

Visual perception & image sensing ›

- Light = visible EM; colour = reflected light. Radiance · luminance · brightness.
- Single sensor · strip · array. **f = i · r**.
- **Sampling** = coordinates; **quantization** = intensity; quantization error.
- Histogram $h(r_k) = n_k$. Contrast stretching (5th/95th %) · equalisation $s_k = \Sigma n_i/n$.

TOPIC 3 · LO3

Arithmetic, logic & geometric ops ›

- Pixel by pixel: add min(255, ·) · subtract max(0, ·) / |A−B| · multiply = mask · divide A/(B+ε) · average.
- Logical AND/OR/NOT/XOR on binary masks.
- Scale · rotate · shear · reflect · translate; **homogeneous coordinates** → affine chains.
- Inverse mapping + nearest neighbour / bilinear; aliasing (jaggies, moiré).

TOPIC 4 · LO4

Noise & filtering ›

- g = f + η. Gaussian (sensor) · Poisson (shot, mean = var = λ) · salt & pepper (impulse) · speckle (g = f·η).
- Mean blurs edges; Gaussian weights the centre; **median** beats salt & pepper and keeps edges.
- Convolution = flipped-kernel cross-correlation (same if symmetric).
- Borders: zero · replicate · reflect · valid · truncated.

TOPIC 5 · LO5

Edge detection ›

- Edge = significant local intensity change.

TOPIC 6 · LO6

Hough transform ›

- Edges are local; Hough finds global lines despite gaps and noise.

- $\nabla I = [I_x, I_y]$, $G = \sqrt{(I_x^2+I_y^2)}$, $\theta = \tan^{-1}(I_y/I_x)$; edge $\perp$ gradient.
- Prewitt $[1\ 1\ 1]^T$ vs Sobel $[1\ 2\ 1]^T$ smoothing; threshold T.
- Laplacian zero crossings · LoG $= \nabla^2(G_\sigma * I)$ · **Canny**: smooth → gradient → NMS → double threshold → hysteresis.

- Polar form **x cosθ + y sinθ = d** (handles vertical lines).
- Point → sinusoid; collinear points → curves meet at (d, θ).
- Accumulator A(d, θ) votes → peak (≥ T) → line. Extensions: circles · probabilistic.

Formula sheet

TOPIC	WHAT	FORMULA / MASK
T2	Image formation	$f(x,y) = i(x,y) \cdot r(x,y)$
T2	Histogram	$h(r_k) = n_k \cdot p(r_k) = n_k/n$
T2	Contrast stretching	$I_{out} = (I_{in} - c)(b - a)/(d - c) + a$
T2	Equalisation	$s_k = \Sigma_{i=0..k}\ n_i/n \cdot new = round(s_k \cdot m)$
T3	Add / subtract	$\min(255, A + B) \cdot \max(0, A - B) \cdot \lvert A - B \rvert$
T3	Divide / average	$A/(B + \varepsilon) \cdot (1/N)\ \Sigma\ I_i$
T3	Rotation	$[\cos\theta\ -\sin\theta;\ \sin\theta\ \cos\theta]$
T3	Reflection about a line at θ	$[\cos2\theta\ \sin2\theta;\ \sin2\theta\ -\cos2\theta]$
T3	Translation (homogeneous)	$[1\ 0\ t_x;\ 0\ 1\ t_y;\ 0\ 0\ 1]$
T3	Nearest neighbour	$I'(x,y) = I(\text{round } x,\ \text{round } y)$
T4	Noise models	$g = f + \eta,\ \eta \sim N(\mu, \sigma^2) \cdot K \sim Poisson(\lambda) \cdot speckle\ g = f \cdot \eta$
T4	Mean / Gaussian 3 × 3	$(1/9)[1\ 1\ 1;\ 1\ 1\ 1;\ 1\ 1\ 1] \cdot (1/16)[1\ 2\ 1;\ 2\ 4\ 2;\ 1\ 2\ 1]$
T4	Cross-corr. / convolution	$\Sigma\ f[m+i,\ n+j]\ h[i,j] \cdot \Sigma\ f[m-i,\ n-j]\ h[i,j]$
T5	Gradient	$G = \sqrt{(I_x^2 + I_y^2)} \cdot \theta = \tan^{-1}(I_y/I_x)$
T5	Sobel G_x	$[-1\ 0\ 1;\ -2\ 0\ 2;\ -1\ 0\ 1]$
T5	Laplacian (4-nbr)	$[0\ 1\ 0;\ 1\ -4\ 1;\ 0\ 1\ 0] \cdot$ edge if $L_1L_2 < 0$ and $\lvert L_1 - L_2 \rvert \geq T$
T5	LoG	$\nabla^2(G_\sigma * I)$
T6	Line, polar form	$x\ \cos\theta + y\ \sin\theta = d$

Noise → filter (Topic 4)

NOISE	MODEL	START WITH
Gaussian (sensor)	additive, $N(\mu, \sigma^2)$	Gaussian or mean filter; more advanced if fine detail matters
Poisson (shot)	$K \sim Poisson(\lambda)$, mean = var = λ	Gaussian or mean filter, then Wiener
Salt & pepper	impulse, isolated extremes	**Median**
Speckle	multiplicative $g = f \cdot \eta$	Lee or Frost filters

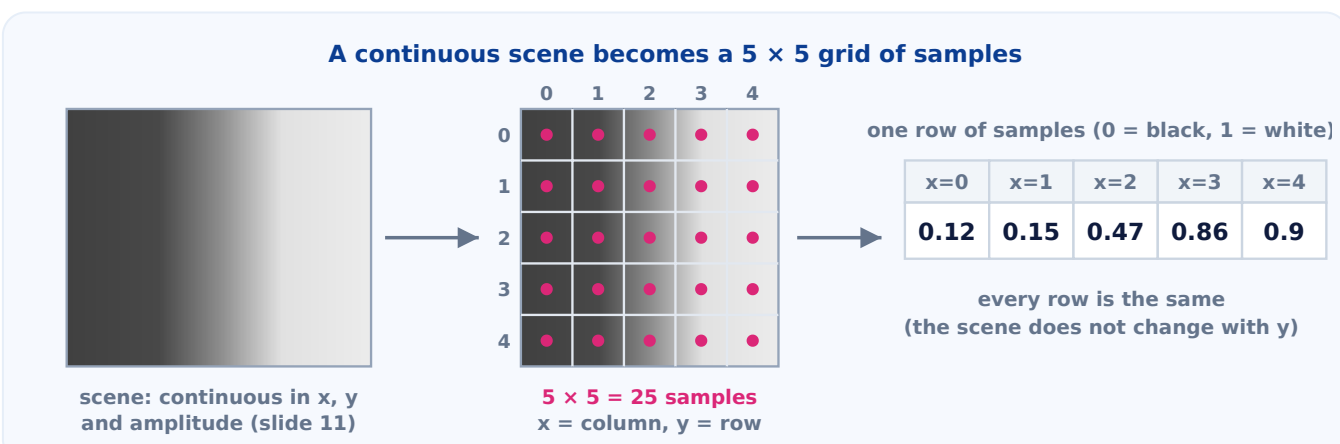
Edge detectors compared (Topic 5)

DETECTOR	LOOKS FOR	GOOD	WEAK
Prewitt / Sobel	gradient peak	simple; Sobel a little more noise-robust	thick edges, noise
Laplacian	zero crossing	one isotropic mask	very noisy, no orientation
LoG	zero crossing after Gaussian	noise handled by σ	σ to choose
Canny	gradient peak + NMS + hysteresis	thin, connected, low-noise	two thresholds to tune

One worked example through the whole pipeline

One tiny grey image with one straight edge, carried from capture to a line equation: Topic 2 → Topic 4 → Topic 5 → Topic 6. Every number below was computed and checked. Each step cites its slides (Topic 2 uses the PDF page numbers, as on the LO2 page). The example image and the choices of T are ours; other parts marked **added** are study aids, not lecture content.

1 · Sample the scene (T2) Topic 2 · slides 10–13, 19



The sensor output is **continuous** in x, y and amplitude, so making a digital image takes two steps: sampling, then quantization (slides 10–11). **Sampling digitises the coordinates**: lay a grid and take one value per cell. Slide 12 uses 10 × 10 = 100 samples; we use **5 × 5 = 25** so the numbers stay small.

Our scene is dark on the left and bright on the right, the same in every row. One row of samples: **0.12, 0.15, 0.47, 0.86, 0.90**. The middle sample (0.47) is where the boundary falls inside a cell. The result is the image as a 2-D array (slide 19).

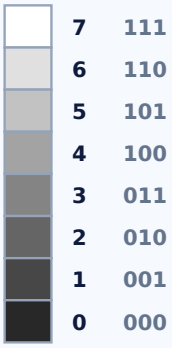
added We write pixels as (x, y) with x = column and y = row, origin top-left, as Topic 6 slide 11 draws it. Slide 19 lists f(row, column), so read its indices the other way round.

→ LO2 deep part

2 · Quantize to 3 bits (T2)

Topic 2 · slides 14–18 · Topic 1 · slide 15

Quantize to 3 bits = 8 levels, then read the error



level · code

x	0	1	2	3	4
v × 7	0.84	1.05	3.29	6.02	6.30
level	1	1	3	6	6
code	001	001	011	110	110
error	0.16	0.05	0.29	0.02	0.30

nearest level = vertical proximity (slide 17)

	0	1	2	3	4
0	1	1	3	6	6
1	1	1	3	6	6
2	1	1	3	6	6
3	1	1	3	6	6
4	1	1	3	6	6

the 5 × 5 image

Quantization digitises the amplitude (slides 14–16). Slide 17: divide black-to-white into **equal parts** and give each sample the **nearest** discrete value. We keep **8 levels, 0–7** (Topic 1 slide 15 shows an 8-level strip), which need **3 bits**: codes 000…111, like the binary labels on slide 18.

Step by step: multiply by 7, take the nearest level. 0.84 → 1, 1.05 → 1, 3.29 → 3, 6.02 → 6, 6.30 → 6. The gaps are the **quantization error** (slide 18): 0.16, 0.05, 0.29, 0.02, 0.30, all at most half a step. Every row of the image is now **1 1 3 6 6**.

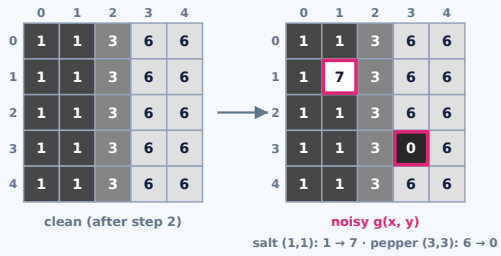
added “Multiply by 7 and round” is our way of writing slide 17’s nearest level; 0 = black and the top level = white as on Topic 2 slide 23.

→ LO2 deep part

3 · Salt-and-pepper noise hits two pixels (T4)

Topic 4 · slides 4, 14–16, 20

Two pixels hit by impulse noise



A transmission or storage error sets two pixels to extremes (slides 14–15): **salt** at (1, 1) goes 1 → **7** (white, the top level) and **pepper** at (3, 3) goes 6 → **0** (black). This is **impulse noise**; the other 23 pixels are untouched. We now only see g(x, y), and the task is to estimate f(x, y) without losing detail (slide 4).

Why it matters (slide 20): random intensity changes are picked up as **false edges**.

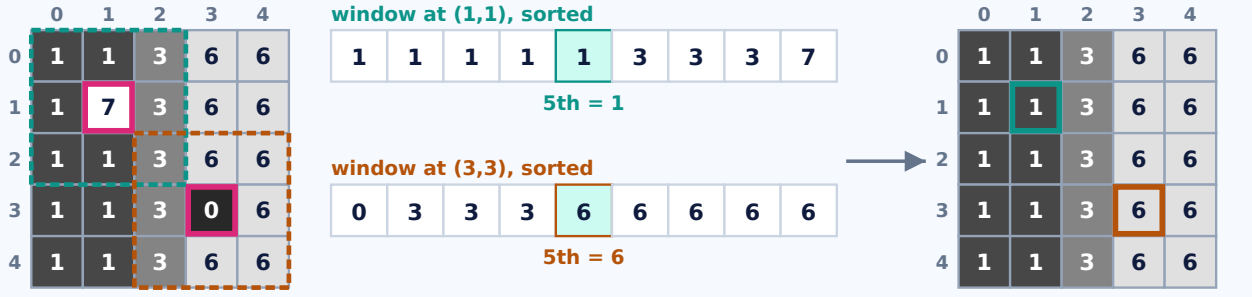
added Checked: running step 5 on this noisy image with the same T = 16 loses the whole true edge (G at x = 2 drops to 8–15.2) and marks two false edge pixels at (3, 2) and (3, 4) with G = 17.

→ LO4 deep part

4 · Remove it with a 3 × 3 median (T4)

Topic 4 · slides 26, 31-33, 38

3 × 3 median: sort the window, keep the 5th value



The median filter **sorts the 9 values and keeps the middle one** (slide 31). At the salt pixel (1, 1): window 1 1 3 / 1 7 3 / 1 1 3 → sorted **1 1 1 1 1 3 3 3 7** → 5th = **1**. At the pepper pixel (3, 3): 3 6 6 / 3 0 6 / 3 6 6 → **0 3 3 3 6 6 6 6 6** → **6**. The extremes sit at the ends of the list and never reach the middle.

On the edge pixel (2, 2) the window is 1 3 6 three times → 1 1 1 3 3 3 6 6 6 → **3**, so the edge stays: the median **preserves edges** (slide 31). The output is exactly the clean image 1 1 3 6 6 again. Compare a 3 × 3 mean at (1, 1): 21/9 = 2.33, the salt is smeared, not removed (slide 26). Slide 33: salt and pepper → median.

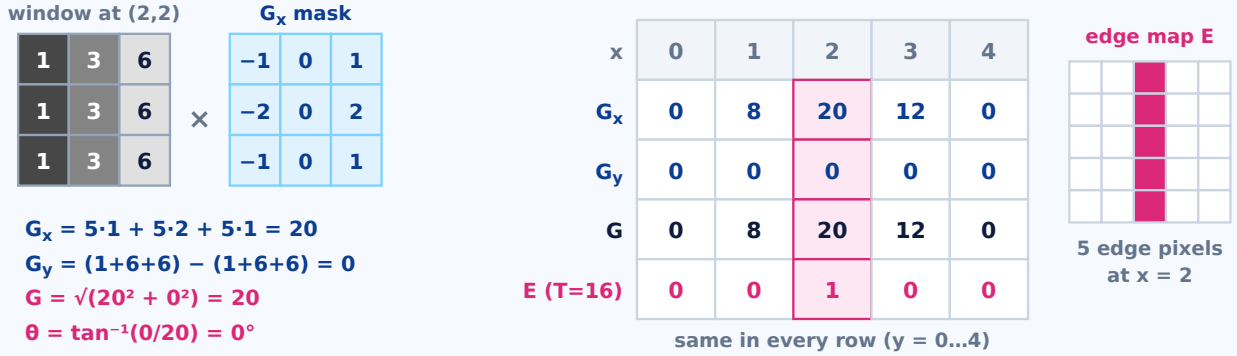
Border pixels use **replication padding** (slide 38). **added** Getting the clean image back exactly is a property of this simple example.

→ LO4 deep part

5 · Sobel gradient and a threshold (T5)

Topic 5 · slides 13-14, 17-18, 22

Sobel at (2, 2), then every column, then threshold T = 16



Sobel = the derivative mask [−1 0 1] across × the smoothing vector $[1\ 2\ 1]^T$ down; G_y uses the transpose (slide 17). At (2, 2) the window rows are all 1 3 6, so $\mathbf{G_x} = (6 − 1)·1 + (6 − 1)·2 + (6 − 1)·1 = \mathbf{20}$ and $\mathbf{G_y} = 0$ (the row below equals the row above).

Slide 13: $\mathbf{G = √(G_x² + G_y²) = 20}$, $\mathbf{θ = \tan^{-1}(0/20) = 0°}$: the gradient points along +x (dark → bright) and the edge runs perpendicular, at 90°, a vertical edge (slide 14). The columns give G = 0, 8, 20, 12, 0. With a **manual threshold T = 16** (slides 18, 22), E = 1 only at x = 2: five edge pixels (2, 0) … (2, 4).

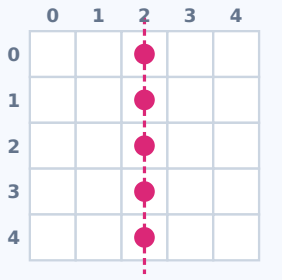
added T = 16 is our choice. Masks are laid on the image without flipping (cross-correlation, Topic 4 slide 35) and borders are replicated.

→ LO5 deep part

6 · Hough: vote, find the peak, write the line (T6)

Topic 6 · slides 10, 13, 18–25, 27

Each edge pixel votes; the peak gives the line



edge pixels (2, 0)…(2, 4)

d = x cosθ + y sinθ for each pixel (x = 2)

θ \ y	y=0	y=1	y=2	y=3	y=4
0°	2	2	2	2	2
45°	1.41	2.12	2.83	3.54	4.24
90°	0	1	2	3	4
135°	−1.41	−0.71	0	0.71	1.41

20 votes over 16 (d, θ) cells

peak A(2, 0°) = 5
other 15 cells = 1
5 ≥ T = 3 → a line

x cos0° + y sin0° = 2
→ x = 2

Each edge pixel (x_0, y_0) votes for every line through it, $x_0 \cos\theta + y_0 \sin\theta = d$ (slide 13), at θ = 0°, 45°, 90°, 135° as in the slide example (slide 21). With x = 2: at **0° every pixel gives d = 2**; at 45° d = 1.41 … 4.24, at 90° d = y, at 135° d = −1.41 … 1.41, all different.

Accumulator A(d, θ) (slide 18): 5 pixels × 4 angles = 20 votes over 16 cells. **A(2, 0°) = 5**, every other cell 1. It is max(A) (slide 25) and passes a threshold such as T = 3 (slide 27). Rebuild the line (slide 23): x cos0° + y sin0° = 2 → **x = 2**, the vertical boundary. This is the line y = mx + c cannot write (slide 9), and the reason for the polar form (slide 10).

added The Sobel θ = 0° at the edge pixels equals the Hough θ of the line: both describe the direction perpendicular to the line. T = 3 is our choice.

→ LO6 deep part

Cross-topic exam questions

Questions that need two or more topics at once. Try each first, then reveal the answer. Answers cite slides; parts marked added are our reasoning beyond the slides.

X1. An image has salt-and-pepper noise and you need Sobel edges. Which filter do you run first, and why not a mean or Gaussian filter?

Answer: Run a **median** filter first. Salt and pepper is isolated extreme values from transmission or storage errors (Topic 4 · slides 14–15), and an edge detector reacts to any abrupt local change (Topic 5 · slides 3, 8), so each impulse becomes a **false edge** (Topic 4 · slide 20). The **mean** is pulled by outliers and blurs edges because it averages across the boundary (Topic 4 · slide 26); the **Gaussian** does not reliably remove isolated extreme pixels (Topic 4 · slide 30). The **median** sorts and keeps the middle value, is particularly effective against salt and pepper and preserves edges better (Topic 4 · slides 31, 33). Then Sobel and the threshold T (Topic 5 · slides 17–18). In the worked example above: the mean at the salt pixel gives 2.33 instead of 1, the median gives 1, and without filtering the true edge is lost and two false edge pixels appear.
LO4 › · **LO5 ›**

X2. Why does the Hough transform need an edge map first, and what does it add that the edge detector cannot give?

Answer: Step 1 of the Hough summary is “obtain edge pixels” (Topic 6 · slide 24): the voters are the 1s of a binary edge image (Topic 6 · slides 19–20), produced by thresholding the gradient, E = 1 where G ≥ T (Topic 5 · slide 18). An edge detector only gives **local** facts: where intensity changes and which pixels are likely boundaries (Topic 6 · slide 3). It does not say whether edge pixels belong to the **same line**, or whether a line is there despite gaps, noise or occlusion (Topic 6 · slide 5). Hough adds that **global** grouping: every edge pixel votes in A(d, θ) and the peak is the line (Topic 6 · slides 18, 25). Two different thresholds appear: T on gradient magnitude (Topic 5 · slide 18) and T on votes (Topic 6 · slide 27). added Thick Sobel edges (Topic 5 · slide 30) put several parallel columns into the vote and spread the peak; thin Canny edges (Topic 5 · slide 33) give a sharper one. **LO5 ›** · **LO6 ›**

X3. How do sampling and quantization choices at capture affect edge detection later in the pipeline?

Answer: **Sampling** sets how many pixels describe the scene (Topic 2 · slides 11–12): more samples, finer detail. Derivatives are approximated by **differences between neighbouring samples** (Topic 5 · slide 15), so coarse sampling squeezes an edge into one or two pixels, and sampling too low for fine detail gives **aliasing** such as jagged edges (Topic 3 · slide 32). **Quantization** sets the size of the intensity steps (Topic 2 · slides 17–18; Topic 1 · slides 15–16 show what few levels do). Worked on the pipeline row 0.12, 0.15, 0.47, 0.86, 0.90: with **8 levels** it becomes 1 1 3 6 6 and Sobel G_x = 0 8 20 12 0, one clear peak at x = 2. With **2 levels** (1 bit) it becomes 0 0 0 1 1 and G_x = 0 0 4 4 0: the in-between pixel is gone, two columns tie, and with T = 16 nothing is marked at all. added T is measured in grey-level units, so it can only be chosen once the bit depth is fixed. **LO2 ›** · **LO3 ›** · **LO5 ›**

X4. Images come from a high-ISO, low-light camera. Name the noise model, the filter and the edge detector you would chain, and justify each link.

Answer: High-ISO and dark images carry **sensor noise**, modelled as additive Gaussian η ~ N(μ, σ²) (Topic 4 · slides 6, 8); low light also brings **shot noise**, K ~ Poisson(λ) (Topic 4 · slides 10–11). For both, start with a **Gaussian** (or mean) filter (Topic 4 · slides 33–34). Do not use the plain **Laplacian**: it is noise-sensitive (Topic 5 · slide 28). Use **LoG** = $\nabla^2(G_\sigma * I)$, which smooths first (Topic 5 · slide 29), or **Canny**, whose step 1 is the same Gaussian smoothing (Topic 5 · slides 31–32) with the kernel from Topic 4 · slide 29; non-maximum suppression thins the edges and double threshold + hysteresis keep weak edges only when connected to strong ones (Topic 5 · slides 33–35). added If the scene is still, averaging N frames (Topic 3 · slide 12) lowers the noise before any of this. **LO3 ›** · **LO4 ›** · **LO5 ›**

X5. An image is rotated (a geometric transform) before edge detection and Hough. How do the interpolation choice and the rotation itself show up in the edges and in the accumulator?

Answer: Rotation (Topic 3 · slide 20) sends pixels to non-integer positions, so the output is built by **inverse mapping** plus interpolation (Topic 3 · slides 25, 27–28). **Nearest neighbour** keeps the original intensities but gives **jagged edges** and aliasing (Topic 3 · slide 29): a straight boundary becomes a staircase, so the edge pixels scatter around the true line. **Bilinear** weights four neighbours (Topic 3 · slide 30): a smoother edge, but it averages across the boundary like a mean filter (Topic 4 · slide 26), so the gradient peak is lower and may drop under T (Topic 5 · slide 18). In Hough, scattered or broken edges split votes over nearby cells (Topic 6 · slide 26). added The rotation itself moves the peak: rotating by φ about the origin turns the line’s perpendicular by φ, so a line at (d, θ) shows up at (d, θ + φ). **LO3 ›** · **LO5 ›** · **LO6 ›**

X6. Should you apply contrast stretching or histogram equalisation before thresholding a gradient image? What happens to the threshold T?

Answer: Contrast stretching maps [c, d] to [a, b] with one scale factor (b − a)/(d − c) (Topic 2 · slide 28), so every **difference** between pixels, and so every derivative (Topic 5 · slides 15, 17), grows by the same factor. Worked on the pipeline row 1 1 3 6 6 stretched from [1, 6] to [0, 7] (factor 1.4): 0 0 2.8 7 7, stored as 0 0 3 7 7, and Sobel G_x = 0 12 28 16 0. With the old **T = 16 the edge becomes two pixels wide** (x = 2 and 3). So a fixed T must be re-chosen after any enhancement; percentile, histogram-analysis or Otsu thresholds (Topic 5 · slide 22) adjust to the new range on their own. Equalisation (Topic 2 · slides 31–33) is non-linear: it spreads the most frequent intensities, so differences change by different amounts in different places. added It can make small, noisy differences in flat regions larger, giving more false edges; enhancement is mainly for viewing. **LO2 ›** · **LO5 ›**

X7. A library's "convolution" is used to run Sobel. Does it matter whether it really convolves or cross-correlates? What changes in G, θ, the edge map and the Hough result?

Answer: Cross-correlation uses the kernel as it is (Topic 4 · slide 35); convolution flips it horizontally and vertically first (Topic 4 · slide 36); the two agree only for **symmetric** kernels (Topic 4 · slide 37). The mean and Gaussian kernels are symmetric (Topic 4 · slides 26, 29), so smoothing is unaffected. The Sobel masks are not: flipping [−1 0 1; −2 0 2; −1 0 1] gives its negative, so $G_x \to -G_x$ and $G_y \to -G_y$. **$G = \sqrt{(G_x^2 + G_y^2)}$ is unchanged** (Topic 5 · slide 13), so the edge map E (Topic 5 · slide 18) and the Hough votes, which use only edge-pixel coordinates (Topic 6 · slide 24), are unchanged too. Only the **direction flips by 180°** (bright → dark instead of dark → bright). In the pipeline: at (2, 2) G_x becomes −20, G is still 20. added $\tan^{-1}(G_y/G_x)$ alone cannot see the flip, since 0/20 and 0/−20 are both 0; you need the signs. **LO4 ›** · **LO5 ›** · **LO6 ›**

Final mixed check: 12 MCQs

M1. Removing white specks (noise) from a photo happens at which stage? CHOOSE 1

A. Capture
B. Pre-processing
C. Analysis
D. Decision

M2. How many grey levels can 8 bits per pixel represent? CHOOSE 1

A. 8
B. 128
C. 255
D. 256

M3. Sampling digitises... CHOOSE 1

A. the amplitude (intensity)
B. the x and y coordinates
C. the colour model
D. the histogram

M4. Which statements about quantization are true? CHOOSE ALL THAT APPLY

A. The range from black to white is divided into equal parts
B. Each value takes the level it is vertically closest to
C. It decides how many pixels the image has
D. It produces quantization error

M5. Image averaging reduces random noise only when… **CHOOSE 1**

A. the scene is stationary across the N images
B. the images are binary
C. N = 2
D. the images are different sizes

M6. Which are true of nearest-neighbour interpolation? **CHOOSE ALL THAT APPLY**

A. Computationally inexpensive
B. Preserves the original pixel intensity values
C. May introduce jagged edges and aliasing
D. Creates new in-between intensity values

M7. Salt and pepper noise is mainly caused by… **CHOOSE 1**

A. thermal effects in the sensor
B. transmission and storage errors
C. coherent imaging such as ultrasound
D. photon statistics

M8. Which are true of the median filter? **CHOOSE ALL THAT APPLY**

A. Particularly effective against salt & pepper noise
B. Preserves edges better than linear smoothing filters
C. It is non-linear
D. It can be written as a fixed 3 × 3 convolution kernel

M9. What is the only difference between the Prewitt and Sobel operators? **CHOOSE 1**

A. Sobel uses $[-1\ 0\ 1]$, Prewitt uses $[1\ -2\ 1]$
B. Sobel smooths with $[1\ 2\ 1]^T$, Prewitt with $[1\ 1\ 1]^T$
C. Sobel is a second-derivative filter
D. Prewitt needs a Gaussian first

M10. What is the correct order of the Canny steps? **CHOOSE 1**

A. Gradient → Gaussian → double threshold → NMS → hysteresis
B. Gaussian → gradient → non-max suppression → double threshold → hysteresis
C. Gaussian → double threshold → gradient → hysteresis → NMS
D. NMS → Gaussian → gradient → hysteresis → double threshold

M11. Why is $y = mx + c$ a problem for the line $x = 1$? **CHOOSE 1**

A. It has a slope of 0
B. Its slope is infinite
C. It passes through the origin
D. It needs three parameters

M12. Which are true of the Hough accumulator? **CHOOSE ALL THAT APPLY**

A. It is a two-dimensional array
B. $A(d, \theta)$ is the number of votes for that combination
C. Each edge pixel casts votes for lines through it
D. It stores the grey level of each pixel

Answer key

1. **M1: B**. Pre-processing cleans the signal. That step is an image-processing task used inside the CV pipeline. **Topic 1 · slides 6, 8**
2. **M2: D**. n bits give 2^n levels: $2^8 = 256$ (values 0…255). 255 is the top value, not the count. **Topic 1 · slides 15, 28**
3. **M3: B**. Sampling digitises the coordinates; quantization digitises the amplitude. **Topic 2 · slides 10–11**
4. **M4: A, B, D**. Quantization works on intensity: equal levels, nearest level, and an error between true and stored value. The number of pixels is set by sampling. **Topic 2 · slides 17–18**
5. **M5: A**. Averaging N images of the same still scene keeps the signal and averages out random noise (astronomy, medical imaging). Moving content would blur. **Topic 3 · slides 12–13**
6. **M6: A, B, C**. NN copies the closest source pixel: cheap and value-preserving, but blocky. Creating in-between values is what bilinear does. **Topic 3 · slides 29–30**
7. **M7: B**. Transmission and storage errors are the primary cause of salt & pepper (impulse) noise. **Topic 4 · slides 14–15**
8. **M8: A, B, C**. Because it is non-linear (sort and pick the middle), it cannot generally be written as a fixed kernel. **Topic 4 · slide 31**
9. **M9: B**. Both use the $[-1\ 0\ 1]$ derivative mask; Sobel's smoothing vector is weighted $[1\ 2\ 1]^T$. **Topic 5 · slide 17**
10. **M10: B**. Smooth first, then differentiate, thin, classify, and finally connect. **Topic 5 · slide 31**
11. **M11: B**. A vertical line has infinite slope and no y-intercept, so slope–intercept form cannot represent it. **Topic 6 · slides 9–10**
12. **M12: A, B, C**. It counts votes per (d, θ); it does not store intensities. **Topic 6 · slides 18–19**

Likely exam questions

1. Distinguish computer vision, image processing and computer graphics. **LO1 ›**
2. Explain sampling and quantization; define quantization error. **LO2 ›**
3. Histogram-equalise a given 8-level histogram (worked table). **LO2 ›**
4. Contrast stretching: apply the formula; why do outliers break it? **LO2 ›**
5. Define the arithmetic operations and give applications (motion detection, masking, averaging). **LO3 ›**
6. Why does translation need homogeneous coordinates? **LO3 ›**
7. Forward vs inverse mapping; nearest-neighbour vs bilinear (scale a 2 × 2 by 2). **LO3 ›**
8. Describe the four noise types, their models and the filter for each. **LO4 ›**
9. Mean vs Gaussian vs median: which for salt & pepper, and why? **LO4 ›**
10. Convolution vs cross-correlation; boundary handling options. **LO4 ›**
11. Compute G, θ and the edge orientation from I_x, I_y. **LO5 ›**
12. Prewitt vs Sobel; Laplacian zero crossings; LoG. **LO5 ›**
13. The five steps of Canny and why each is needed. **LO5 ›**
14. Why polar form? Give (d, θ) for a line such as $y = x + 1$ or $x = 3$. **LO6 ›**
15. Run a Hough accumulator on a few edge pixels and recover the line. **LO6 ›**

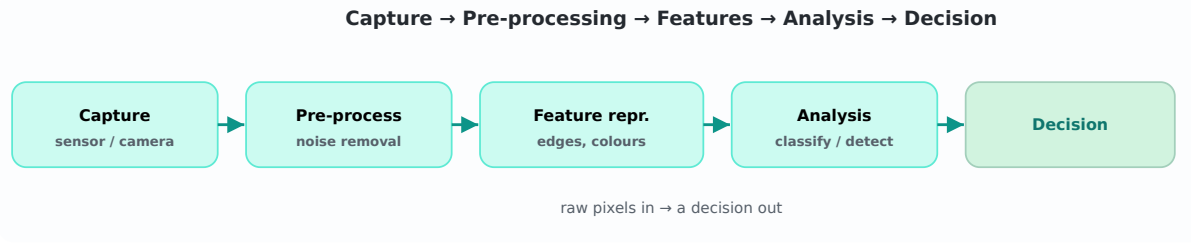
L01 Introduction to Computer Vision

What computer vision is, how it differs from image processing and computer graphics, the CV pipeline, and what a digital image really is — pixels, grey levels, the five resolutions and colour models.

Short Note — Quick Revision

- **Computer Vision (CV)** = enabling computers to **extract information from images & video** — recognising objects, detecting people, reading text, identifying faces.
- **Human vs computer vision:** human vision is a **biological perception system**; CV **computationally reproduces** useful aspects of that perception.
- **Three related fields: Image Processing** is *image* → *image* (enhance/filter/restore), **Computer Graphics** is *model* → *image* (render/animate), and **CV** is *image* → *meaning* (detect/recognise/segment/track).
- **CV pipeline: Capture → Pre-processing → Feature representation → Analysis → Decision** (raw pixels in, a decision out).
- **Digital image** = a **2-D function I(x, y)**; in a *digital* image x, y and the intensity I are all **finite & discrete** — a finite grid of **pixels**, each with a location and a grey/intensity value.
- **Five resolutions: pixel**, **spatial**, **spectral**, **temporal**, **radiometric**. **Colour models: RGB** (additive), **CMY** (subtractive), **HSI/HSV** (closer to human description).

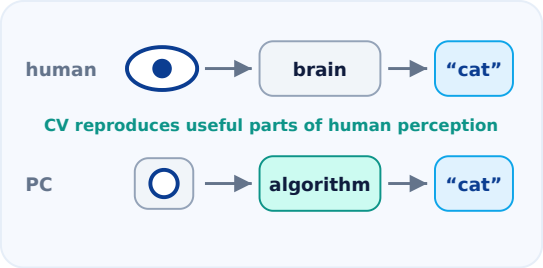
The computer-vision pipeline



A typical CV pipeline. We **capture** with a sensor, **pre-process** to clean the signal, build a **feature representation** (edges, colours), **analyse** it (classification/detection) and output a **decision** — the stage where raw pixels become meaning.

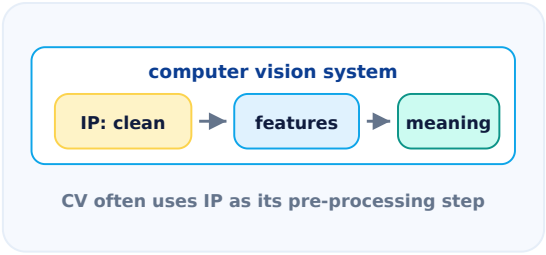
Discussion (following the lecture)

What is computer vision? (and how does it relate to human vision)

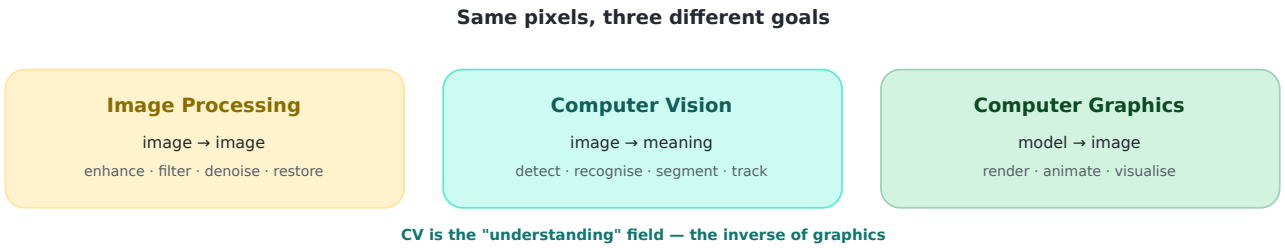


Computer Vision is the field concerned with **enabling computers to extract information from images and video**. Typical tasks include *recognising objects, detecting people, reading text and identifying faces*. The motivating analogy is **human vision** — a biological perception system — and CV **attempts to computationally reproduce useful aspects of that perception**. Everyday uses range from a smartphone camera and photo search to professional ones: medical diagnosis, robotics, autonomous vehicles and satellite imagery.

CV vs Image Processing vs Computer Graphics

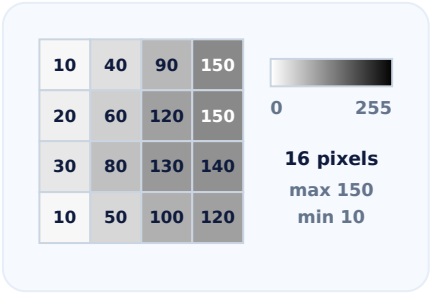


Three neighbouring fields are easy to confuse. **Image Processing (IP)** *manipulates and transforms images* — enhancing quality through filtering, noise removal, contrast enhancement and restoration (**image → image**). **Computer Graphics** goes the other way: it *generates* images and visual content from 2D/3D models and scenes — rendering, animation, simulation, visualisation (**model → image**). **Computer Vision** is the *understanding* field: it *interprets* images and **extracts meaningful information** — object detection, recognition, segmentation and tracking (**image → meaning**). CV is effectively the inverse of graphics, and it often *uses* IP as a pre-processing step.



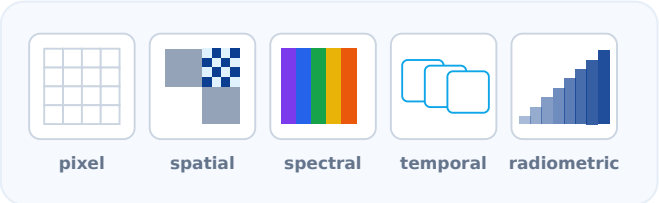
IP cleans a picture, graphics draws one from a model, and CV reads **meaning** out of one. Only CV turns pixels into a description of *what is in the scene*.

What is a digital image — pixels & grey levels



An image is a **two-dimensional function I(x, y)**, where x and y are spatial coordinates and the value is the intensity at that point. In a **digital** image, x, y and the intensity I are all **finite, discrete quantities** — the image is composed of a finite number of components, each with a particular *location* and *value*. Each such element is a **pixel** (picture element). A **grey-level** grid stores the brightness of each pixel; from a small matrix you can read off the image width, height, number of pixels, and the brightest and darkest pixels (e.g. a 4×4 grid of values 10–150 has 16 pixels, brightest 150, darkest 10).

The five resolutions & colour models



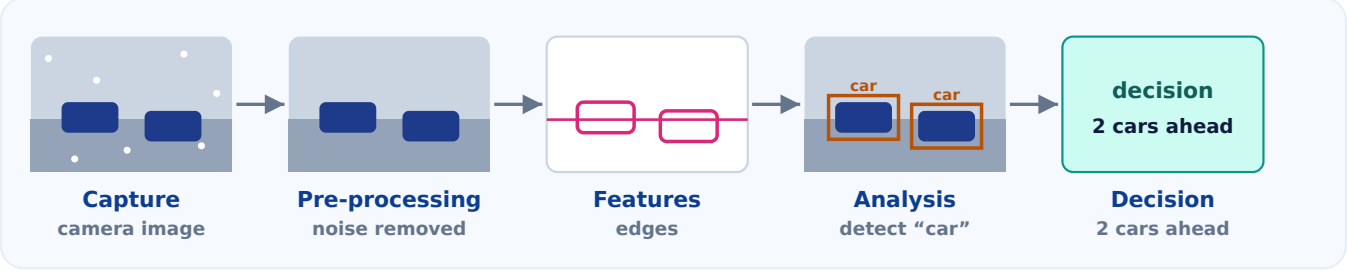
Resolution comes in five flavours: **pixel resolution** (the total pixel count / dimensions), **spatial resolution** (how much detail per unit area — the finest distinguishable detail), **spectral resolution** (the bands / colours captured), **temporal resolution** (how often frames are captured over time), and **radiometric resolution** (bits per intensity → how many grey levels can be represented). Colour is encoded with **colour models**: **RGB** is an *additive* model (mixing light), **CMY** is a *subtractive* model (mixing pigments/inks), and **HSI/HSV** (hue, saturation, intensity/value) describes colour the way humans tend to — making it convenient for analysis.

RESOLUTION	WHAT IT MEASURES	MORE OF IT MEANS…
Pixel	Total number of pixels / image dimensions	A bigger grid of samples
Spatial	Detail per unit area (finest distinguishable detail)	Sharper, finer detail
Spectral	Number of wavelength bands / colours captured	Richer colour / multispectral information
Temporal	How frequently frames are captured over time	Smoother motion / faster sampling
Radiometric	Bits used per intensity value	More grey levels / finer brightness steps

Step by step — deeper notes pilot

Everything here follows the Topic 1 slides; each part shows its slide numbers. Parts marked added are study aids we wrote, not lecture content.

1 · The pipeline on one picture Topic 1 · slides 3–4, 8

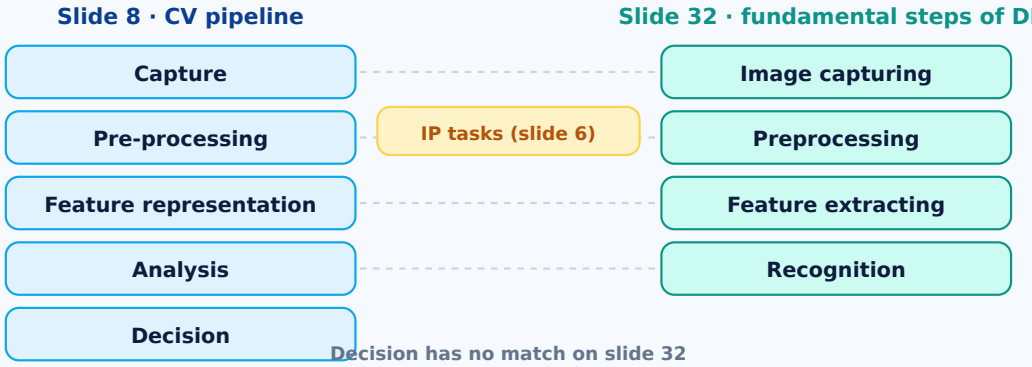


Slide 8’s five stages on the kind of street photo shown on slide 4. **Capture** gets the pixels from a sensor or camera. **Pre-processing** removes noise (the white specks). **Feature representation** keeps what matters, such as edges and colours. **Analysis** classifies and detects (“car”). The **decision** is the output.

The “2 cars ahead” decision is an example we chose; the slide names the stage only.

2 · Compare: two lists of steps, and where image processing sits

Topic 1 · slides 6, 8, 32



The lecture gives the steps twice. Slide 8 has five stages for a **CV pipeline**; slide 32 has four **fundamental steps of digital image processing**. They line up one to one, and only slide 8 adds a final **decision**.

Why it matters: the IP tasks on slide 6 (filtering, noise removal, contrast enhancement, restoration) are exactly what the **pre-processing** stage does. So CV uses image processing inside its pipeline, then goes further to extract meaning.

3 · Worked exercise: read a grey-level grid

Topic 1 · slide 17



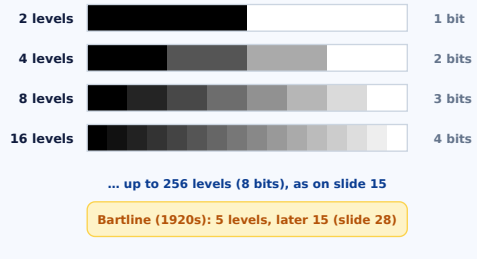
Slide 17 asks five questions about this grid. Each number is a pixel's **grey level** (intensity), so a bigger number means brighter.

1. **Width** = 4 (pixels per row)
2. **Height** = 4 (rows)
3. **Pixels** = 4 × 4 = 16
4. **Brightest** = 150 (row 3, column 3)
5. **Darkest** = 10 (row 1, column 1)

This is what slide 13 means by "a finite number of components, each with a particular location and value".

4 · Grey levels: how many shades?

Topic 1 · slides 15-16, 28



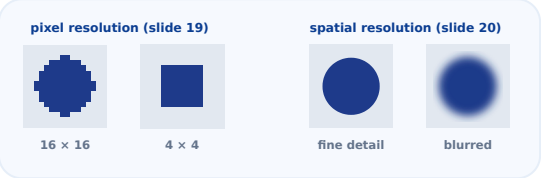
Slide 15 shows the same black-to-white ramp with **2, 4, 8 … 256 levels**. Slide 16 shows a face with fewer and fewer levels: with very few levels, smooth skin turns into flat patches, and at 2 levels the face is just black and white.

History link (slide 28): the first newspaper picture system, the **Bartline cable** (1920s, London → New York), could send only **5 intensity levels**, later **15**.

added The bits column: n bits give 2^n levels, so 8 bits = 256.

5 · Resolution: pixel vs spatial

Topic 1 · slides 18–20



Slide 18 lists five resolutions; slides 19–20 show two of them. **Pixel resolution** is how many pixels the image has: the same car with fewer pixels turns into blocks. **Spatial resolution** is how much fine detail the image shows: the same face can have the same pixel count but look blurred.

Slide 18 also names **spectral**, **temporal** and **radiometric** resolution without examples; in short: bands of colour captured, frames over time, and grey levels per pixel (as in section 4).

6 · Colour models: adding light vs adding ink

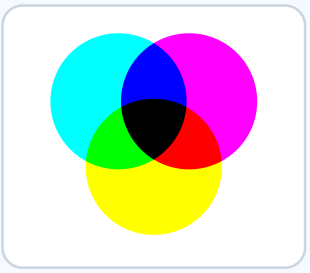
Topic 1 · slides 21–26

RGB · additive (slide 23)



+G = yellow · all three = white

CMY · subtractive (slide 24)



C+M = blue · all three = black

Each ink absorbs one light (slide 25)

ink	absorbs	reflects
cyan	red	green + blue
magenta	green	red + blue
yellow	blue	red + green

no ink = white · all three inks = black

Slide 21 splits colour models into **additive** and **subtractive**. **RGB** (slide 23) adds coloured *light*: red + green = yellow, and all three give white. **CMY** (slides 24–25) adds *ink*, and each ink **absorbs** one light: cyan absorbs red, magenta absorbs green, yellow absorbs blue. More ink means less light, and all three give black.

HSV (slide 26) is drawn as a cone: **hue** is the angle around it, **saturation** the distance from the centre, **value** the height.

7 · Where image processing came from, and where it is used

Topic 1 · slides 9, 27-31



slides 28–30

Origins: newspaper pictures over the Bartline cable (1920s) → the first computers able to do real image processing (1960s; in 1964 Ranger 7's images were corrected for distortion) → medical use in the 1970s (CAT / CT).

Two aims of IP (slide 27): improve pictures for **human interpretation**, and process image data for storage, transmission and **machine perception**.

Fields (slide 31): character recognition · machine vision for assembly and inspection · military reconnaissance · fingerprints · satellite images for weather · X-ray and gamma-ray imaging. **CV applications** (slide 9): smartphone camera, photo search, medical diagnosis, robotics, autonomous vehicles, satellite imagery.

8 · Exam traps

Topic 1 · slides 6–8, 17, 19–20, 21–25 added

- **IP vs CV:** IP outputs an *image*; CV outputs *information* (slide 6). Noise removal alone is IP, not CV.
- **Graphics is the reverse:** it *creates* images from models (slide 7); it doesn't read them.
- **Brightest = biggest number** in a grey-level grid (slide 17), not the pixel in the middle.
- **Pixel ≠ spatial resolution:** more pixels doesn't always mean more detail (slides 19–20).
- **RGB adds light, CMY absorbs it:** all three RGB = white, all three CMY inks = black (slides 23–25).

9 · Quick check

Topic 1 · slides 6, 8, 17, 25 added

▼ **Q1.** Which stage of the pipeline would remove the white specks in a photo, and is that IP or CV?

Pre-processing (slide 8). Removing noise is an image-processing task (slide 6), used inside the CV pipeline.

▼ **Q2.** A 3 × 5 grey-level image: how many pixels? If its values are 12–240, which is the brightest?

3 × 5 = 15 pixels. The brightest pixel is the one with value 240 (slide 17 logic).

▼ **Q3.** Magenta ink on white paper: which light does it absorb, and which does it reflect?

It absorbs green and reflects red + blue, which we see as magenta (slide 25).

Key terms

computer vision · image processing (image → image) · computer graphics (model → image) · image → meaning · CV pipeline · I(x, y) · finite & discrete · pixel · grey level · pixel / spatial / spectral / temporal / radiometric resolution · RGB (additive) · CMY (subtractive) · HSI / HSV

Exam relevance

Try each question yourself first, then reveal the answer.

Q1. "Define computer vision and give examples of tasks it performs. How does it relate to human vision?"

Answer: Computer vision is the field concerned with **enabling computers to extract information from images and video**. Example tasks: recognising objects, detecting people, reading text and identifying faces. It relates to human vision as follows — human vision is a **biological perception system**, and CV **attempts to computationally reproduce useful aspects of that perception**.

Q2. "Distinguish computer vision, image processing and computer graphics."

Answer: **Image Processing** manipulates/transforms images to improve or restore them (filtering, noise removal, contrast enhancement, restoration) — **image → image**. **Computer Graphics** generates images and visual content from 2D/3D models and scenes (rendering, animation, visualisation) — **model → image**. **Computer Vision** understands and interprets images, extracting meaningful information (detection, recognition, segmentation, tracking) — **image → meaning**. CV is the "understanding" field and is essentially the inverse of graphics.

Q3. "What is a digital image? Explain pixels and grey levels, and the stages of the CV pipeline."

Answer: An image is a **2-D function I(x, y)**. In a **digital** image x, y and the intensity I are all **finite, discrete** quantities — a finite number of components, each with a location and a value. Each element is a **pixel** (picture element); a **grey level** is its intensity/brightness value. The CV pipeline runs **Capture → Pre-processing → Feature representation → Analysis → Decision** — raw pixels in, a decision out.

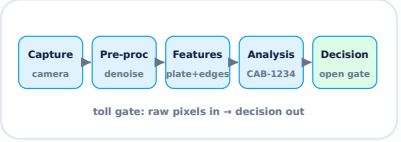
Q4. "Name the five types of image resolution and the main colour models."

Answer: The five resolutions are **pixel** (total pixels/dimensions), **spatial** (detail per area), **spectral** (bands/colours captured), **temporal** (frames over time) and **radiometric** (bits per intensity → number of grey levels). Colour models: **RGB** — additive (mixing light); **CMY** — subtractive (mixing inks/pigments); and **HSI/HSV** (hue, saturation, intensity/value) which is closer to how humans describe colour.

Practice: 10 questions

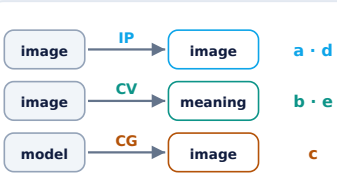
Exam-style questions on Topic 1: apply, classify, calculate, explain. Answer on paper first, then reveal. Each answer cites its slides; **added** marks anything that goes beyond the slides. "Show all answers" above opens these too.

P1. **APPLY** An automatic number-plate reader at a toll gate: describe what happens at each of the five CV pipeline stages.



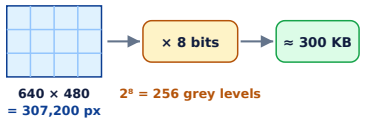
Answer · Topic 1 slide 8**:** **Capture**: the camera takes a picture of the car. **Pre-processing**: remove noise and fix the contrast (night, rain). **Feature representation**: find the plate region and the edges and shapes of each character. **Analysis**: recognise the characters ("CAB-1234"). **Decision**: look up the plate and open the gate or charge the toll. *Exam tip:* name all five stages in order, then give one line each for *this* system.

P2. **CLASSIFY** Is each task Image Processing (IP), Computer Vision (CV) or Computer Graphics (CG)? (a) sharpening a blurred photo (b) a phone unlocking with your face (c) rendering a 3-D game scene (d) removing noise from an X-ray (e) counting cars in a traffic video.



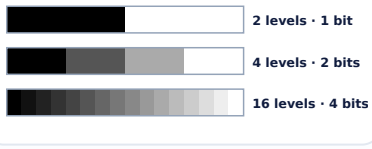
Answer · Topic 1 slides 6–7**:** (a) **IP**: image → better image. (b) **CV**: image → identity. (c) **CG**: model → image. (d) **IP**: image → cleaner image. (e) **CV**: video → a number. *The rule:* look at the **output**. An image means IP, information means CV, and making an image from a model means CG.

P3. **CALCULATE** An image is 640 × 480 pixels with 8 bits per pixel. (a) How many pixels? (b) How many grey levels? (c) How much storage, uncompressed?



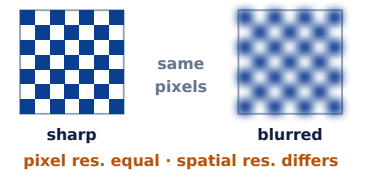
Answer · Topic 1 slides 13–15, 18**:** (a) 640 × 480 = **307,200 pixels**. (b) 2^8 = **256 grey levels** (0 = black … 255 = white). (c) 307,200 × 8 bits = 307,200 bytes ≈ **300 KB**. *Show the working:* pixels × bits ÷ 8 = bytes. **added** The storage part goes one step beyond the slides.

P4. **CALCULATE** How many bits per pixel are needed for (a) 16 grey levels (b) 2 grey levels? What do you see when a face image drops from 256 levels to 4?



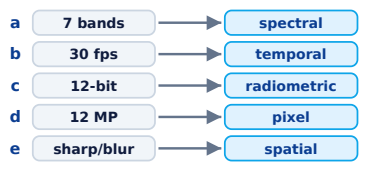
Answer · Topic 1 slides 15–16**:** (a) 2^4 = 16, so **4 bits**. (b) 2^1 = 2, so **1 bit** (a binary, black-and-white image). With 4 levels, smooth shading turns into **flat patches with visible steps** (false contours), as on slide 16. Fewer levels means less smooth brightness, not fewer pixels.

P5. **EXPLAIN** Two images are both 1000 × 1000 pixels, but one shows much more detail. How is this possible?



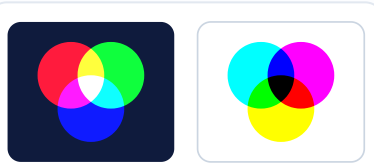
Answer · Topic 1 slides 18–20**:** They have the same **pixel resolution** (the pixel count) but different **spatial resolution** (how much fine detail per area is actually captured). Blur, a poor lens, focus or up-scaling a small image all give many pixels with little detail. *Trap:* more pixels ≠ more detail.

P6. **MATCH** Match each to its type of resolution: (a) a satellite sensor capturing 7 bands (b) CCTV recording 30 frames per second (c) a 12-bit medical X-ray (d) a 12-megapixel phone camera (e) a sharp vs a blurred face of the same size.



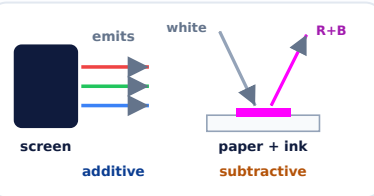
Answer · Topic 1 slide 18**:** (a) **spectral**: how many bands or colours. (b) **temporal**: frames over time. (c) **radiometric**: bits per pixel, so 2^{12} = 4,096 grey levels. (d) **pixel**: the total pixel count. (e) **spatial**: detail per area.

P7. COLOUR What colour do you get? (a) RGB light: red + green (b) red + blue (c) green + blue (d) CMY inks: cyan + yellow (e) all three CMY inks.



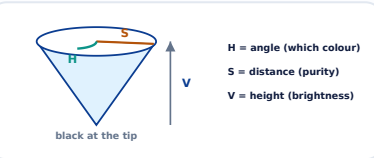
Answer · Topic 1 slides 21–25**:** (a) **yellow** (b) **magenta** (c) **cyan**: RGB is additive, so light is added. (d) **green**: cyan absorbs red and yellow absorbs blue, so only green is reflected. (e) **black**: all light is absorbed. CMY is subtractive: ink takes light away.

P8. EXPLAIN Why do screens use RGB but printers use CMY?



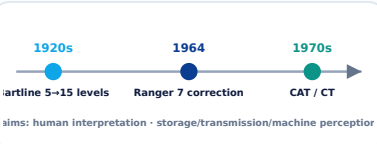
Answer · Topic 1 slides 21–25**:** A **screen emits light**, so colours are made by **adding** red, green and blue light (additive; all three = white). **Paper reflects light**, so colours are made by inks that **absorb** (subtract) parts of white light (subtractive; all three = black). added Real printers add a separate black ink (K → CMYK), because C+M+Y gives a muddy black. The slides show CMY only.

P9. DESCRIBE Describe the three components of the HSV model using its cone, and give one reason to prefer HSV over RGB.



Answer · Topic 1 slide 26**:** **Hue** is the angle around the cone (which colour: red, green …). **Saturation** is the distance from the centre (how pure or vivid; centre = grey). **Value** is the height (how bright; bottom = black). Why: it is **closer to how people describe colour**. added Example: to find "all red objects" you can select by hue alone, so shadows and highlights don't break it. In RGB, brightness is mixed into all three numbers.

P10. OUTLINE Outline three milestones in the history of image processing, and state the two main aims of image processing.



Answer · Topic 1 slides 27–28, 30**:** **1920s**: the Bartline cable sent newspaper pictures London → New York, with only **5 grey levels, later 15**. **1964**: **Ranger 7** Moon images corrected for distortion by computer (the first real computer image processing). **1970s**: medical imaging, **CAT/CT**. **Aims** (slide 27): ① improve pictures for **human interpretation**; ② process image data for **storage, transmission and machine perception**.

Quick MCQs: check your understanding

M1. What does computer vision produce from an image? CHOOSE 1

A. A better-looking image
B. Information about the scene
C. A 3-D model to render
D. A compressed file

M2. Which of these are image-processing tasks? **CHOOSE ALL THAT APPLY**

A. Noise removal
B. Contrast enhancement
C. Recognising a face
D. Image restoration

M3. Computer graphics works in which direction? **CHOOSE 1**

A. image → image
B. image → meaning
C. model → image
D. meaning → model

M4. Which is the correct order of the CV pipeline? **CHOOSE 1**

A. Capture → Analysis → Pre-processing → Features → Decision
B. Capture → Pre-processing → Feature representation → Analysis → Decision
C. Pre-processing → Capture → Features → Decision → Analysis
D. Capture → Features → Pre-processing → Decision → Analysis

M5. Removing white specks (noise) from a photo happens at which stage? **CHOOSE 1**

A. Capture
B. Pre-processing
C. Analysis
D. Decision

M6. Which statements about a digital image are true? **CHOOSE ALL THAT APPLY**

A. x and y are finite, discrete values
B. The intensity is a continuous value
C. Each pixel has a location and a value
D. It is a 2-D function I(x, y)

M7. A grey-level grid is 4 pixels wide and 4 pixels high. How many pixels? **CHOOSE 1**

A. 8
B. 12
C. 16
D. 32

M8. In a grey-level grid, the brightest pixel is the one with… **CHOOSE 1**

A. the smallest value
B. the largest value
C. the value in the middle of the grid
D. the value in the top-left corner

M9. How many grey levels can 8 bits per pixel represent? **CHOOSE 1**

A. 8
B. 128
C. 255
D. 256

M10. Which are types of resolution listed in the lecture? **CHOOSE ALL THAT APPLY**

A. Pixel
B. Spatial
C. Chromatic
D. Spectral
E. Temporal
F. Radiometric

M11. Two images both have 1000 × 1000 pixels. Must they show the same detail? **CHOOSE 1**

A. Yes: same pixels, same detail
B. No: spatial resolution can differ

M12. Red light + green light (RGB) gives... **CHOOSE 1**

A. Yellow
B. Magenta
C. Cyan
D. White

M13. Which statements about CMY are true? **CHOOSE ALL THAT APPLY**

A. Each ink absorbs one colour of light
B. Cyan ink absorbs red
C. All three inks together give black
D. Screens mix colours using CMY

M14. In the HSV cone, hue is... **CHOOSE 1**

A. the height of the cone
B. the angle around the cone
C. the distance from the centre
D. the black tip

Answer key

1. **M1: B**. CV goes image → meaning (information). Image → image is image processing; model → image is graphics. **Topic 1 · slides 6–7**
2. **M2: A, B, D**. Filtering, noise removal, contrast enhancement and restoration all output an image, so they are IP. Recognising a face outputs information, so it is CV. **Topic 1 · slide 6**
3. **M3: C**. Graphics creates images from 2-D/3-D models (rendering, animation). It is the reverse of CV. **Topic 1 · slide 7**
4. **M4: B**. Raw pixels are captured, cleaned (pre-processing), reduced to features, analysed, and turned into a decision. **Topic 1 · slide 8**
5. **M5: B**. Pre-processing cleans the signal. That step is an image-processing task used inside the CV pipeline. **Topic 1 · slides 6, 8**
6. **M6: A, C, D**. In a digital image x, y and the intensity are all finite and discrete: a finite grid of pixels, each with a location and a value. **Topic 1 · slide 13**
7. **M7: C**. Width × height = 4 × 4 = 16 pixels. **Topic 1 · slide 17**
8. **M8: B**. A grey level is brightness: a bigger number means brighter (0 = black). **Topic 1 · slide 17**
9. **M9: D**. n bits give 2^n levels: $2^8 = 256$ (values 0…255). 255 is the top value, not the count. **Topic 1 · slides 15, 28**
10. **M10: A, B, D, E, F**. The five are pixel, spatial, spectral, temporal and radiometric. "Chromatic resolution" is not one of them. **Topic 1 · slide 18**
11. **M11: B**. Pixel resolution is the count; spatial resolution is the detail actually captured. A blurred image can have many pixels but little detail. **Topic 1 · slides 19–20**
12. **M12: A**. RGB is additive: red + green = yellow. All three together give white. **Topic 1 · slide 23**
13. **M13: A, B, C**. CMY is subtractive: inks absorb light (cyan → red, magenta → green, yellow → blue), and all three give black. Screens emit light, so they use RGB. **Topic 1 · slides 24–25**
14. **M14: B**. Hue = angle (which colour), saturation = distance from the centre, value = height (brightness). **Topic 1 · slide 26**

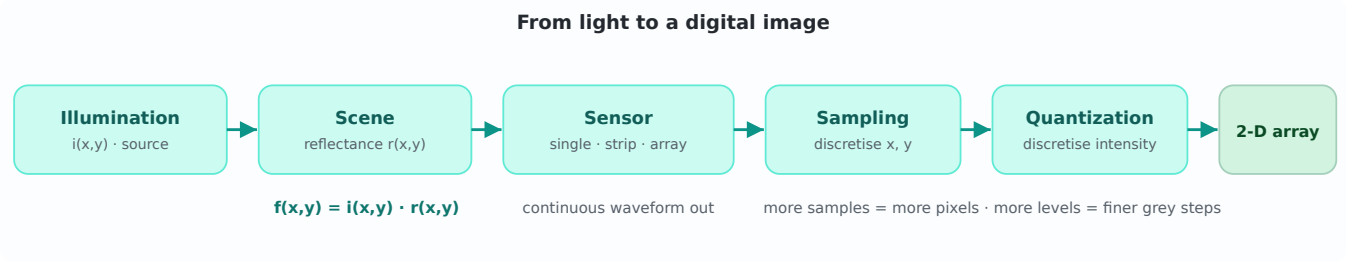
L02 Visual perception & image sensing

How light becomes a digital image: the EM spectrum, sensor arrangements and the image formation model, then sampling and quantization. After that, the intensity histogram and two ways to improve contrast with it: contrast stretching and histogram equalisation. slides

Short Note — Quick Revision

- **Light** = the part of the EM spectrum the eye senses. The colour we see = the light **reflected**. **Monochromatic** = grey level only. **Chromatic** adds **radiance (W), luminance (lm), brightness**.
- **Acquisition: single sensor** (moves in x and y) · **sensor strip** (a line at a time) · **sensor array** (2-D grid, cameras).
- **Image formation model:** `f(x,y) = i(x,y) · r(x,y)`, illumination × reflectance.
- **Sampling** digitises the **coordinates**. **Quantization** digitises the **intensity**. Rounding to the nearest level causes **quantization error**. Result = a 2-D array; 8-bit = 256 levels (0 black, 255 white).
- **Histogram** $h(r_k) = n_k$, the pixel count per intensity level. Its shape shows dark, bright and low- or high-contrast images.
- **Contrast stretching:** $I_{out} = (I_{in}-c)(b-a)/(d-c)+a$. Outliers spoil it, so use the 5th/95th percentiles or a cut-off. **Equalisation:** $s_k = \Sigma\, n_i/n$, new = round($s_k \cdot m$).

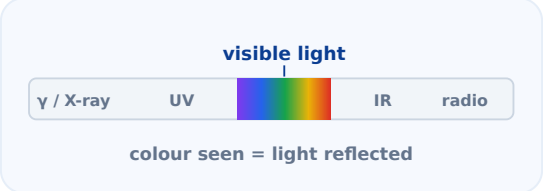
From light to pixels



Light from a source (**illumination**) is **reflected** by the scene, picked up by a sensor, and made digital in two steps: **sampling** (where) and **quantization** (how bright).

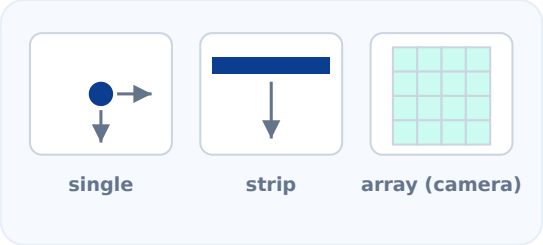
Discussion (following the lecture)

Light and the electromagnetic spectrum



The EM spectrum can be described by **wavelength, frequency or energy**. **Light** is the part of it the human eye can sense, and the colour we see in an object is set by the light it **reflects**. **Monochromatic** light has no colour and is described only by its intensity, the **grey level**. **Chromatic** (colour) light needs three more quantities besides frequency: **radiance** (energy from the source, in watts), **luminance** (energy the observer perceives, in lumens) and **brightness** (a subjective sense of intensity).

Image sensing and acquisition



Acquisition turns illumination energy into a digital image. There are three sensor arrangements. A **single sensor** must be moved in both x and y to build the 2-D image (slow, but precise). A **sensor strip** captures one line at a time while the strip or object moves (flatbed scanners). A **sensor array** is a 2-D grid that captures the whole image at once (CCD/CMOS cameras). The **simple image formation model** is $f(x,y) = i(x,y) \cdot r(x,y)$: the recorded value is the *illumination* falling on the scene times the *reflectance* of the surface. Intensities are proportional to the energy radiated from the physical source.

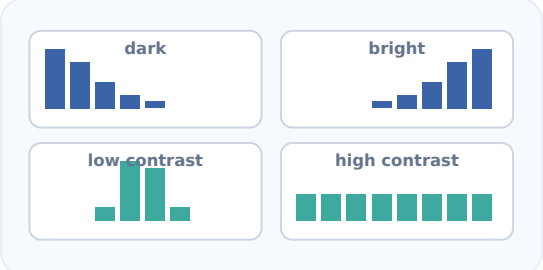
Sampling and quantization



A sensor produces a **continuous** waveform, continuous in both position and amplitude. Two steps make it digital. **Sampling** digitises the *coordinates* (x, y): a 10 × 10 grid gives 100 samples, and more pixels means more samples and finer spatial detail. **Quantization** digitises the *amplitude*: the range from black to white is split into equal steps, and each sample is given the nearest discrete level (by its vertical closeness to the levels). The gap between the true value and the level it is given is the **quantization error**. The result is a **2-D array** of M × N values with L grey levels, usually $L = 2^k$ (8-bit gives 256 levels, 0 = black and 255 = white).

STEP	DISCRETISES	TOO LITTLE OF IT GIVES…
Sampling	Spatial coordinates x, y	blocky images, lost fine detail (aliasing)
Quantization	Intensity / amplitude	false contours (banding), larger quantization error

The intensity histogram

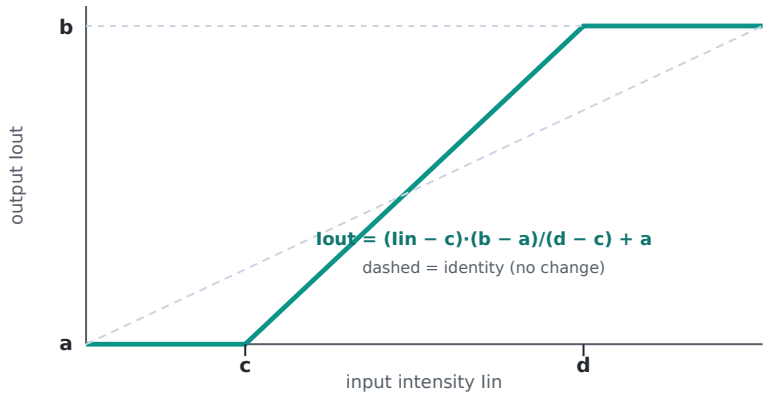


For an image with levels in $[0, L-1]$, the histogram is the discrete function $\mathbf{h(r_k) = n_k}$, where r_k is the k-th intensity and n_k is the number of pixels with that intensity. The x-axis is the intensity level and the y-axis is the pixel count. Its shape tells you about the image: bunched to the left means a **dark** image, bunched to the right a **bright** one, and a narrow spike means **low contrast**. Spread across the range means good contrast. Dividing by n gives the probability $p(r_k) = n_k/n$.

Enhancement 1: contrast stretching



Stretch the input range `[c, d]` to the wanted output range `[a, b]` with $\mathbf{I_{out} = (I_{in} - c)(b - a)/(d - c) + a}$. Example: c = 50, d = 150, a = 0, b = 255, then I_{in} = 100 gives (50 × 255)/100 = **127.5**. **The problem is outliers**: a single very dark or very bright pixel fixes c or d and the stretch does almost nothing. **Fixes**: take c and d at the **5th and 95th percentiles** of the histogram, or set a **cut-off** relative to the histogram peak and ignore the data below it.



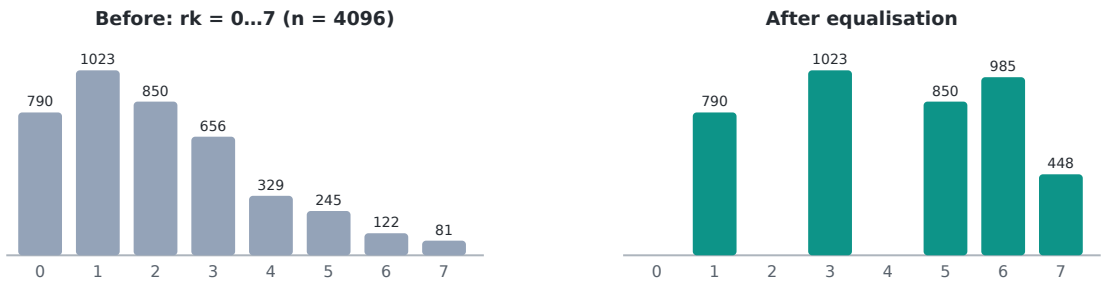
Everything below c goes to a, everything above d goes to b, and the range in between is stretched linearly.

Enhancement 2: histogram equalisation (worked example from the slides)



Equalisation **spreads out the most frequent intensities** over the whole range. With n pixels and levels 0…m: `p(k) = n_k/n`, the transform is the cumulative sum $\mathbf{s_k = \Sigma_{i=0..k}\, n_i/n}$, and the **new level = round(s_k × m)**.

R_K	N_K	CUMULATIVE	S_K	S_K × 7	NEW LEVEL
0	790	790	0.193	1.35	**1**
1	1023	1813	0.443	3.10	**3**
2	850	2663	0.650	4.55	**5**
3	656	3319	0.810	5.67	**6**
4	329	3648	0.891	6.23	**6**
5	245	3893	0.950	6.65	**7**
6	122	4015	0.980	6.86	**7**
7	81	4096	1.000	7.00	**7**



Most pixels sat in the dark levels 0–3; equalisation pushes them up so the image uses the whole range.

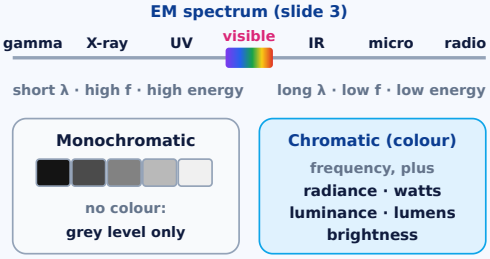
Same image before and after. n = 4096 and m = 7. Levels 3 and 4 both map to 6, and levels 5–7 all map to 7. Equalisation can merge levels but never splits them.

Step by step — deeper notes

Everything here follows the Topic 2 slides; each part shows its slide numbers. Parts marked **added** are study aids we wrote, not lecture content.

1 · Light: the part of the spectrum we can see

Topic 2 · slides 3–5



The electromagnetic spectrum can be described by **wavelength**, **frequency** or **energy** (slide 4). Going left on slide 3 (towards gamma rays) means shorter wavelength, higher frequency and higher energy; going right (towards radio) the opposite. **Light** is the small band the human eye can sense.

The colour we see in an object is set by the light **reflected** from it (slide 4). Slide 5 then splits light in two: **monochromatic** light has no colour, and its intensity is given as a **grey level**. **Chromatic** (colour) light needs frequency plus three quantities: **radiance** (watts), **luminance** (lumens) and **brightness**.

The slides give only the names and units of the three quantities, so learn them as a list of three.

2 · Sensors, and the simple image formation model

Topic 2 · slides 6–9

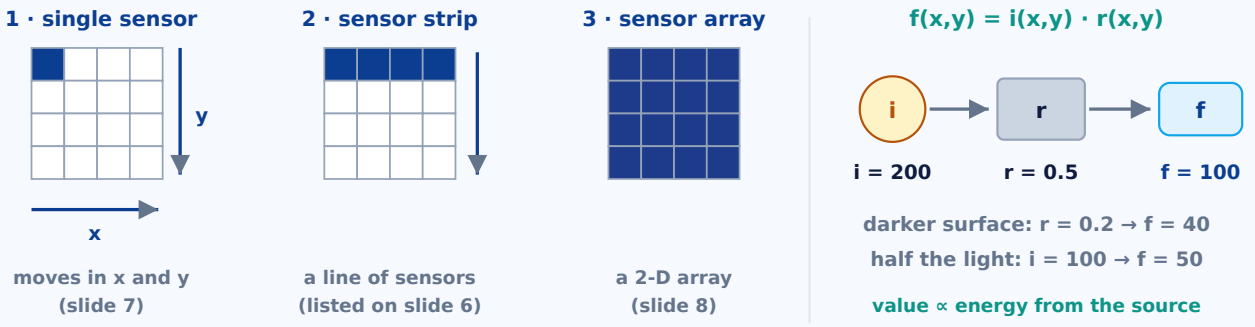


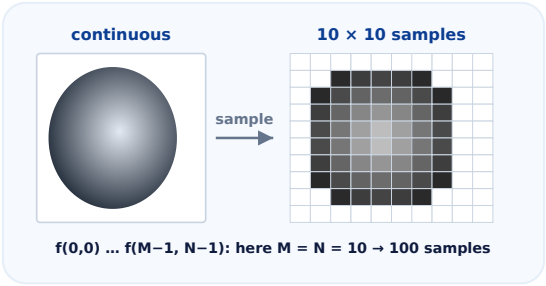
Image acquisition transforms illumination energy into a digital image (slide 6). Slide 6 lists three arrangements: (1) a **single sensor**, which has to **move in x and y** to build up the 2-D image (slide 7: energy passes a filter onto the sensing material, which gives out a voltage waveform); (2) **sensor strips**; (3) **sensor arrays**, sensors arranged as a **2-D array** (slide 8).

Formation model (slide 9): intensity values are proportional to the energy radiated from the physical source, written **f(x,y) = i(x,y) r(x,y)**. Worked: i = 200 and r = 0.5 give f = 200 × 0.5 = **100**. A darker surface (r = 0.2) gives 40; halving the light (i = 100) gives 50.

Calling i “illumination” and r “reflectance” is the usual reading; slide 9 gives only the formula. The numbers are our example.

3 · Sampling: digitise the coordinates, get a 2-D array

Topic 2 · slides 10–13, 19–20



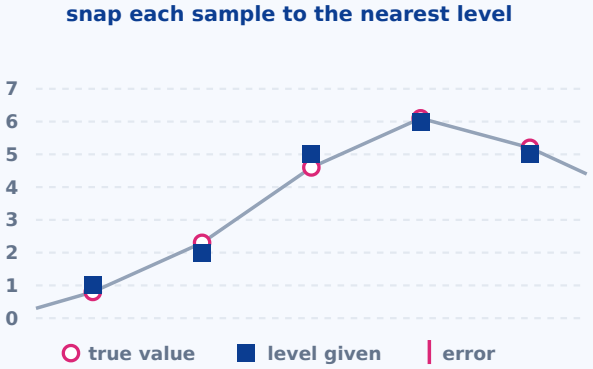
A sensor outputs a **continuous waveform**, so making a digital image takes two processes: **sampling** and **quantization** (slide 10). The image is continuous in **x, y** and in **amplitude**; sampling digitises the **x and y coordinates** (slide 11).

Step by step (slide 12): 1. lay a grid over the image, 10 on x and 10 on y. 2. take one value per cell: 10 × 10 = **100 samples** = 100 pixels. 3. more pixels means more samples, so finer detail. Slide 13 shows the same idea: a smooth shape becomes blocks.

The result is the image as a **2-D array** (slide 19): f(x,y) has rows f(0,0) … f(0,N−1) down to f(M−1,0) … f(M−1,N−1), so M × N samples. For a 10 × 10 grid, x and y each run 0…9. Slide 20 shows a digit drawn on a grid next to its numbers (0.0 = black, 1.0 = white).

4 · Quantization, and the error it leaves

Topic 2 · slides 14–18



sample	true	level	error
1	0.8	1	0.2
2	2.3	2	0.3
3	4.6	5	0.4
4	6.1	6	0.1
5	5.2	5	0.2

error never above half a step (0.5)

Quantization digitises the **amplitude**. Slides 14–16 take the intensity along a line **A–B** through the image: the profile is continuous, sampling picks points along it, and each point is then put on a grey scale.

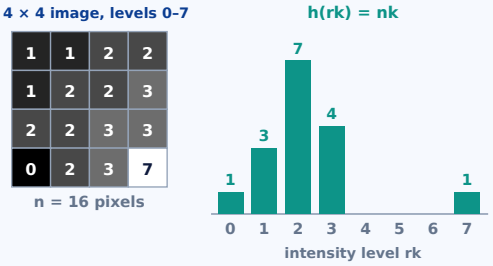
Step by step (slide 17): 1. divide the range from black to white into **equal parts**. 2. each continuous value is given **one of the discrete values**. 3. which one depends on the **vertical proximity** of the sample to the levels, so take the nearest. Worked with levels 0–7: 0.8 → 1, 2.3 → 2, 4.6 → 5, 6.1 → 6, 5.2 → 5.

The gap between the true value and the level it gets is the **quantization error** (slide 18 shows it as a staircase hugging a smooth curve, with binary codes 0000, 0001 … on the levels). Here the errors are 0.2, 0.3, 0.4, 0.1, 0.2.

added With nearest-level rounding the error is never more than half a step; more levels means smaller steps and smaller error.

5 · The histogram: count the pixels at each level

Topic 2 · slides 21–27



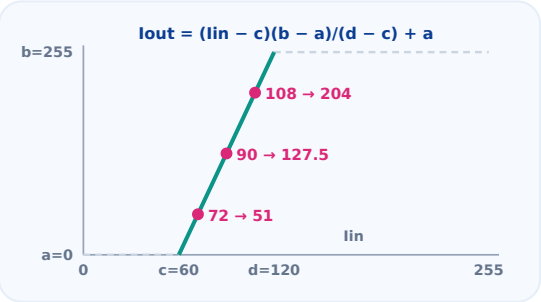
For levels in [0, L−1] the histogram is $\mathbf{h(r_k) = n_k}$: r_k is the k-th intensity and n_k is **how many pixels** have it (slide 21). The x-axis is the intensity level and the y-axis is the number of pixels (slides 22–23). An 8-bit grey image has 256 levels, 0 = black and 255 = white (slide 23).

Step by step on a 4 × 4 image with levels 0–7: 1. go through all 16 pixels. 2. tally each value: level 0 → 1, level 1 → 3, level 2 → 7, level 3 → 4, level 7 → 1, the rest 0. 3. check: 1 + 3 + 7 + 4 + 1 = **16** = number of pixels. 4. draw a bar per level.

Reading the shape (slides 24–27): bunched at the low end = **dark** image; at the high end = **bright**; squeezed into a narrow band = **low contrast**; spread across 0–255 = **high contrast**. Our image sits mostly on levels 1–3, so it is dark and low in contrast, with one lone pixel at 7.

6 · Contrast stretching, one pixel at a time

Topic 2 · slides 28-30



Contrast stretching stretches the range of intensities from its original range **[c, d]** to span the desired range **[a, b]** (slides 28–29): $\mathbf{I_{out} = (I_{in} - c)(b - a)/(d - c) + a}$. On slide 29, c and d mark where the histogram starts and ends, and a = 0, b = 255 are the ends of the scale.

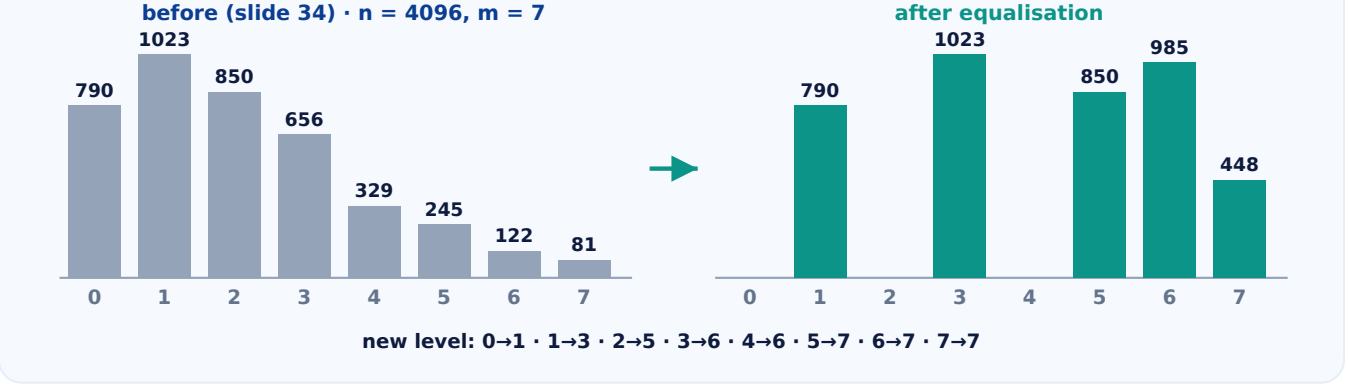
Worked (c = 60, d = 120, a = 0, b = 255): 1. scale = (b − a)/(d − c) = 255/60 = **4.25**. 2. for each pixel subtract c, multiply by 4.25, add a. I_{in} = 72: (72 − 60) × 4.25 = **51**. I_{in} = 90: 30 × 4.25 = **127.5**. I_{in} = 108: 48 × 4.25 = **204**. The ends go 60 → 0 and 120 → 255.

Problem: outliers (slide 30). Add one pixel at 0 and one at 255: if c and d are the min and max, then c = 0, d = 255, the scale is 1 and nothing changes. **Fixes:** take c and d at the **5th and 95th percentiles** of the histogram, or take the histogram **peak**, define a **cut-off** and ignore the data below it.

added Values below c or above d are usually clipped to a and b (the grey dashed ends).

7 · Histogram equalisation on the slide 34 table

Topic 2 · slides 31-34



Equalisation **spreads out the most frequent intensities** through the intensity range (slide 31: the dull car photo becomes clear). Notation (slide 32): n pixels, grey levels 0…m, n_k pixels at level k, probability n_k/n. Transform (slide 33): $\mathbf{s_k = \Sigma_{i=0..k}\, n_i/n}$, and the new value is **round($\mathbf{s_k \times m}$)**.

Step by step on slide 34: 1. n = 790 + 1023 + … + 81 = **4096**; levels 0–7, so **m = 7**. 2. running total of n_k: 790, 1813, 2663, 3319, 3648, 3893, 4015, 4096. 3. divide by 4096: s_0 = 0.193, s_1 = 0.443, s_2 = 0.650 … s_7 = 1. 4. × 7 and round: 0.193 × 7 = 1.35 → **1**; 0.443 × 7 = 3.10 → **3**; 0.650 × 7 = 4.55 → **5**; and so on (full table above).

5. move the counts: level 6 gets 656 + 329 = **985** and level 7 gets 245 + 122 + 81 = **448**. The bars now use levels 1–7, and the busy dark levels are pushed apart.

8 · Exam traps

Topic 2 · slides 11, 17, 21, 23, 28, 33 added

- **Sampling vs quantization:** sampling digitises the *coordinates* (how many pixels); quantization digitises the *amplitude* (how many grey levels). Don't swap them (slides 11, 17).
- **8-bit = 256 levels numbered 0–255**, 0 black and 255 white (slide 23), not 1–256.
- **n_k is a count of pixels**, not an intensity. The bars of a histogram add up to n (slide 21).
- **Stretching:** subtract **c** (the input low end), divide by **(d − c)**, then add **a** (slide 28). Mixing up a/c is the usual slip.
- **Equalisation:** use the *running* sum of n_i/n, and multiply by **m**, the top level (7 for levels 0–7), not by the number of levels 8 (slide 33).
- **Equalisation can merge levels** (3 and 4 both go to 6) but never splits one level into two.

9 · Quick check

Topic 2 · slides 12, 23, 28, 33–34 added

▼ **Q1.** A 16 × 16 sampling grid with 4-bit levels: how many samples, and how many grey levels?

16 × 16 = 256 samples (slide 12 logic). 4 bits give $2^4 = 16$ levels, 0–15 (slide 23 does the 8-bit case: 256 levels).

▼ **Q2.** Stretch [c, d] = [20, 70] to [a, b] = [0, 250]. What is I_{out} for $I_{in} = 45$?

(45 − 20) × 250/50 + 0 = 25 × 5 = 125 (slide 28).

▼ **Q3.** In the slide 34 example, why does level 0 become 1 and not stay 0?

$s_0 = 790/4096 = 0.193$, and 0.193 × 7 = 1.35, which rounds to 1 (slide 33). Level 0 holds about a fifth of the pixels, so it moves up.

Key terms

electromagnetic spectrum · monochromatic / chromatic light · radiance · luminance · brightness · single sensor · strip · array · f = i · r · illumination · reflectance · sampling · quantization · quantization error · 2-D array · $h(r_k) = n_k$ · contrast stretching · 5th / 95th percentile · histogram equalisation · cumulative distribution

Exam relevance

Try each question yourself first, then reveal the answer.

Q1. "Explain image sampling and quantization. What is quantization error?"

Answer: A sensor outputs a **continuous** signal. **Sampling** digitises the spatial **coordinates** (x, y), so it decides how many pixels there are: more samples means more spatial detail. **Quantization** digitises the **amplitude**: the intensity range is split into equal discrete levels and each sample takes the nearest level. **Quantization error** is the difference between the true continuous value and the level it was given. Fewer levels means a bigger error and visible banding.

Q2. "Describe the three ways of acquiring an image and the simple image formation model."

Answer: (1) A **single sensor** that must move in x and y to build up the 2-D image. (2) A **sensor strip** that captures a line at a time while it or the object moves. (3) A **sensor array**, a 2-D grid that captures the whole frame at once (cameras). **Formation model:** `f(x,y) = i(x,y)·r(x,y)`, the illumination falling on the scene times the reflectance of the surface. Intensities are proportional to the energy radiated from the source.

Q3. "Define the intensity histogram. What does it tell you about an image?"

Answer: For levels in [0, L−1], $\mathbf{h(r_k) = n_k}$: the number of pixels with intensity r_k (x-axis = level, y-axis = count). Bunched at the low end means a dark image, at the high end a bright image, and in a narrow band low contrast. Spread across the range means high contrast. Normalising, $p(r_k) = n_k/n$, gives the probability of each level.

Q4. "Apply contrast stretching: input range [50, 150] mapped to [0, 255]. What is the output for 100? What is the weakness of the method, and how is it fixed?"

Answer: $I_{out} = (I_{in} - c)(b - a)/(d - c) + a = (100 - 50)(255 - 0)/(150 - 50) + 0 =$ **127.5** (about 128). **Weakness:** outliers. One extreme pixel sets c or d, and the stretch then has almost no effect. **Fixes:** choose c and d at the **5th and 95th percentiles** of the histogram, or take the histogram **peak**, set a cut-off and ignore data below it.

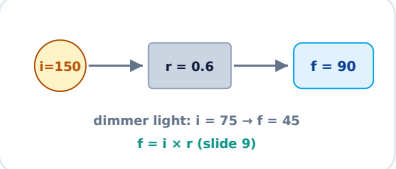
Q5. "Histogram-equalise an 8-level image (n = 4096) with n_k = 790, 1023, 850, 656, 329, 245, 122, 81."

Answer: Cumulative: 790, 1813, 2663, 3319, 3648, 3893, 4015, 4096. s_k = cum/4096: 0.193, 0.443, 0.650, 0.810, 0.891, 0.950, 0.980, 1.000. × m (7) and round: **0→1, 1→3, 2→5, 3→6, 4→6, 5→7, 6→7, 7→7**. The new histogram has 790 at level 1, 1023 at 3, 850 at 5, 985 at 6 and 448 at 7. It now covers the range much more evenly.

Practice: 10 questions

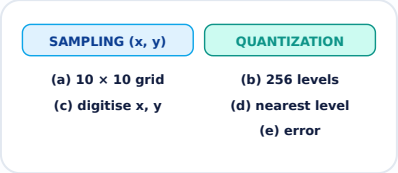
Exam-style questions on Topic 2: calculate, classify, outline, match, apply, explain, compare. Answer on paper first, then reveal. Each answer cites its slides; **added** marks anything that goes beyond the slides. "Show all answers" above opens these too.

P1. **CALCULATE** Using the simple image formation model, a surface with r(x,y) = 0.6 is lit with i(x,y) = 150. What is f(x,y)? What is f if the light is halved?



Answer · Topic 2 slide 9**:** f(x,y) = i(x,y) r(x,y) = 150 × 0.6 = **90**. Halving the light gives i = 75, so f = 75 × 0.6 = **45**. The value recorded follows the energy reaching the sensor, so it halves too. *Exam tip:* write the formula first, then substitute.

P2. **CLASSIFY** Sampling or quantization? (a) choosing a 10 × 10 grid of pixels (b) using 256 grey levels (c) digitising the x and y coordinates (d) giving each value the level it is vertically closest to (e) the difference between the true value and the stored level.



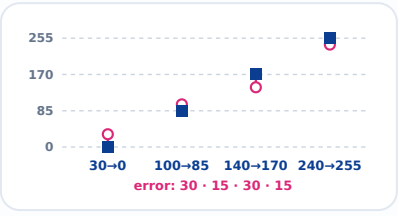
Answer · Topic 2 slides 11–12, 17–18**:** **Sampling:** (a) and (c), because they decide *where* values are taken. **Quantization:** (b), (d) and (e), because they decide *which value* is stored and what is lost doing it (quantization error). *The rule:* positions → sampling; intensity → quantization.

P3. **OUTLINE** Outline the difference between monochromatic and chromatic light, naming the quantities used to describe a chromatic light source.



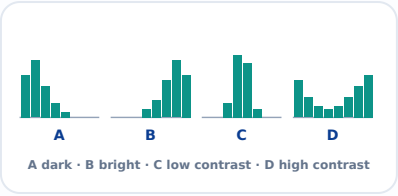
Answer · Topic 2 slides 4–5**:** **Monochromatic** light is void of colour; its intensity is given by a **grey level**. **Chromatic** light is colour light; besides its **frequency**, three basic quantities describe the source: **radiance** (watts), **luminance** (lumens) and **brightness**. Light itself is the part of the EM spectrum the eye can sense, and the colour of an object comes from the light it reflects.

P4. **CALCULATE** A 0–255 range is quantized to four equal levels: 0, 85, 170, 255. Quantize the samples 30, 100, 140 and 240 and give the quantization error of each.



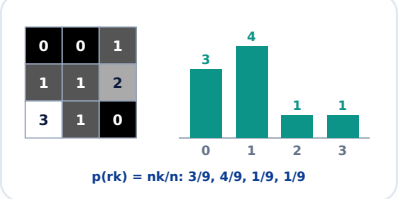
Answer · Topic 2 slides 17–18**:** Give each sample the level it is **vertically closest** to: 30 → **0** (error 30), 100 → **85** (error 15), 140 → **170** (error 30), 240 → **255** (error 15). The quantization error is the gap between the true value and the stored level. With only four levels the steps are 85 apart, so errors up to 42.5 are possible; more levels means smaller errors.

P5. **MATCH** Match each histogram shape to an image: (A) piled up at the left (B) piled up at the right (C) a narrow hump in the middle (D) spread over the whole range. Images: bright, high contrast, dark, low contrast.



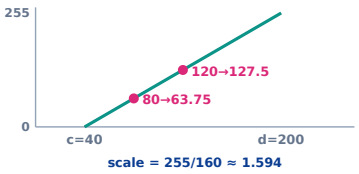
Answer · Topic 2 slides 24–27**:** **A → dark** (most pixels near 0, slide 24). **B → bright** (near 255, slide 25). **C → low contrast** (a narrow band of levels, slide 26). **D → high contrast** (spread across 0–255, slide 27). Remember the axes (slide 23): x = intensity, y = number of pixels.

P6. **CALCULATE** A 3 × 3 image has rows (0, 0, 1), (1, 1, 2), (3, 1, 0) with levels 0–3. Give $h(r_k)$ for each level and the probability n_k/n.



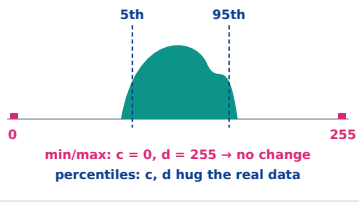
Answer · Topic 2 slides 21, 32**:** n = 9 pixels. Tally: level 0 → **3**, level 1 → **4**, level 2 → **1**, level 3 → **1** (check: 3 + 4 + 1 + 1 = 9). Probabilities n_k/n: **0.333, 0.444, 0.111, 0.111** (they add up to 1). The image is mostly dark: 7 of 9 pixels sit at levels 0–1.

P7. APPLY An image only uses intensities 40–200. Stretch it to 0–255 and find I_{out} for $I_{in} = 80$ and $I_{in} = 120$.



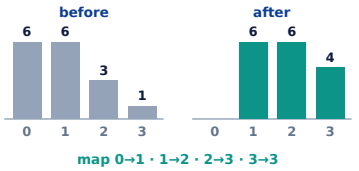
Answer · Topic 2 slides 28–29**:** Here c = 40, d = 200, a = 0, b = 255, so $I_{out} = (I_{in} - 40) \times 255/160$. $I_{in} = 80$: $40 \times 255/160 =$ **63.75 ≈ 64**. $I_{in} = 120$: $80 \times 255/160 =$ **127.5 ≈ 128**. Check the ends: 40 → 0 and 200 → 255. Write c, d, a, b down before substituting.

P8. EXPLAIN A photo has almost all its pixels between 90 and 160, but one stuck pixel is 0 and one is 255. Explain why plain contrast stretching fails here and how the slides fix it.



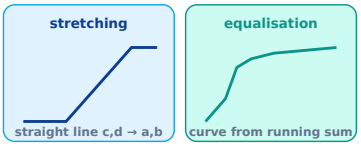
Answer · Topic 2 slide 30**:** If c and d are taken as the smallest and largest values, the two **outliers** give c = 0 and d = 255. With a = 0, b = 255 the formula becomes $I_{out} = I_{in}$, so the image is unchanged. **Fixes** (slide 30): pick c and d at the **5th and 95th percentiles** of the histogram (about 90 and 160 here), or find the histogram **peak**, set a **cut-off**, and ignore the data below it.

P9. CALCULATE Equalise a 16-pixel image with levels 0–3 (m = 3) and $n_k = 6, 6, 3, 1$.



Answer · Topic 2 slides 32–33**:** Running totals: 6, 12, 15, 16. s_k = total/16: 0.375, 0.75, 0.9375, 1. × m = 3: 1.125, 2.25, 2.81, 3. Round: **0 → 1, 1 → 2, 2 → 3, 3 → 3**. New histogram: level 1 = 6, level 2 = 6, level 3 = 3 + 1 = 4. Levels 2 and 3 have merged, and the busy dark levels are pushed up the scale.

P10. COMPARE Compare contrast stretching and histogram equalisation: what each uses from the histogram, the shape of the mapping, and one weakness.



Answer · Topic 2 slides 28–33**:** **Stretching** uses only two points of the histogram, c and d, and maps [c, d] to [a, b] with a **straight line**. It is spoiled by **outliers** unless c and d come from percentiles or a cut-off. **Equalisation** uses the **whole histogram**: the mapping is round($s_k \times m$) with s_k the running sum of n_i/n, so busy levels get spread out. Its weakness added: rounding can **merge** levels, and you don't choose the output range.

Quick MCQs: check your understanding

M1. According to the slides, what is light? CHOOSE 1

A. Any electromagnetic radiation
B. The electromagnetic radiation the human eye can sense
C. Only the infrared part of the spectrum
D. Radiation measured in lumens only

M2. The colour we see in an object is set by… CHOOSE 1

A. the light the object absorbs
B. the light reflected from the object
C. the sensor in our eye only
D. the object's temperature

M3. Which quantities are used, besides frequency, to describe a chromatic light source? **CHOOSE ALL THAT APPLY**

A. Radiance
B. Luminance
C. Brightness
D. Grey level

M4. Moving to the left of the spectrum on slide 3 (towards gamma rays) gives… **CHOOSE 1**

A. longer wavelength, lower energy
B. shorter wavelength, higher frequency and energy
C. the visible band
D. lower frequency, higher energy

M5. With a single sensor, how is a 2-D image formed? **CHOOSE 1**

A. All pixels are captured at once
B. The sensor moves in x and y
C. A strip of sensors sweeps once
D. The image is computed from f = i + r

M6. In f(x,y) = i(x,y) r(x,y), the two factors are combined by… **CHOOSE 1**

A. addition
B. multiplication
C. division
D. taking the maximum

M7. Sampling digitises… **CHOOSE 1**

A. the amplitude (intensity)
B. the x and y coordinates
C. the colour model
D. the histogram

M8. Which statements about quantization are true? **CHOOSE ALL THAT APPLY**

A. The range from black to white is divided into equal parts
B. Each value takes the level it is vertically closest to
C. It decides how many pixels the image has
D. It produces quantization error

M9. A 10 × 10 sampling grid gives how many samples? **CHOOSE 1**

A. 20
B. 100
C. 256
D. 1000

M10. An 8-bit grey-scale image has… **CHOOSE 1**

A. 8 levels, 0–7
B. 255 levels, 1–255
C. 256 levels, 0 = black and 255 = white
D. 256 levels, 0 = white and 255 = black

M11. In h(rk) = nk, what is nk? **CHOOSE 1**

A. the k-th intensity value
B. the number of pixels with intensity rk
C. the total number of pixels
D. the probability of level k

M12. Which pairs of histogram shape and image are correct? **CHOOSE ALL THAT APPLY**

A. bunched at the low end → dark image
B. bunched at the high end → bright image
C. a narrow band in the middle → high contrast
D. spread across the whole range → high contrast

M13. Stretch [c, d] = [50, 100] to [a, b] = [0, 200]. What is Iout for Iin = 75? **CHOOSE 1**

A. 75
B. 100
C. 150
D. 37.5

M14. Which are true of histogram equalisation as taught? **CHOOSE ALL THAT APPLY**

A. It uses the running sum of ni/n
B. New value = round(sk × m)
C. It is a straight-line map from [c, d] to [a, b]
D. Two old levels can map to the same new level

Answer key

1. **M1: B**. Light is the EM radiation that can be sensed by the human eye. The spectrum itself also covers gamma, X-ray, microwave and radio. **Topic 2 · slides 3–4**
2. **M2: B**. The colour humans perceive in an object is determined by the nature of the light reflected from it. **Topic 2 · slide 4**
3. **M3: A, B, C**. Radiance (watts), luminance (lumens) and brightness. Grey level describes monochromatic light. **Topic 2 · slide 5**
4. **M4: B**. Left = shorter wavelength, higher frequency, higher energy; right (radio) = the opposite. **Topic 2 · slide 3**
5. **M5: B**. A single sensor has to move in both x and y to build up the 2-D image. A 2-D array captures the grid of positions directly. **Topic 2 · slides 6–8**
6. **M6: B**. The model is a product: f = i × r, so halving i halves f. **Topic 2 · slide 9**
7. **M7: B**. Sampling digitises the coordinates; quantization digitises the amplitude. **Topic 2 · slides 10–11**
8. **M8: A, B, D**. Quantization works on intensity: equal levels, nearest level, and an error between true and stored value. The number of pixels is set by sampling. **Topic 2 · slides 17–18**
9. **M9: B**. 10 on x and 10 on y gives 10 × 10 = 100 samples (pixels). **Topic 2 · slide 12**
10. **M10: C**. 8 bits give 256 levels from 0 to 255, with 0 black and 255 white. **Topic 2 · slide 23**
11. **M11: B**. rk is the k-th intensity; nk counts the pixels that have it. nk/n would be the probability. **Topic 2 · slide 21**
12. **M12: A, B, D**. A narrow band means LOW contrast. Dark = bunched left, bright = bunched right, high contrast = spread out. **Topic 2 · slides 24–27**
13. **M13: B**. (75 − 50) × (200 − 0)/(100 − 50) + 0 = 25 × 4 = 100. **Topic 2 · slide 28**
14. **M14: A, B, D**. sk is the cumulative sum and the new level is round(sk × m); in the slide 34 example levels 3 and 4 both become 6. The straight line is contrast stretching. **Topic 2 · slides 33–34**

L03 Arithmetic, logic & geometric operations

Operations that change pixel *values* (arithmetic and logical) and operations that change pixel *positions* (geometric transforms), then how a transformed image is filled back in by resampling and interpolation, and why aliasing appears. slides

Short Note — Quick Revision

- **Arithmetic** works pixel by pixel on images of the same size: **add** min(255, A+B) · **subtract** max(0, A−B), or |A−B| for change · **multiply** A·B for masking · **divide** A/(B+ε) · **average** $(1/N)\Sigma I_i$ for noise.
- **Logical** (AND, OR, NOT, XOR) works on **binary images and masks**: region selection and segmentation. Arithmetic risks overflow and clipping; logical loses the intensity range.
- **Geometric:** p′ = T(p). **Scale** [Sx 0; 0 Sy] · **rotate** [cos −sin; sin cos] · **shear** [1 kx; ky 1] · **reflect** Fx, Fy · **translate** x+tx, y+ty.
- **Homogeneous coordinates** (x, y, 1) turn translation into a matrix multiply, so a chain of transforms becomes one product (**affine**).
- **Forward mapping** leaves holes. **Inverse mapping** asks where each output pixel came from, then **interpolates**: **nearest neighbour** (fast, jagged) or **bilinear** (4 neighbours, smooth).
- **Aliasing** = sampling too coarsely gives jagged edges and moiré. **Anti-aliasing** reduces it.

Arithmetic operations: pixel by pixel

Pixel by pixel: A op B (8-bit, so results are clipped to 0…255)

A			B			C	
200	100	+	100	100	=	255	200
50	0		30	10		80	10

Addition
C = min(255, A + B): 200+100 → 255 (clipped)

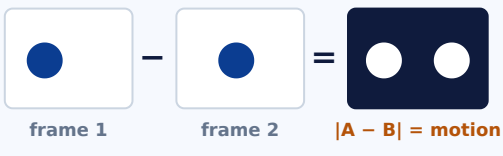
A			B			C	
200	100	−	100	100	=	100	0
50	0		30	10		20	0

Subtraction
C = max(0, A − B): 0−10 → 0 (clipped)

For two images of the same size, each output pixel uses only the two pixels at the same (x, y). An 8-bit result must stay in 0…255, so addition **saturates** at 255 and subtraction at 0.

Discussion (following the lecture)

Arithmetic operations



Arithmetic operations change pixel **intensity values** directly and are applied **pixel by pixel** to two images A and B of the same size.

OPERATION	DEFINITION	USED FOR
Addition	$C = \min(255, A + B)$	combining several exposures · adding a constant to brighten · averaging to reduce noise
Subtraction	$C = \max(0, A - B)$; change detection uses $\lvert A - B \rvert$	motion detection between video frames · finding defects in inspection · background subtraction
Multiplication	$C = A \cdot B$ (element-wise)	**masking**: a mask M(x,y) of 0/1 removes or keeps pixels
Division	$C = A / (B + \varepsilon)$	relative reflectance · normalising intensity · correcting uneven illumination (ε avoids divide-by-zero)
Averaging	$\bar{I}(x,y) = (1/N) \Sigma I_i(x,y)$	reduces random noise in a **still** scene: astronomy, medical imaging (e.g. the average of 500 frames)

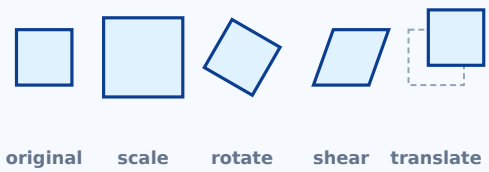
Logical operations



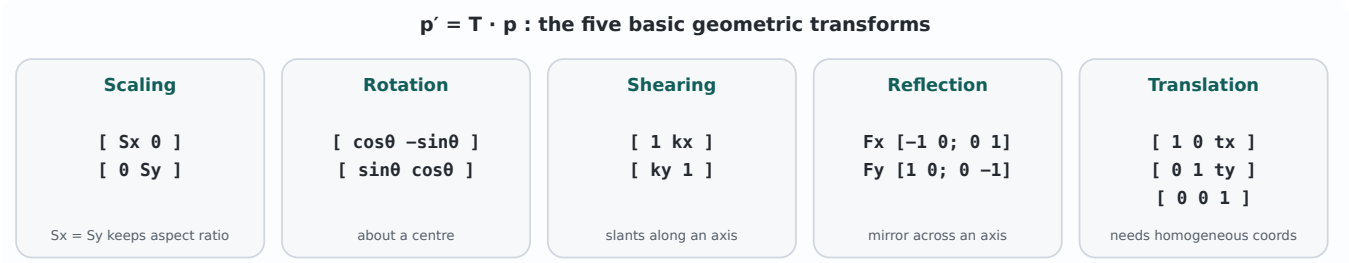
Logical operations work mostly on **binary images and masks**, using Boolean algebra: a pixel is foreground/background or true/false. The main ones are **AND** (keep where both are set, e.g. crop to a region of interest), **OR** (union of regions), **NOT** (invert a mask) and **XOR** (where two masks differ, a simple change map).

ASPECT	ARITHMETIC	LOGICAL
Main advantage	quantitative change of intensities: brightness, contrast, blending, differences	precise selection, extraction and combination of regions with masks
Main disadvantage	overflow, underflow and loss of information from clipping, rounding and scaling	binary only, so the full intensity range is lost
Typical limitation	needs normalisation or saturation to stay in range	needs a good threshold or mask to find meaningful regions
Best for	enhancement, blending, intensity transforms, differencing	masking, binary segmentation, isolating objects, region selection

Geometric operations



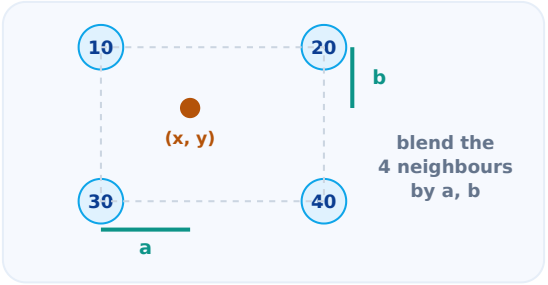
Arithmetic changes *values* at fixed locations. Geometric operations change the **coordinates**, that is, where the content sits. They are the basis of **image registration, object alignment, camera calibration, rectification, data augmentation** and recognition. Each maps an original coordinate p to a new one: **$p' = T(p)$**.



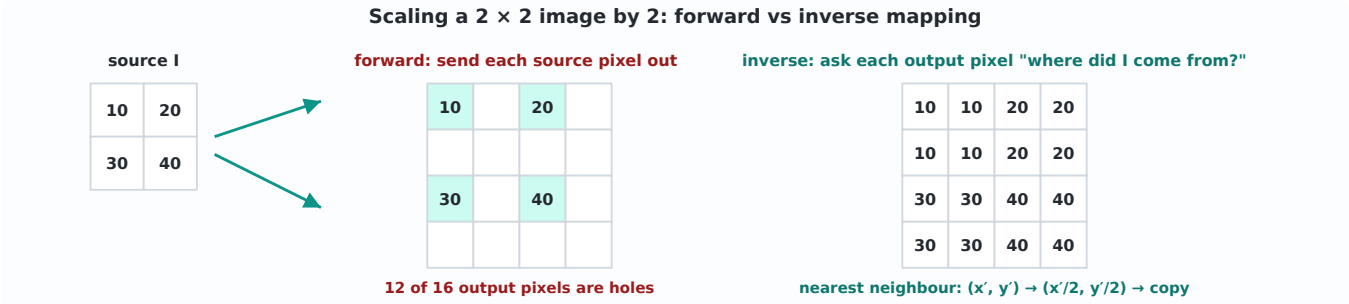
The 2 × 2 matrices handle scaling, rotation, shearing and reflection. A reflection about any line through the origin at angle θ is `[cos2θ sin2θ; sin2θ −cos2θ]`. Translation ($x' = x + t_x$, $y' = y + t_y$) is **not** a 2 × 2 linear map, so it needs a 3 × 3 matrix in homogeneous coordinates.

Homogeneous coordinates add a third coordinate set to 1: (x, y) → (x, y, 1). Translation then becomes a matrix multiplication like the others, so a whole chain of transforms (for example rotate, then scale, then shift) can be **combined into one matrix** by multiplying. Combinations like these are **affine transformations**.

Resampling and interpolation



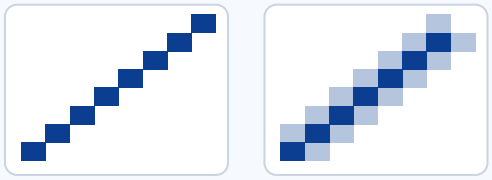
After a transform the new coordinates rarely land exactly on whole pixel positions. **Forward mapping** sends each source pixel to its new place. When scaling up, this leaves **holes** (and when shrinking, overlaps). **Inverse mapping** asks, for every output pixel, "*where did this pixel come from in the original?*", and computes $p = T^{-1}(p')$. That position usually falls *between* source pixels, so its value is estimated by **interpolation**.



The slide example: I = [10 20; 30 40] scaled by 2. Forward mapping fills only 4 of the 16 pixels. Inverse mapping with nearest-neighbour interpolation fills every pixel; bilinear would blend instead (e.g. 15 between 10 and 20).

METHOD	HOW	PROS	CONS
Nearest neighbour	I′(x,y) = I(round(x), round(y)): copy the closest source pixel	cheap · keeps the original intensity values	jagged edges · visible aliasing
Bilinear	weighted average of the 4 surrounding pixels, using the fractional offsets a, b: $(1-a)(1-b)\cdot I_{00} + a(1-b)\cdot I_{10} + (1-a)b\cdot I_{01} + ab\cdot I_{11}$	smooth result	blurs a little · creates new intensity values · more work

Aliasing and anti-aliasing



aliased: jaggies | anti-aliased: soft

Aliasing is an unwanted effect that appears when an image is sampled at **too low a resolution** for its fine detail: *jagged edges* on diagonals and curves, *false patterns* (moiré), and lost or distorted detail. **Anti-aliasing** reduces these artefacts, typically by smoothing (low-pass filtering) before sampling or by using a better interpolation.

Step by step — deeper notes

Everything here follows the Topic 3 slides; each part shows its slide numbers. Parts marked **added** are study aids we wrote, not lecture content.

1 · Arithmetic, pixel by pixel, with numbers

Topic 3 · slides 3–13

operation	A			B (or mask M)			C		what happened
Add	200	90	+	100	40	=	255	130	min(255, A+B): 300 → 255
Subtract	90	40	−	60	70	=	30	0	max(0, A−B): −30 → 0
|A − B|	90	40	−	60	70	=	30	30	both changes kept
Multiply	90	200	×	1	0	=	90	0	mask: 1 keeps, 0 removes
Divide	100	50	÷	200	100	=	0.5	0.5	A/(B+ε): light evened out
Average N=4	104	97	+	101	98	=	100		(1/N)ΣI: 400 ÷ 4 = 100

one pixel position (x, y) at a time; every row uses 8-bit values 0…255

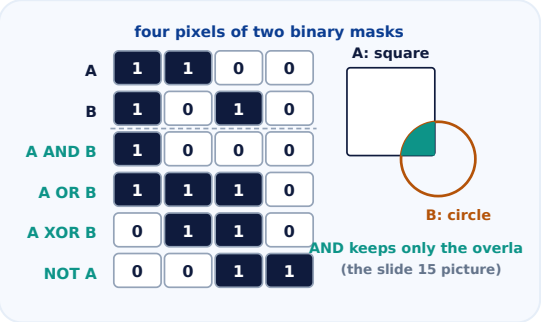
Slide 3: arithmetic works **pixel by pixel** on two images A and B of equal size, so each row above is one or two pixel positions.

Add (slide 4) clips at 255: 200 + 100 = 300 → **255**. **Subtract** (slide 5) clips at 0: 40 − 70 = −30 → **0**, so that change is lost; **|A − B|** keeps it (30). Slide 7 subtracts a smooth bright-to-dark background from a microscope image to flatten the lighting.

Multiply by a 0/1 mask M (slides 8–9: a white circle keeps the region of interest). **Divide** A/(B + ε) (slides 10–11): the same surface lit 200 or 100 gives 100/200 = 50/100 = **0.5**, so uneven light cancels; ε stops a divide by zero. **Average** N frames of a still scene (slides 12–13, 500 images): noise of +4, −3, +1, −2 cancels to the true 100.

2 · Logical operations as truth tables on pixels

Topic 3 · slides 14–16



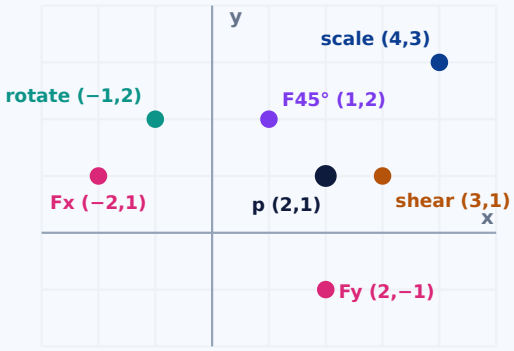
Slide 14: logical operations are for **binary images and masks**; 1 = foreground (true), 0 = background (false). Read each column of the grid as one pixel.

AND is 1 only where both are 1 · **OR** where either is 1 · **XOR** where they *differ* · **NOT** flips. Slide 15 does this with a square A and a circle B: AND leaves the small overlap, OR the union, XOR the union minus the overlap, NOT turns each mask inside out.

Slide 16 sums up the trade-off: arithmetic changes intensities by an amount but can overflow or clip; logical selects regions precisely but keeps only two states and needs a good threshold or mask.

3 · One point through the four 2 × 2 transforms

Topic 3 · slides 18–22



	T × (2, 1)	
Scaling	[2 0; 0 3]	(4, 3)
Rotation 90°	[0 −1; 1 0]	(−1, 2)
Shear kx=1	[1 1; 0 1]	(3, 1)
Reflection Fx	[−1 0; 0 1]	(−2, 1)
Reflection Fy	[1 0; 0 −1]	(2, −1)
F, θ = 45°	[0 1; 1 0]	(1, 2)

cos 90° = 0, sin 90° = 1; for F: cos 2θ = 0, sin 2θ = 1

Slide 18: a geometric operation moves the *coordinates*: **p′ = T p** with p = (x, y). Take one point **p = (2, 1)** and apply each 2 × 2 matrix from slides 19–22.

Scale S_x=2, S_y=3: (2·2, 3·1) = (4, 3); $S_x \neq S_y$, so the aspect ratio changes. **Rotate** θ = 90°: (cos·2 − sin·1, sin·2 + cos·1) = (−1, 2). **Shear** k_x=1, k_y=0: (2 + 1·1, 1) = (3, 1).

Reflect: F_x negates x → (−2, 1); F_y negates y → (2, −1). The general F with θ = 45° becomes [0 1; 1 0], which swaps x and y → (1, 2): a mirror in the line y = x.

4 · Translation and homogeneous coordinates

Topic 3 · slides 23–24

① translate (2, 1) by tx = 3, ty = 4

$$\begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 4 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \\ 1 \end{bmatrix}$$

→ (5, 5)

2+3, 1+4

② rotate 90°, then translate: one matrix

$$T \cdot R = \begin{bmatrix} 0 & -1 & 3 \\ 1 & 0 & 4 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 6 \\ 1 \end{bmatrix}$$

check by hand: R(2, 1) = (−1, 2), then + (3, 4) = (2, 6) ✓

the other order, R · T: (2, 1) → (5, 5) → rotate → (−5, 5), a different point

Slide 23: translation $\mathbf{x' = x + t_x}$, $\mathbf{y' = y + t_y}$ adds a constant, and a 2 × 2 matrix cannot do that. Slide 24's fix: write the point with a third coordinate equal to one, **(x, y, 1)**, and use a 3 × 3 matrix.

① The last column holds t_x, t_y, so (2, 1, 1) → (5, 5, 1). ② Because every transform is now a matrix multiply, a chain collapses into **one** matrix (slide 24: affine transformations). The 2 × 2 rotation sits in the top-left corner.

added The order of multiplication matters: rotate-then-translate (T · R) and translate-then-rotate (R · T) land on different points.

5 · Inverse mapping, one row at a time

Topic 3 · slides 25–29

source I (slide 26)

10	20
30	40

x = 0 x = 1

inverse map one row: x = (x′ + 0.5)/2 − 0.5

x′	x	round	I
0	−0.25	0	10
1	0.25	0	10
2	0.75	1	20
3	1.25	1*	20

* clamped to the last column (x ≤ 1)

output row 0 (NN)

10	10	20	20

forward mapping row 0:

10		20	

(x, y) → (2x, 2y): gaps at x′ = 1, 3

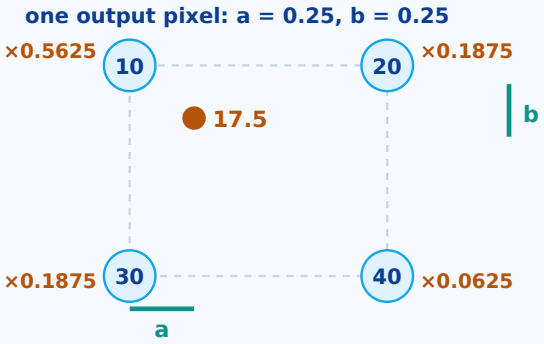
Slides 25–27: after a transform the new coordinates rarely fall on whole pixels. **Forward mapping** pushes each source pixel to (2x, 2y), so in a 4 × 4 output only 4 pixels are hit and the rest are holes. **Inverse mapping** asks, for each output pixel, *"where did this pixel come from?"* (T^{-1}, slide 27).

For scale 2 the inverse is "divide by 2". **added** We measure from pixel centres, x = (x′ + 0.5)/2 − 0.5, a common convention the slides don't specify. Then **nearest neighbour** (slide 29) rounds: I′(x, y) = I(round x, round y). Row 0 becomes 10 10 20 20, matching the 4 × 4 result in the Discussion.

Slide 28: because x = 0.25 or 0.75 is *between* pixels, the value must be **estimated** by interpolation; NN and bilinear are the two main ways.

6 · Bilinear interpolation: weights worked out

Topic 3 · slides 30–31



bilinear 4 × 4

10	12.5	17.5	20
15	17.5	22.5	25
25	27.5	32.5	35
30	32.5	37.5	40

nearest neighbour 4 × 4

10	10	20	20
10	10	20	20
30	30	40	40
30	30	40	40

same output pixel (1, 1): bilinear 17.5 · NN 10

Slide 30: bilinear interpolation estimates a pixel lying **between four neighbours**, weighting each by how close it is:

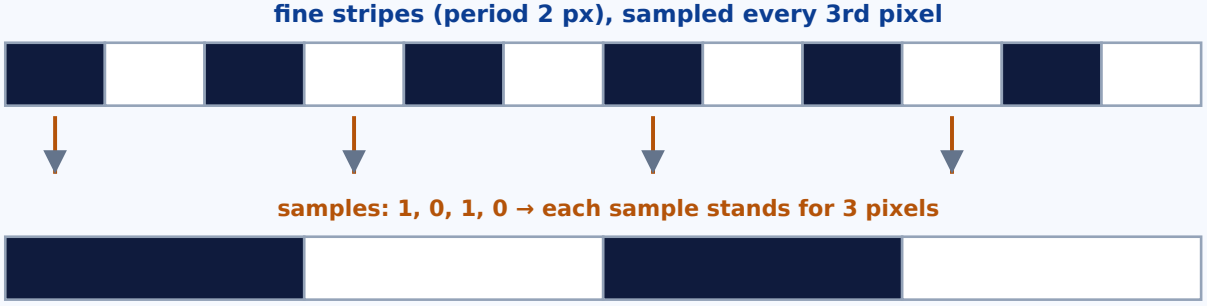
$I(x, y) = (1-a)(1-b)\cdot I(x_0,y_0) + a(1-b)\cdot I(x_0+1,y_0) + (1-a)b\cdot I(x_0,y_0+1) + ab\cdot I(x_0+1,y_0+1)$.

Output pixel (1, 1) maps back to (0.25, 0.25), so a = b = 0.25. Weights: 0.75·0.75 = **0.5625**, 0.25·0.75 = **0.1875** (twice), 0.25·0.25 = **0.0625** (they add to 1). Value = 5.625 + 3.75 + 5.625 + 2.5 = **17.5**. Nearest neighbour just copies **10**.

Bilinear makes new in-between values (12.5, 22.5 …) and a smooth ramp; NN keeps only 10/20/30/40 in blocks. Slide 31 compares the two on a photo (and also shows bicubic). *Note:* slide 30 prints all four terms as $I(x_0, y_0)$; the four different neighbours are meant.

7 · Aliasing: when sampling invents a pattern

Topic 3 · slides 29, 32–34



Slide 32: **aliasing** happens when an image is sampled at **too low a resolution** for its fine detail. In images it shows as jagged edges on diagonals and curves, **false patterns (moiré)**, and lost or distorted detail.

added The diagram is a 1-D example: stripes that repeat every 2 pixels, kept at only every 3rd pixel, come out as stripes 6 pixels wide. Slide 33 shows the same thing on a striped shirt: the left copy has coloured wavy bands that aren't on the shirt.

Slide 29 links this to resampling: nearest neighbour "can cause noticeable aliasing". **Anti-aliasing** (slide 34) is the process of reducing these artefacts; the slides define it without naming a method.

8 · Exam traps

Topic 3 · slides 5, 12, 14, 22–24, 26–30 added

- **max(0, A−B) is not change detection:** where B is brighter the change clips to 0. Use **|A − B|** (slide 5).
- **Averaging needs a still scene** (slide 12): anything that moves between frames blurs.
- **Logical ops are for binary masks** (slide 14), not grey-level intensities.
- **F_x negates x** (mirror across the y-axis); F_y negates y (slide 22). Don't swap them.
- **Translation is not a 2 × 2 matrix** (slide 23): it needs (x, y, 1) and a 3 × 3 matrix (slide 24). When combining, the order of the matrices matters.
- **Forward mapping leaves holes**; inverse mapping + interpolation fills every pixel (slides 26–28).
- **NN keeps the original values, bilinear creates new ones** (slides 29–30).

9 · Quick check

Topic 3 · slides 5, 23–24, 30 added

▼ **Q1.** A pixel is 100 in frame A and 180 in frame B. What do max(0, A − B) and |A − B| give?

max(0, −80) = **0**, so the change is hidden; |−80| = **80** shows it (slide 5).

▼ **Q2.** Translate (1, 2) by $t_x = 4$, $t_y = -1$ using homogeneous coordinates.

[1 0 4; 0 1 −1; 0 0 1] × (1, 2, 1) = (5, 1, 1), so the point moves to **(5, 1)** (slides 23–24).

▼ **Q3.** A point sits exactly in the middle of four pixels 0, 100, 100, 200. What does bilinear interpolation give?

a = b = 0.5, so every weight is 0.25: 0.25 × (0 + 100 + 100 + 200) = **100** (slide 30).

Key terms

pixel-by-pixel · saturation / clipping · image addition · image subtraction · |A − B| change detection · background subtraction · masking · image division · image averaging · AND · OR · NOT · XOR · binary mask · p′ = T(p) · scaling · rotation · shearing · reflection · translation · homogeneous coordinates · affine transformation · forward / inverse mapping · nearest-neighbour interpolation · bilinear interpolation · aliasing · moiré · anti-aliasing

Exam relevance

Try each question yourself first, then reveal the answer.

Q1. "Define image addition and subtraction for 8-bit images and give two applications of each."

Answer: **Addition:** C(x,y) = min(255, A(x,y) + B(x,y)). Uses: combining several exposures, adding a constant to brighten, averaging frames to reduce noise. **Subtraction:** C(x,y) = max(0, A(x,y) − B(x,y)); for change detection use |A − B|. Uses: motion detection between video frames, defect detection in industrial inspection, background subtraction. The min/max keep the result inside the valid 0–255 range.

Q2. "How does image averaging reduce noise, and when does it work?"

Answer: Given N images of the same scene, $\bar{I}(x,y) = (1/N)\ \Sigma\ I_i(x,y)$. The true signal is the same in every frame, while random noise varies and averages towards zero, so the noise falls as N grows (its standard deviation drops by about $1/\sqrt{N}$). It works only for **still** scenes, where the content doesn't move between frames. Examples: astronomical and medical imaging.

Q3. "Compare arithmetic and logical operations on images."

Answer: **Arithmetic** changes intensities quantitatively: brightness, contrast, blending, differencing. Its drawback is overflow, underflow and information loss from clipping and rounding, so it often needs normalisation. **Logical** (AND, OR, NOT, XOR) works on binary images and masks to select, extract and combine regions precisely. Its drawback is that it keeps only binary states, not the intensity range, and it depends on a good threshold or mask. Arithmetic is best for enhancement and differencing; logical for masking, segmentation and selecting a region of interest.

Q4. "Why can't translation be written as a 2 × 2 matrix? How do homogeneous coordinates help?"

Answer: Translation adds a constant ($x' = x + t_x$, $y' = y + t_y$). A 2 × 2 matrix can only form linear combinations of x and y, and it always maps the origin to itself, so it can't add an offset. **Homogeneous coordinates** write the point as (x, y, 1), and translation becomes the 3 × 3 matrix [1 0 t_x; 0 1 t_y; 0 0 1]. All transforms then use the same multiply, so a sequence of them can be **combined into one matrix** (an affine transformation).

Q5. "Scale I = [10 20; 30 40] by a factor of 2. Explain forward vs inverse mapping and the result with nearest-neighbour interpolation."

Answer: **Forward mapping** sends each source pixel to (2x, 2y). Only 4 of the 16 output pixels get a value and the other 12 are **holes**. **Inverse mapping** takes each output pixel (x′, y′) and asks where it came from: (x′/2, y′/2). That is usually not a whole pixel position, so we interpolate. With **nearest neighbour** each source pixel fills a 2 × 2 block: [10 10 20 20; 10 10 20 20; 30 30 40 40; 30 30 40 40]. Bilinear interpolation would blend instead (e.g. 15 between 10 and 20), giving a smoother but slightly blurred result.

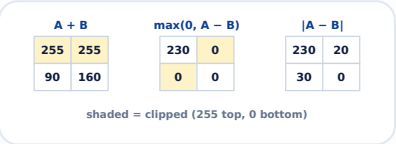
Q6. "Compare nearest-neighbour and bilinear interpolation, and explain aliasing."

Answer: **Nearest neighbour**, I′(x,y) = I(round x, round y), is cheap and keeps the original values, but it gives jagged edges and aliasing. **Bilinear** takes a weighted average of the 4 surrounding pixels using the fractional offsets a and b. It is smoother but slightly blurred, creates new values and costs more. **Aliasing** is the unwanted effect of sampling too coarsely for the detail: jagged diagonals and curves, moiré patterns, lost detail. **Anti-aliasing** reduces it, e.g. by smoothing before resampling.

Practice: 10 questions

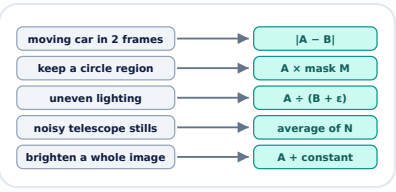
Exam-style questions on Topic 3: calculate, match, apply, classify, describe, compare, outline, explain. Answer on paper first, then reveal. Each answer cites its slides; **added** marks anything that goes beyond the slides. "Show all answers" above opens these too.

P1. **CALCULATE** For 8-bit images A = [250 120; 30 80] and B = [20 140; 60 80], compute A + B, max(0, A − B) and |A − B|. Which pixels were clipped?



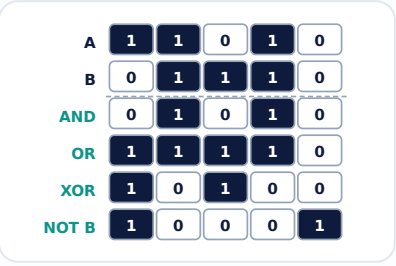
Answer · Topic 3 slides 4–5**:** **A + B** = [min(255, 270) min(255, 260); 90 160] = **[255 255; 90 160]**, both top pixels clipped at 255. **max(0, A − B)** = [230 max(0, −20); max(0, −30) 0] = **[230 0; 0 0]**, two pixels clipped at 0. **|A − B|** = **[230 20; 30 0]**: no clipping, and the two changes lost by subtraction reappear. Only the bottom-right pixel is truly unchanged.

P2. **MATCH** Match each task to the arithmetic operation the slides give for it: (a) spotting a moving car between two video frames (b) keeping only a circular region of interest (c) correcting non-uniform illumination (d) reducing random noise in still astronomical images (e) making the whole image brighter.



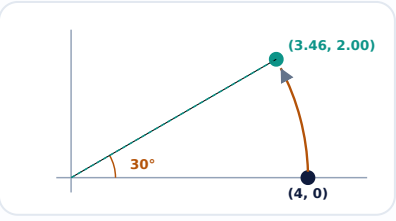
Answer · Topic 3 slides 4–12**:** (a) **subtraction, |A − B|**: motion detection between frames (slides 5–6). (b) **multiplication by a 0/1 mask** (slides 8–9). (c) **division A/(B + ε)** (slide 10); slide 7 also flattens a background gradient by subtraction. (d) **averaging** N frames of a stationary scene (slide 12). (e) **addition of a constant offset** (slide 4).

P3. **APPLY** Two binary masks along one row are A = 1 1 0 1 0 and B = 0 1 1 1 0. Work out A AND B, A OR B, A XOR B and NOT B. Which result is a simple “where do they differ” map?



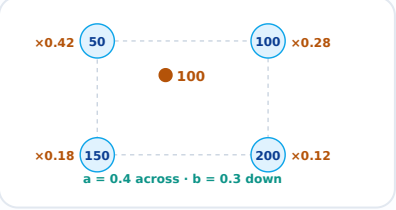
Answer · Topic 3 slides 14–15**:** **AND** = 0 1 0 1 0 (both 1). **OR** = 1 1 1 1 0 (either 1). **XOR** = 1 0 1 0 0 (exactly one is 1). **NOT B** = 1 0 0 0 1. **XOR** is the “differ” map: it is 1 only where the masks disagree, like the XOR panel on slide 15 (square and circle minus their overlap).

P4. **CALCULATE** Rotate the point p = (4, 0) by θ = 30° about the origin using the rotation matrix. Give the new coordinates to two decimals.



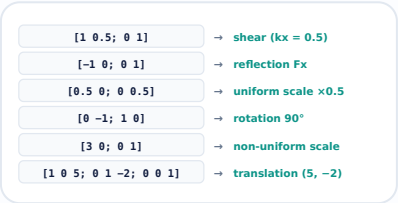
Answer · Topic 3 slide 20**:** R = [cos θ −sin θ; sin θ cos θ] with cos 30° = 0.866 and sin 30° = 0.5. $x' = 0.866 \cdot 4 - 0.5 \cdot 0 =$ **3.46**; $y' = 0.5 \cdot 4 + 0.866 \cdot 0 =$ **2.00**. So p′ = (3.46, 2.00). Check: the distance from the origin stays 4 ($3.46^2 + 2^2 \approx 16$), as a rotation should keep it. The answer is not a whole pixel position, which is why resampling (slide 25) is needed.

P5. **CALCULATE** A sample point lies at a = 0.4, b = 0.3 inside the four pixels $I(x_0,y_0) = 50$, $I(x_0+1,y_0) = 100$, $I(x_0,y_0+1) = 150$, $I(x_0+1,y_0+1) = 200$. Find its value by bilinear and by nearest-neighbour interpolation.



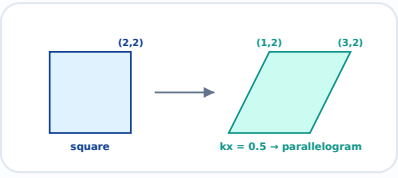
Answer · Topic 3 slides 29–30**:** Weights: (1−a)(1−b) = 0.6·0.7 = **0.42**; a(1−b) = 0.4·0.7 = **0.28**; (1−a)b = 0.6·0.3 = **0.18**; ab = 0.4·0.3 = **0.12** (sum 1). Bilinear = 0.42·50 + 0.28·100 + 0.18·150 + 0.12·200 = 21 + 28 + 27 + 24 = **100**. Nearest neighbour: round(0.4) = 0 and round(0.3) = 0, so it copies $I(x_0,y_0) =$ **50**. Bilinear gives a value not found among the four pixels; NN keeps an original one.

P6. **CLASSIFY** Name the transform each matrix performs: (a) [1 0.5; 0 1] (b) [−1 0; 0 1] (c) [0.5 0; 0 0.5] (d) [0 −1; 1 0] (e) [3 0; 0 1] (f) [1 0 5; 0 1 −2; 0 0 1].



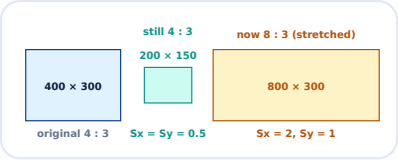
Answer · Topic 3 slides 19–24**:** (a) **shear** along x, $k_x = 0.5$ (slide 21). (b) **reflection F_x**, negates x (slide 22). (c) **uniform scaling**, $S_x = S_y = 0.5$, halves the size and keeps the aspect ratio (slide 19). (d) **rotation by 90°**: cos 90° = 0, sin 90° = 1 (slide 20). (e) **non-uniform scaling**, $S_x = 3 \neq S_y = 1$, stretches only x (slide 19). (f) **translation** by (5, −2) in homogeneous form; the only 3 × 3 one (slides 23–24).

P7. **DESCRIBE** Describe what shearing does, using the square with corners (0,0), (2,0), (2,2), (0,2) and $k_x = 0.5$, $k_y = 0$. Give two uses of shearing.



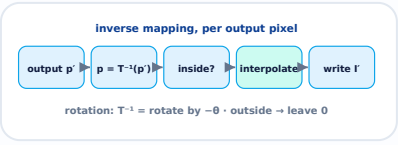
Answer · Topic 3 slide 21**:** Shearing **slants** the image along an axis: $x' = x + k_x \cdot y$, $y' = k_y \cdot x + y$. Corners: (0,0) → (0,0); (2,0) → (2,0); (2,2) → (2 + 1, 2) = **(3,2)**; (0,2) → **(1,2)**. The bottom edge stays put and the top slides right by 0.5 × its height, so the square becomes a **parallelogram**. Uses (slide 21): geometric correction, modelling affine distortions, generating augmented training images.

P8. **COMPARE** A 400 × 300 image is scaled with $(S_x, S_y) = (0.5, 0.5)$ and, separately, with (2, 1). Give each new size and compare the two kinds of scaling.



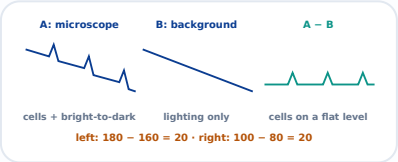
Answer · Topic 3 slide 19**:** (0.5, 0.5): 400·0.5 × 300·0.5 = **200 × 150**. Since $S_x = S_y$ the scaling is **uniform** and the aspect ratio stays 4 : 3. (2, 1): 800 × 300. $S_x \neq S_y$, so it is **non-uniform**: the ratio becomes 8 : 3 and shapes look stretched (a circle becomes an ellipse). Slide 19: "when $S_x = S_y$ scaling is uniform and preserves the aspect ratio, and vice versa."

P9. **OUTLINE** Outline the steps to rotate an image by θ using inverse mapping and an interpolation method from the slides.



Answer · Topic 3 slides 20, 25–30**:** ① Create the empty output image. ② For **every output pixel** $p' = (x', y')$, ask where it came from: $p = T^{-1}(p')$ (slide 27). **added** For a rotation the inverse is a rotation by −θ. ③ If p falls outside the source, leave the pixel empty (e.g. 0). ④ p is usually between pixels (slide 28), so estimate the value: **nearest neighbour** I(round x, round y) (slide 29) or **bilinear** from the 4 neighbours (slide 30). ⑤ Write that value to $I'(x', y')$. Every output pixel gets a value, so there are no holes.

P10. **EXPLAIN** Slide 7 shows a microscope image minus a smooth bright-to-dark image giving a cleaner result. Explain why subtraction helps here, using a cell edge of 180 on the bright side (background 160) and 100 on the dark side (background 80).



Answer · Topic 3 slides 5–7**:** The second image holds only the **uneven lighting** (a gradient), so A − B removes that slow trend and leaves the detail. Bright side: 180 − 160 = **20**; dark side: 100 − 80 = **20**. Before, the same cell edge read 180 or 100 depending on where it was; after, both read 20, so the cells look alike across the image. Values where B > A clip to 0 (max(0, A − B), slide 5). **added** In practice the result is often rescaled to use the full 0–255 range.

Quick MCQs: check your understanding

M1. Two 8-bit pixels are added: A = 200, B = 100. What is C = min(255, A + B)? **CHOOSE 1**

A. 300
B. 255
C. 44
D. 100

M2. Which formula should you use to detect change between two frames, whichever one is brighter? **CHOOSE 1**

A. max(0, A − B)
B. |A − B|
C. min(255, A + B)
D. A · B

M3. Which are applications of image subtraction listed in the slides? **CHOOSE ALL THAT APPLY**

A. Motion detection between video frames
B. Identifying defects in industrial inspection
C. Background subtraction
D. Adding a constant offset to increase brightness

M4. Image masking with a mask M(x, y) of 0s and 1s is done by which operation? **CHOOSE 1**

A. Addition
B. Subtraction
C. Multiplication
D. Division

M5. Why does image division use A / (B + ε)? **CHOOSE 1**

A. To brighten the result
B. To avoid dividing by zero where B = 0
C. To keep the result binary
D. To rotate the image

M6. Image averaging reduces random noise only when… **CHOOSE 1**

A. the scene is stationary across the N images
B. the images are binary
C. N = 2
D. the images are different sizes

M7. Logical operations (AND, OR, NOT, XOR) are primarily applied to… **CHOOSE 1**

A. colour images
B. binary images and masks
C. frequency spectra
D. 3-D models

M8. Which statements are disadvantages or limitations of arithmetic operations (not logical ones)? **CHOOSE ALL THAT APPLY**

A. Intensity overflow or underflow
B. Loss of information from clipping, rounding and scaling
C. Do not preserve the full range of intensities
D. Often need normalisation or saturation to stay in range

M9. Which of these can be written as a 2 × 2 matrix acting on (x, y)? **CHOOSE ALL THAT APPLY**

A. Scaling
B. Rotation
C. Shearing
D. Translation

M10. The point (3, 1) is reflected with Fy = [1 0; 0 −1]. Where does it go? **CHOOSE 1**

A. (−3, 1)
B. (3, −1)
C. (1, 3)
D. (−3, −1)

M11. What do homogeneous coordinates add to a 2-D point? **CHOOSE 1**

A. A third coordinate equal to 1
B. A colour channel
C. A second copy of x
D. A third coordinate equal to 0

M12. Which are true of nearest-neighbour interpolation? **CHOOSE ALL THAT APPLY**

A. Computationally inexpensive
B. Preserves the original pixel intensity values
C. May introduce jagged edges and aliasing
D. Creates new in-between intensity values

M13. Scaling a 2 × 2 image by 2 with forward mapping fills how many of the 16 output pixels? **CHOOSE 1**

A. 16
B. 8
C. 4
D. 2

M14. A point is exactly in the middle of four pixels 10, 20, 30, 40. What does bilinear interpolation give? **CHOOSE 1**

A. 10
B. 20
C. 25
D. 40

Answer key

1. **M1: B**. 200 + 100 = 300 is over the 8-bit maximum, so it saturates at 255. 44 would be wrap-around (300 − 256), which the slide's min() avoids. **Topic 3 · slide 4**
2. **M2: B**. max(0, A − B) clips negative differences to 0 and hides them; the absolute difference keeps changes in both directions. **Topic 3 · slide 5**
3. **M3: A, B, C**. The first three are on slide 6. Adding a constant offset is an application of addition (slide 4). **Topic 3 · slides 4, 6**
4. **M4: C**. A × 1 keeps the pixel and A × 0 removes it, which is element-wise multiplication. **Topic 3 · slides 8–9**
5. **M5: B**. ε is a tiny constant so that a zero pixel in B never causes a divide by zero. **Topic 3 · slide 10**
6. **M6: A**. Averaging N images of the same still scene keeps the signal and averages out random noise (astronomy, medical imaging). Moving content would blur. **Topic 3 · slides 12–13**
7. **M7: B**. They are Boolean operations, so pixel values represent logical states such as foreground/background. **Topic 3 · slide 14**
8. **M8: A, B, D**. Losing the intensity range and needing a threshold are the logical side of the slide 16 table; overflow, clipping loss and needing normalisation are the arithmetic side. **Topic 3 · slide 16**
9. **M9: A, B, C**. Scaling, rotation, shearing (and reflection) are 2 × 2 linear transforms. Translation adds a constant, so it needs homogeneous coordinates. **Topic 3 · slides 19–23**
10. **M10: B**. Fy keeps x and negates y: (3, −1). (−3, 1) would be Fx; (1, 3) is the 45° reflection that swaps x and y. **Topic 3 · slide 22**
11. **M11: A**. (x, y) becomes (x, y, 1), so translation becomes a 3 × 3 matrix multiply and transforms can be combined by multiplying matrices. **Topic 3 · slides 23–24**
12. **M12: A, B, C**. NN copies the closest source pixel: cheap and value-preserving, but blocky. Creating in-between values is what bilinear does. **Topic 3 · slides 29–30**
13. **M13: C**. Each of the 4 source pixels lands on one output pixel; the other 12 are holes. Inverse mapping fixes this. **Topic 3 · slides 26–27**
14. **M14: C**. a = b = 0.5, so each weight is 0.25: (10 + 20 + 30 + 40)/4 = 25. Nearest neighbour would copy one of the four instead. **Topic 3 · slide 30**

L04 Image noise & filtering

Where image noise comes from and how each kind is modelled: Gaussian, Poisson (shot), salt-and-pepper and speckle. Then the spatial filters that reduce it (mean, Gaussian, median), the difference between convolution and cross-correlation, and what to do at the image border. slides

Short Note — Quick Revision

- **Noise model:** g(x,y) = f(x,y) + η(x,y). The goal is to estimate f from g **without losing detail**. **The noise model decides the filter.**
- **Gaussian** (sensor) η ~ N(μ, σ²) · **Poisson** (shot) K ~ Poisson(λ), mean = var = λ · **salt & pepper** (transmission/storage, impulse) · **speckle** (ultrasound/radar, **multiplicative** g = f·η).
- Noise causes false edges, broken segmentation, hidden features and lost small structures.
- **Spatial** (on pixels, linear or non-linear) vs **frequency** domain. **Mean** (1/9)[1…1] blurs edges · **Gaussian** (1/16)[1 2 1; 2 4 2; 1 2 1] · **median** for salt & pepper, keeps edges, non-linear.
- **Which filter:** Gaussian noise → Gaussian/mean · salt & pepper → median · Poisson → Gaussian/mean, then Wiener · speckle → Lee/Frost.
- **Convolution** = cross-correlation with the kernel **flipped**. They are the same for symmetric kernels. CNN "convolution" (PyTorch) is really cross-correlation. **Borders:** zero · replicate · reflect · valid · truncated.

Pick the filter from the noise model

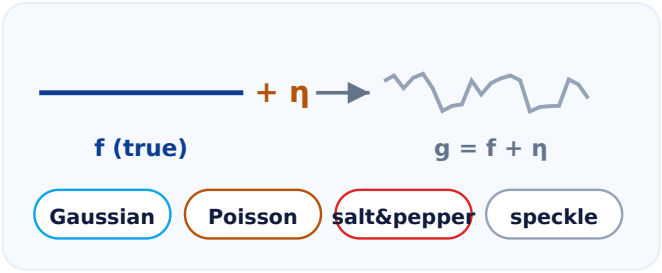
Noise type → cause → model → filter to start with

noise	where it comes from	model	start with
Gaussian (sensor)	circuitry + heat · high ISO · dark areas	g = f + η, η ~ N(μ, σ²)	Gaussian / mean filter
Poisson (shot)	photons arrive discretely · low light	K ~ Poisson(λ), mean = var = λ	Gaussian/mean → Wiener
Salt & pepper	transmission/storage errors · dead pixels	impulse: random 0 or 255	Median filter
Speckle	coherent imaging: ultrasound, radar	g = f · η (multiplicative)	Lee / Frost filters

The lecture's main point: **the noise model matters**. A filter that works well for one type can do badly on another.

Discussion (following the lecture)

What is image noise?



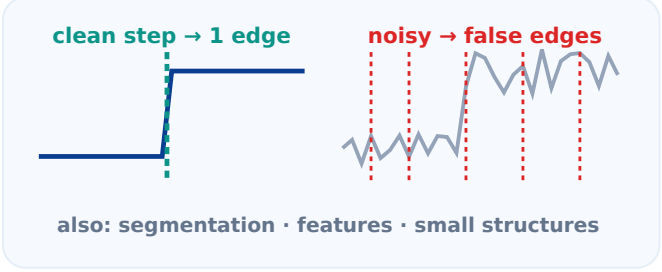
The recorded pixel values can differ from those that would ideally represent the scene, because of **sensor limitations, photon statistics, transmission errors and the imaging process**. The additive model is **g(x,y) = f(x,y) + η(x,y)**: observed = true scene + noise. The **filtering problem** is to estimate f from g *without removing meaningful image detail*.

The four noise sources

NOISE	CAUSE	MODEL	SEEN IN
Sensor (Gaussian)	random variations from the electronics and heat; worse in low light	additive, η ~ N(μ, σ²). To simulate: draw one sample per pixel and add the noise matrix to the image	high-ISO photos · dark regions · scientific imaging
Photon / shot (Poisson)	light arrives as discrete photons, so the count fluctuates	K ~ Poisson(λ), **mean = variance = λ**. λ = 10 gives σ = 3.16; λ = 100 gives σ = 10	low-light photography · fluorescence microscopy · astronomy
Transmission / storage (impulse)	corruption in transmission or storage, defective pixels, processing faults, deliberate damage	**salt & pepper**: isolated extreme values	corrupted files · broken sensor pixels
Imaging process (speckle)	coherent imaging gives a granular pattern	**multiplicative** g = f · η: the change depends on the pixel value	ultrasound · synthetic aperture radar

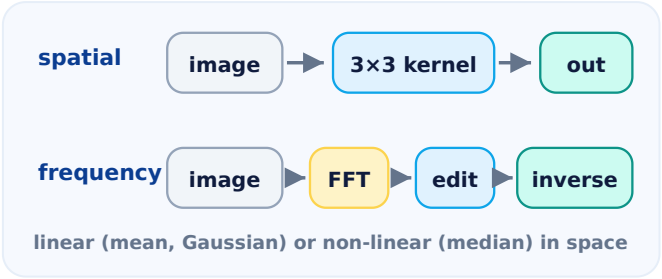
Why shot noise shows more in dark images: σ = √λ grows with the signal, but the *relative* noise σ/λ = 1/√λ shrinks. At λ = 10 the noise is about 32% of the signal; at λ = 100 it is only 10%. So low-light images look noisier even though the absolute σ is smaller.

How noise hurts computer vision



Edge detection: random intensity changes show up as false edges. **Segmentation**: pixels from the same object can look like they belong to different regions. **Object recognition**: important features get hidden or distorted. **Medical image analysis**: small structures can be lost when the image is denoised.

Noise filtering: spatial vs frequency domain



Noise filtering reduces unwanted random variation *while keeping edges, textures and fine detail*. **Spatial-domain** filters work directly on pixels and use each pixel's neighbourhood to compute the output. They are either **linear** (mean, Gaussian) or **non-linear** (median). **Frequency-domain** filters transform the image into frequencies, change some of them, and transform back. A **spatial filter** computes a new value from a local neighbourhood (3 × 3 = 9 pixels around the centre), for example its average, median or weighted average.

One salt pixel in a flat region: mean vs median

10	10	10
10	255	10
10	10	10

3 × 3 neighbourhood

Mean: (8·10 + 255)/9
= 37 → a grey smudge

Gaussian [1 2 1; 2 4 2; 1 2 1]/16
(12·10 + 4·255)/16 = 71

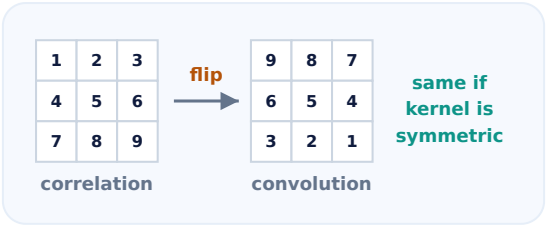
Median: sort, take the 5th
10 10 10 10 [10] 10 10 10 255 → 10

Median removes the impulse
and keeps edges sharp

The same neighbourhood under three filters. Linear filters spread the outlier into its neighbours; the median simply ignores it.

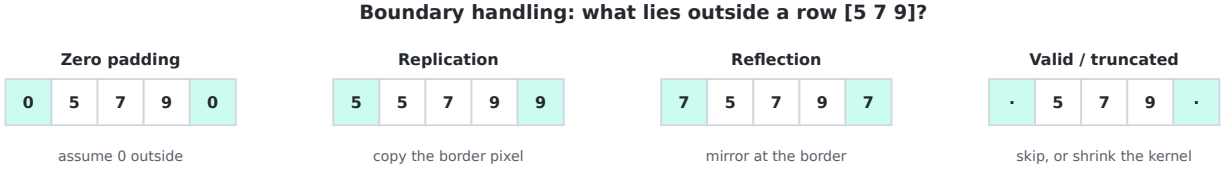
FILTER	KERNEL / RULE	GOOD AT	WEAKNESS
Mean (n × n)	$g'(x,y) = (1/n^2)\ \Sigma\ g(x+i, y+j)$; 3 × 3 = (1/9)·[1 1 1; 1 1 1; 1 1 1]	Gaussian noise, simple	pulled by outliers · **blurs edges** (averages across both sides)
Gaussian	weights from $G(x,y) = (1/2\pi\sigma^2)\cdot e^{-(x^2+y^2)/2\sigma^2}$; 3 × 3 ≈ (1/16)·[1 2 1; 2 4 2; 1 2 1]	Gaussian noise; centre pixels count more	does not reliably remove isolated extreme pixels
Median	sort the neighbourhood, take the middle value	**salt & pepper**; keeps edges better than linear filters	**non-linear**: cannot be written as a fixed convolution kernel

Cross-correlation vs convolution



Cross-correlation multiplies the neighbourhood by the kernel *as it is*: $(f \star h)[m,n] = \Sigma_i\Sigma_j\, f[m+i, n+j]\cdot h[i,j]$. **Convolution** does the same multiply-and-sum but **flips the kernel horizontally and vertically** first: $(f * h)[m,n] = \Sigma_i\Sigma_j\, f[m-i, n-j]\cdot h[i,j]$. With a **symmetric** kernel (mean, Gaussian) the results are identical. They differ only for **asymmetric** kernels, such as derivative filters, where the sign changes. Note that what CV and deep-learning libraries call "convolution" is often really cross-correlation; standard CNN layers in PyTorch do not flip the learned kernel.

Boundary handling



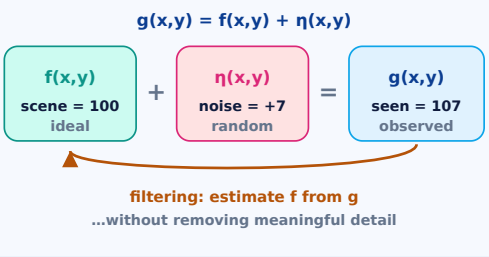
An n × n kernel does not fit when its centre is near the edge. The five strategies from the slides.

Step by step — deeper notes

Everything here follows the Topic 4 slides; each part shows its slide numbers. Parts marked **added** are study aids we wrote, not lecture content.

1 · The noise model and the filtering problem

Topic 4 · slides 3–5, 20–21



An image is a digital record of a scene, and the recorded values can differ from the ideal ones because of **sensor limitations**, **photon statistics**, **transmission errors** and **imaging process noise** (slide 3).

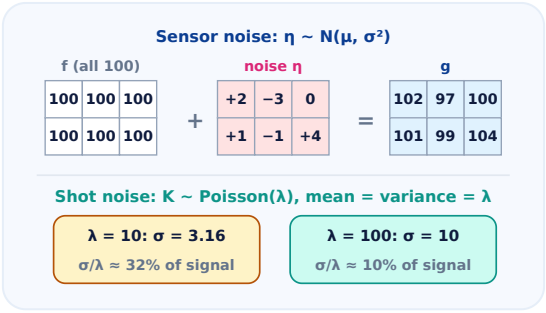
Model (slide 4): the observed pixel is the original scene value plus the noise at that position, **g(x,y) = f(x,y) + η(x,y)**. Worked: f = 100 and η = +7 give g = **107**. We only ever see 107; the **filtering problem** is to estimate f from g **without removing meaningful image details**.

The **noise model matters**: a filter that works well for one type may do poorly on another (slide 5). Why bother? Noise gives **false edges** in edge detection, splits one object into different regions in **segmentation**, hides or distorts features in **recognition**, and small structures can be lost when **medical images** are denoised (slides 20–21).

The values 100 and +7 are our example.

2 · Two random models: Gaussian (sensor) and Poisson (shot)

Topic 4 · slides 6–13



Sensor noise: the sensor's electronics and thermal effects add random variations, most visible in **low-light** photos (slide 6): high-ISO shots, dark regions, scientific imaging. Slide 7 shows it growing from ISO 100 to ISO 25600. It is modelled as additive **Gaussian** noise, **η ~ N(μ, σ²)** (slide 8).

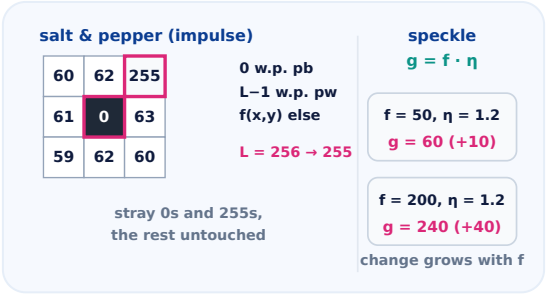
Adding Gaussian noise, step by step (slide 9): 1. choose the distribution (mean 0). 2. draw independent random samples, **one per pixel**. 3. **add** the noise matrix to the image. Worked on a 2 × 3 patch of 100s with noise +2, −3, 0, +1, −1, +4 → g = 102, 97, 100, 101, 99, 104.

Shot noise: light arrives as discrete **photons**, so the count fluctuates, worse in low light (slide 10). **K ~ Poisson(λ)**: K = photons detected, λ = expected count, and **mean = variance = λ** (slide 11). Slide 12: λ = 10 → σ = √10 = **3.16**; λ = 100 → σ = **10**. σ is bigger at 100, but relative to the signal it is smaller (3.16/10 ≈ 32% vs 10/100 = 10%), so dim images show shot noise more. Seen in low-light photography, fluorescence microscopy, astronomy (slide 13).

The noise values are our example. Slide 9 literally says "mean and standard deviation of zero"; a σ of 0 would add no noise, so read it as mean 0.

3 · Extreme pixels and multiplied noise: salt & pepper, speckle

Topic 4 · slides 14–19



Transmission and storage errors (transmission errors, defective sensor pixels, processing or storage faults, deliberate corruption) produce **extreme or irregular** values (slide 14). They are the primary cause of **salt and pepper** noise, also called **impulse** noise (slide 15):

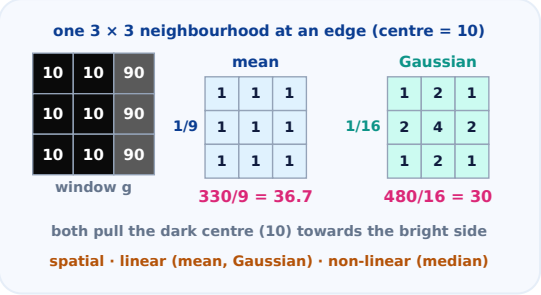
g(x,y) = **0** with probability p_b (pepper), **L − 1** with probability p_w (salt), and **f(x,y)** otherwise. For an 8-bit image L − 1 = 255. So a few pixels jump to black or white and the rest stay exact (slide 16).

Speckle (slides 17–19): coherent systems such as **ultrasound** and **synthetic aperture radar** produce a granular pattern. The noise **multiplies**: **g(x,y) = f(x,y) · η(x,y)**, so the change depends on the pixel value. Worked with η = 1.2: f = 50 → 60 (+10), f = 200 → 240 (+40). Same η, four times the change on the bright pixel.

The window values and η = 1.2 are our examples.

4 · Spatial filters: mean vs Gaussian on one neighbourhood

Topic 4 · slides 22–30



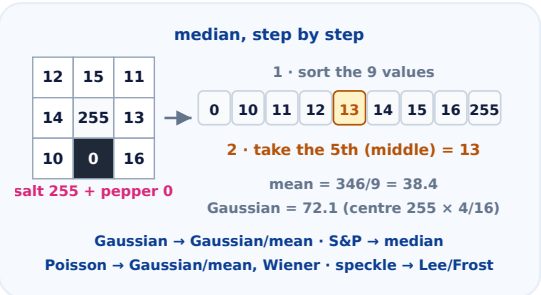
Noise filtering reduces unwanted random variation while keeping edges, textures and fine detail (slide 22). Two families: **spatial domain** (works directly on pixels using neighbours; **linear** or **non-linear**) and **frequency domain** (transform, change the frequency components, transform back) (slide 23). A spatial filter computes a new value from the local neighbourhood: for 3 × 3, the 9 pixels around the centre (slide 24).

Mean (slides 25–26): $g'(x,y) = (1/n^2)\ \Sigma_i\ \Sigma_j\ g(x+i, y+j)$, so for 3 × 3 the kernel is (1/9) × all ones. Worked on the window above: six 10s and three 90s → (60 + 270)/9 = **36.7**. **Gaussian** (slides 28–29): weights from $G(x,y) = (1/2\pi\sigma^2)\ e^{-(x^2+y^2)/2\sigma^2}$, 3 × 3 ≈ (1/16)[1 2 1; 2 4 2; 1 2 1]. Worked: column weights are 4, 8, 4 → (4×10 + 8×10 + 4×90)/16 = 480/16 = **30**.

Both are weighted sums, so both pull the dark centre towards the bright side: that is the edge **blurring** on slides 26–27. The Gaussian moves it less because the centre counts most (4/16 against 1/9). The mean is also pulled hard by **outliers** (slide 26), and the Gaussian does not reliably remove isolated extreme pixels (slide 30).

5 · The median filter, and which filter for which noise

Topic 4 · slides 31–34



The **median filter** sorts the values in the neighbourhood and selects the **middle** one (slide 31). It is particularly effective against **salt and pepper** noise, preserves edges better than linear smoothing filters, and is **non-linear**, so it cannot generally be written as a fixed convolution kernel.

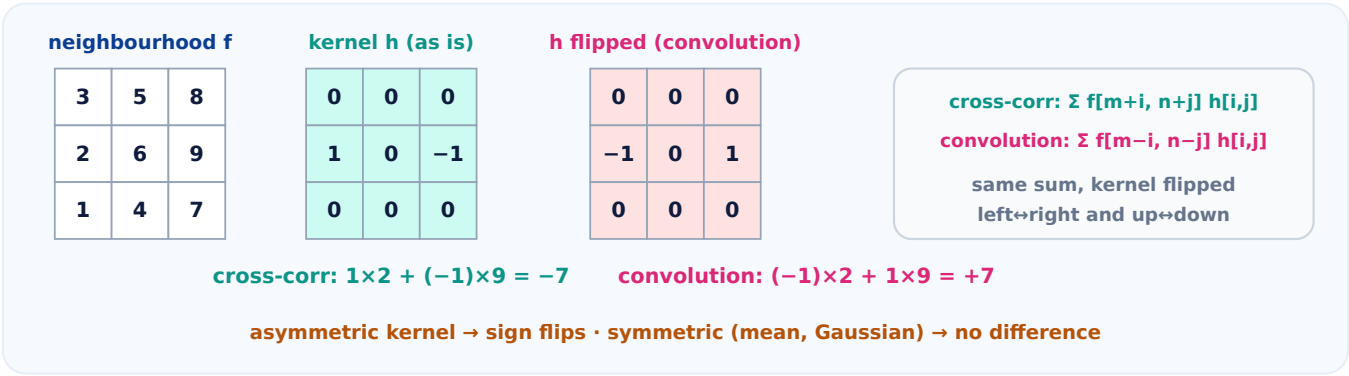
Worked on a window hit by one salt (255) and one pepper (0): 1. list the 9 values. 2. sort: 0, 10, 11, 12, 13, 14, 15, 16, 255. 3. take the 5th: **13**. Compare the linear filters on the same window: mean 346/9 = **38.4**, Gaussian 1153/16 = **72.1**. The extremes sit at the ends of the sorted list and never reach the middle, which is why slide 32's speckled brain scan comes out clean.

Which filter? (slides 33–34) Gaussian noise → start with **Gaussian or mean**, move to advanced filters if fine detail must be kept. Salt & pepper → **median**. Poisson → Gaussian or mean, then specialised ones such as **Wiener**. Speckle → speckle-specific methods such as **Lee or Frost**.

The window values are our example.

6 · Cross-correlation vs convolution, worked

Topic 4 · slides 35–37



Cross-correlation multiplies the neighbourhood by the kernel in its **original orientation**: $(f * h)[m,n] = \Sigma_i \Sigma_j f[m+i, n+j]\, h[i,j]$ (slide 35). **Convolution** uses the same multiply-and-sum, but the kernel is **flipped horizontally and vertically** first: $\Sigma_i \Sigma_j f[m-i, n-j]\, h[i,j]$ (slide 36).

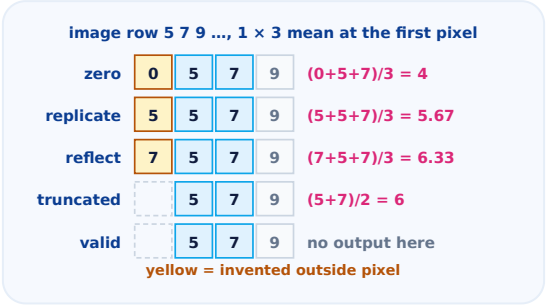
Worked with $i, j \in \{-1, 0, 1\}$ and the asymmetric kernel $h[0,-1] = 1$, $h[0,1] = -1$ (all others 0). Cross-correlation: $f[m, n-1]\cdot 1 + f[m, n+1]\cdot(-1) = 2 - 9 = \mathbf{-7}$. Convolution: $f[m, n+1]\cdot 1 + f[m, n-1]\cdot(-1) = 9 - 2 = \mathbf{+7}$. Same size, opposite **sign**, exactly the "sign difference" of slide 37. With a symmetric kernel such as the mean or Gaussian, flipping changes nothing.

Slide 37 also warns that many CV libraries and deep-learning frameworks implement "convolution" that is really cross-correlation: standard CNN layers in PyTorch don't flip the learned kernel.

The 3 × 3 numbers and kernel are our example. Slide 35 writes cross-correlation with the same $*$ symbol as convolution.

7 · Boundary handling: five answers at one edge pixel

Topic 4 · slide 38



An **n × n kernel cannot fit** inside the image when its centre is near an edge (slide 38). The five strategies: **zero padding** (pixels outside are 0), **replication padding** (extend the nearest border pixel outward), **reflection padding** (reflect the image at its boundary), **valid filtering** (only output where the whole kernel fits) and **truncated kernel** (use only the pixels caught inside the image).

Worked on a row starting 5, 7, 9 with a 3-wide mean centred on the first pixel (5): zero → (0 + 5 + 7)/3 = **4**, the border darkens. Replicate → (5 + 5 + 7)/3 = **5.67**. Reflect → (7 + 5 + 7)/3 = **6.33**. Truncated → (5 + 7)/2 = **6**. Valid → **no value** at this pixel, so the output is smaller than the input.

added The 1 × 3 row keeps the arithmetic short; a 3 × 3 kernel works the same way in both directions. "Reflect" here mirrors about the edge pixel; some tools repeat the edge pixel instead.

8 · Exam traps

Topic 4 · slides 4, 11–12, 15, 17, 26, 30–31, 36–38 added

- **Additive vs multiplicative:** Gaussian sensor noise is g = f + η (slide 4); speckle is g = f · η (slide 17). Only speckle's change depends on the pixel value.
- **Poisson:** mean = variance = λ, so σ = √λ, not λ (slide 11). λ = 100 gives σ = 10.
- **Salt & pepper = impulse** noise, caused mainly by transmission/storage errors (slide 15), not by the sensor's heat.
- **Gaussian filter ≠ cure for salt & pepper:** it does not reliably remove isolated extremes (slide 30). Use the median (slide 31).
- **The median is non-linear:** no fixed kernel exists for it (slide 31). Mean and Gaussian are linear.
- **Convolution flips** the kernel both ways (slide 36). If the kernel is symmetric, convolution and cross-correlation are equal (slide 37).
- **Valid filtering shrinks** the output; the padding methods keep its size (slide 38).

9 · Quick check

Topic 4 · slides 11–12, 26, 31 added

▼ **Q1.** λ = 64 photons on average. What are the mean, variance and standard deviation of the count?

Mean = variance = 64, so σ = √64 = 8 (slides 11–12). Relative noise 8/64 = 12.5%.

▼ **Q2.** A 3 × 3 window holds eight 20s and one 200. Give the 3 × 3 mean and the median.

Mean = (8 × 20 + 200)/9 = 360/9 = 40. Median: sorted, the 5th value is 20 (slides 26, 31). The median ignores the outlier.

▼ **Q3.** Why can't the median filter be written as a kernel H like the mean's (1/9)[1…1]?

Because it sorts and picks the middle value rather than taking a weighted sum; it is non-linear (slide 31).

Key terms

g = f + η · sensor noise · Gaussian N(μ, σ²) · photon / shot noise · Poisson(λ) · salt & pepper / impulse · speckle (multiplicative) · spatial vs frequency domain · linear vs non-linear filter · mean filter · Gaussian filter · median filter · Wiener filter · Lee / Frost · kernel · cross-correlation · convolution (flipped kernel) · zero / replication / reflection padding · valid filtering · truncated kernel

Exam relevance

Try each question yourself first, then reveal the answer.

Q1. "What is image noise? Give the additive noise model and state the filtering problem."

Answer: Noise is the difference between the recorded pixel values and the values that would ideally represent the scene. It comes from sensor limitations, photon statistics, transmission errors and the imaging process. Additive model: **g(x,y) = f(x,y) + η(x,y)**, observed = original + noise. The **filtering problem** is to estimate f(x,y) from g(x,y) **without removing meaningful image detail** (edges, textures). The noise model matters because a filter that suits one type can fail on another.

Q2. "Describe the four main sources of image noise with their models and where each is seen."

Answer: **Sensor (Gaussian):** electronics and heat; additive $\eta \sim N(\mu, \sigma^2)$; high-ISO and low-light photos, scientific imaging. **Photon/shot (Poisson):** light arrives as discrete photons; $K \sim \text{Poisson}(\lambda)$ with mean = variance = λ; low light, fluorescence microscopy, astronomy. **Transmission/storage (salt & pepper, impulse):** corruption, defective pixels, processing faults; isolated extreme values. **Imaging process (speckle):** coherent systems such as ultrasound and SAR; **multiplicative** $g = f \cdot \eta$, so the change depends on the pixel value.

Q3. "A pixel receives on average 10 photons per exposure. What are the mean and standard deviation of the count? Why does shot noise show more in dark images?"

Answer: For Poisson the mean = variance = $\lambda = 10$, so $\sigma = \sqrt{10} \approx$ **3.16**. At $\lambda = 100$, $\sigma = 10$. The absolute σ grows with the signal, but the **relative** noise $\sigma/\lambda = 1/\sqrt{\lambda}$ shrinks: 32% at $\lambda = 10$ versus 10% at $\lambda = 100$. So images taken in **low light** show far more noticeable shot noise.

Q4. "Compare the mean, Gaussian and median filters. Which would you use for salt-and-pepper noise, and why?"

Answer: **Mean**: the average of the n × n window (3 × 3 kernel = 1/9 of all ones). Simple, but outliers pull it and it **blurs edges** because it averages across both sides of a boundary. **Gaussian**: a weighted average with larger weights near the centre ($3 \times 3 \approx 1/16 \cdot [1\ 2\ 1;\ 2\ 4\ 2;\ 1\ 2\ 1]$). Good for Gaussian noise but it does not reliably remove isolated extreme pixels. **Median**: sort the window and take the middle value. It is **non-linear** (no fixed kernel), very robust to isolated extremes and **keeps edges** better. For salt & pepper use the **median**. Example: a 3 × 3 window of eight 10s and one 255 gives mean ≈ 37 but median = 10.

Q5. "Distinguish convolution from cross-correlation. When do they give the same result?"

Answer: Both multiply the neighbourhood by a kernel and sum. **Cross-correlation** uses the kernel as it is: $\Sigma\, f[m+i, n+j] \cdot h[i,j]$. **Convolution** flips the kernel horizontally and vertically first: $\Sigma\, f[m-i, n-j] \cdot h[i,j]$. They are **identical for symmetric kernels** (mean, Gaussian) and differ in sign for asymmetric ones (derivatives). Many CV and deep-learning libraries call cross-correlation "convolution"; standard PyTorch CNN layers don't flip the learned kernel.

Q6. "An n × n kernel does not fit at the image border. List the boundary-handling strategies."

Answer: **Zero padding**: assume outside pixels are 0 (can darken borders). **Replication padding**: extend the nearest border pixel outward. **Reflection padding**: mirror the image at its boundaries. **Valid filtering**: compute outputs only where the whole kernel fits (smaller output). **Truncated kernel**: at the border, use only the kernel pixels that fall inside the image.

Practice: 10 questions

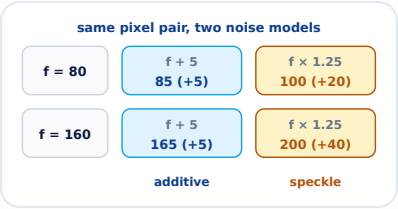
Exam-style questions on Topic 4: classify, calculate, apply, match, explain, describe, compare. Answer on paper first, then reveal. Each answer cites its slides; **added** marks anything that goes beyond the slides. "Show all answers" above opens these too.

P1. **CLASSIFY** Which noise model fits each case? (a) a high-ISO night photo (b) an astronomy image with very few photons (c) an image sent over a faulty link, with random black and white dots (d) the granular texture of an ultrasound scan (e) a defective sensor pixel.



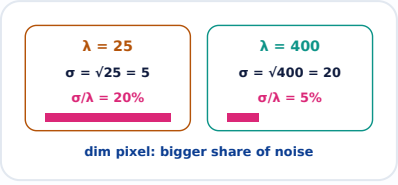
Answer · Topic 4 slides 6, 10–11, 13–15, 17–19**:** (a) **Gaussian** sensor noise: high ISO, low light (slide 6). (b) **Poisson** shot noise: photon-limited imaging, astronomy (slides 11, 13). (c) **Salt & pepper**: transmission errors cause impulse noise (slides 14–15). (d) **Speckle**: coherent imaging such as ultrasound (slides 17–19). (e) **Salt & pepper**: defective sensor pixels are listed with transmission and storage errors (slide 14).

P2. **CALCULATE** Two pixels have f = 80 and f = 160. Find g under additive noise with η = 5, and under speckle noise with η = 1.25. What does the result show?



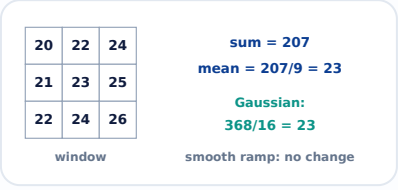
Answer · Topic 4 slides 4, 17**:** Additive, g = f + η: 80 → **85**, 160 → **165** (both +5). Speckle, g = f · η: 80 → **100** (+20), 160 → **200** (+40). With additive noise the change is the same everywhere; with speckle it **depends on the original pixel value** (slide 17), so bright areas change more.

P3. **CALCULATE** One pixel expects λ = 25 photons, another λ = 400. Give the standard deviation of each count and say which will look noisier.



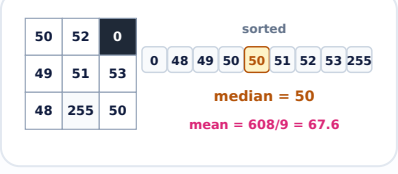
Answer · Topic 4 slides 11–12**:** Poisson: mean = variance = λ, so σ = √λ. λ = 25 → σ = **5**; λ = 400 → σ = **20**. The bright pixel has the larger σ, but relative to its signal it is smaller: 5/25 = **20%** vs 20/400 = **5%**. So the dim (λ = 25) pixel looks noisier, which is why shot noise is most noticeable in low light (slides 10, 12).

P4. **APPLY** Apply the 3 × 3 mean filter and the 3 × 3 Gaussian approximation to the window (20, 22, 24 / 21, 23, 25 / 22, 24, 26). What is the new centre value?



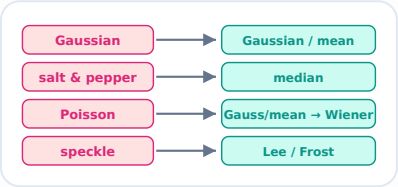
Answer · Topic 4 slides 25–26, 29**:** **Mean**: (1/9) × sum = 207/9 = **23**. **Gaussian**: (1/16)[1 2 1; 2 4 2; 1 2 1]: corners 20 + 24 + 22 + 26 = 92; edges 2 × (22 + 21 + 25 + 24) = 184; centre 4 × 23 = 92; total 368, /16 = **23**. Both give 23, the original centre: on a smooth, evenly sloping patch, smoothing changes nothing. The blurring only shows where there are edges or noise.

P5. **CALCULATE** A 3 × 3 window is (50, 52, 0 / 49, 51, 53 / 48, 255, 50). Give the median-filter output and the mean-filter output for its centre. Which is right for this noise?



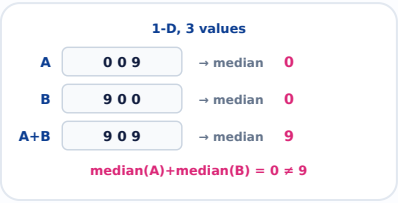
Answer · Topic 4 slides 26, 31, 33**:** Sort: 0, 48, 49, 50, 50, 51, 52, 53, 255 → the 5th value, **median = 50**. Mean = 608/9 = **67.6**, dragged up by the 255. The 0 and 255 are **salt and pepper**, isolated extremes, so the **median** is the right choice (slide 33); the mean is strongly affected by outliers (slide 26).

P6. **MATCH** Match each noise type to the filter the slides recommend: Gaussian, salt & pepper, Poisson, speckle.



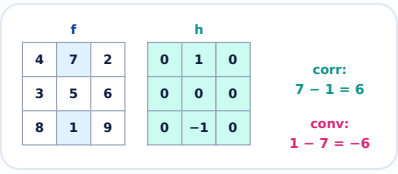
Answer · Topic 4 slides 33–34**:** **Gaussian** → start with a Gaussian or mean filter, then advanced filters if fine detail must be preserved. **Salt & pepper → median**, robust to isolated extreme values. **Poisson** → Gaussian or mean first, then specialised filtering such as the **Wiener** filter. **Speckle** → speckle-specific methods such as **Lee or Frost**, depending on the image model and task.

P7. **EXPLAIN** Explain why the median filter is called non-linear and cannot be written as a fixed convolution kernel. Use a small example.



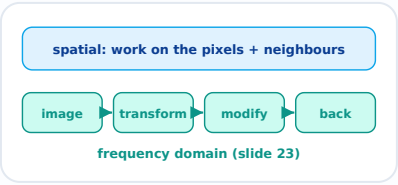
Answer · Topic 4 slides 23, 31**:** A linear filter (mean, Gaussian) is a fixed weighted sum, so filtering A + B equals filtering A plus filtering B. The median **sorts and picks the middle value** (slide 31), which breaks this. **added** Example with three values: median(0, 0, 9) = 0 and median(9, 0, 0) = 0, but median(9, 0, 9) = **9 ≠ 0 + 0**. Since no weights could do this, there is no fixed kernel. That is why spatial filters are split into linear and non-linear (slide 23).

P8. **CALCULATE** Neighbourhood f = (4, 7, 2 / 3, 5, 6 / 8, 1, 9) and kernel h with h[−1,0] = 1, h[1,0] = −1, all others 0. Compute the centre output by cross-correlation and by convolution.



Answer · Topic 4 slides 35–37**:** **Cross-correlation**, Σ f[m+i, n+j] h[i,j]: f[m−1, n]·1 + f[m+1, n]·(−1) = 7 − 1 = **6**. **Convolution**, Σ f[m−i, n−j] h[i,j]: f[m+1, n]·1 + f[m−1, n]·(−1) = 1 − 7 = **−6**. The kernel is asymmetric, so flipping it changes the sign (slide 37). With a symmetric kernel both would agree.

P9. **DESCRIBE** Describe the two main types of noise filtering and how each produces its output.



Answer · Topic 4 slides 22–24**:** **Spatial domain filtering** works directly on image pixels: each output pixel is computed from neighbouring pixel values (for 3 × 3, the 9 pixels around the centre), using rules such as the average, median or weighted average. It splits into **linear** (mean, Gaussian) and **non-linear** (median) filters. **Frequency domain filtering** transforms the image into its frequency representation, modifies frequency components, and transforms back to the spatial domain. Both aim to reduce random variation while keeping edges, textures and fine detail (slide 22).

P10. **COMPARE** A 6 × 6 image is filtered with a 3 × 3 kernel. Compare the output size under padding (zero, replication, reflection) and under valid filtering, and say what a truncated kernel does instead.



Answer · Topic 4 slide 38**:** **Padding** invents the missing outside pixels (zeros, copies of the border, or a mirror), so every pixel gets an output: **6 × 6**. **Valid filtering** only outputs where the whole kernel fits; a 3 × 3 kernel needs one pixel on every side, so the outer ring is lost: **4 × 4**. A **truncated kernel** keeps 6 × 6 without inventing pixels: at the border it uses only the pixels the kernel catches inside the image. **added** In general the valid size is (6 − 3 + 1) = 4 per side.

Quick MCQs: check your understanding

M1. In g(x,y) = f(x,y) + η(x,y), what is η(x,y)? **CHOOSE 1**

A. The observed pixel
B. The original scene value
C. The noise at that pixel position
D. The filter kernel

M2. The image filtering problem is to... **CHOOSE 1**

A. estimate f(x,y) from g(x,y) without removing meaningful details
B. remove every edge from g
C. make g as bright as possible
D. compute η exactly for each pixel

M3. Where is sensor (Gaussian) noise usually encountered? **CHOOSE ALL THAT APPLY**

A. High-ISO photographs
B. Dark image regions
C. Scientific imaging systems
D. Corrupted file transfers

M4. For shot noise with $\lambda = 100$, the standard deviation is... **CHOOSE 1**

A. 100
B. 10
C. 50
D. 3.16

M5. Salt and pepper noise is mainly caused by... **CHOOSE 1**

A. thermal effects in the sensor
B. transmission and storage errors
C. coherent imaging such as ultrasound
D. photon statistics

M6. Speckle noise is modelled as... **CHOOSE 1**

A. $g = f + \eta$
B. $g = f \cdot \eta$
C. $K \sim \text{Poisson}(\lambda)$
D. $g = 0$ or $L - 1$

M7. Which describe spatial domain filtering? **CHOOSE ALL THAT APPLY**

A. Operates directly on image pixels
B. Uses neighbouring pixel values to compute the output
C. Can be linear or non-linear
D. Transforms the image into its frequency representation

M8. Each weight in the 3 × 3 mean kernel is... **CHOOSE 1**

A. 1
B. 1/3
C. 1/9
D. 1/16

M9. In the 3 × 3 Gaussian approximation, the centre pixel's weight is... **CHOOSE 1**

A. 1/16
B. 2/16
C. 4/16
D. 1/9

M10. Which are true of the median filter? **CHOOSE ALL THAT APPLY**

A. Particularly effective against salt & pepper noise
B. Preserves edges better than linear smoothing filters
C. It is non-linear
D. It can be written as a fixed 3 × 3 convolution kernel

M11. A 3 × 3 window holds eight 30s and one 0. The median output is... **CHOOSE 1**

A. 30
B. 26.7
C. 15
D. 0

M12. For speckle noise the slides suggest... **CHOOSE 1**

A. a median filter
B. Lee or Frost filters
C. only valid filtering
D. a Wiener filter first

M13. Convolution differs from cross-correlation because convolution... **CHOOSE 1**

A. adds instead of multiplies
B. flips the kernel horizontally and vertically first
C. only works on colour images
D. uses the median of the window

M14. When do convolution and cross-correlation give the same result? **CHOOSE ALL THAT APPLY**

A. With the 3 × 3 mean kernel
B. With the 3 × 3 Gaussian kernel
C. With any symmetric kernel
D. With an asymmetric kernel

Answer key

1. **M1: C**. g is observed, f is the original scene value and η is the noise at that position. **Topic 4 · slide 4**
2. **M2: A**. Estimate the original f from the observed g while keeping meaningful image detail. **Topic 4 · slide 4**
3. **M3: A, B, C**. Slide 6 lists high-ISO photos, dark regions and scientific imaging. File corruption gives salt & pepper noise. **Topic 4 · slide 6**
4. **M4: B**. Poisson: mean = variance = λ = 100, so σ = √100 = 10. 3.16 is the λ = 10 case. **Topic 4 · slides 11–12**
5. **M5: B**. Transmission and storage errors are the primary cause of salt & pepper (impulse) noise. **Topic 4 · slides 14–15**
6. **M6: B**. Speckle is multiplicative, g = f · η, so the change depends on the pixel value. **Topic 4 · slide 17**
7. **M7: A, B, C**. The frequency transform is frequency-domain filtering. **Topic 4 · slides 22–23**
8. **M8: C**. H = (1/9) × a 3 × 3 matrix of ones, so each of the 9 pixels counts 1/9. **Topic 4 · slide 26**
9. **M9: C**. H = (1/16)[1 2 1; 2 4 2; 1 2 1]: the centre gets 4/16, the largest weight. **Topic 4 · slide 29**
10. **M10: A, B, C**. Because it is non-linear (sort and pick the middle), it cannot generally be written as a fixed kernel. **Topic 4 · slide 31**
11. **M11: A**. Sorted, the 5th of the 9 values is 30, so the pepper pixel is ignored. The mean would be 240/9 = 26.7. **Topic 4 · slide 31**
12. **M12: B**. Speckle → speckle-specific methods such as Lee or Frost. Wiener is suggested for Poisson noise. **Topic 4 · slides 33–34**
13. **M13: B**. Same multiply-and-sum; convolution flips the kernel both ways before multiplying. **Topic 4 · slides 35–36**
14. **M14: A, B, C**. Flipping a symmetric kernel leaves it unchanged. An asymmetric kernel gives a sign difference. **Topic 4 · slide 37**

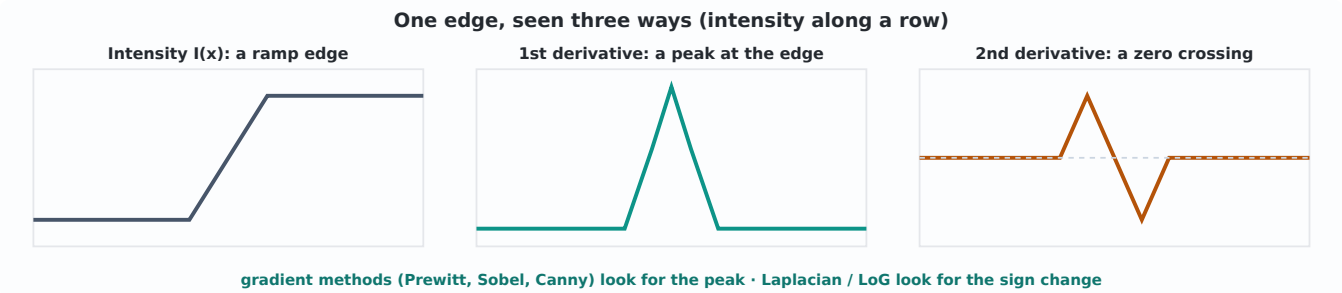
L05 Edge detection

What an edge is and what causes it, the gradient and its magnitude and direction, and how derivatives become filters: Prewitt, Sobel, the Laplacian and LoG. The topic ends with the five-step Canny detector. slides

Short Note — Quick Revision

- **Edge** = a significant local intensity change, usually on a region boundary. **Edge pixels** change abruptly; an **edge** = connected edge pixels. Causes: object/surface boundaries (geometric), and specularity, shadows, inter-reflections (non-geometric).
- **Gradient** $\nabla I = [I_x, I_y]$ · **$G = \sqrt{I_x^2+I_y^2}$** · **$\theta = \tan^{-1}(I_y/I_x)$**. The edge runs ⊥ to the gradient. $I_x = I_y = 50$ gives $G \approx 70.7$, $\theta = 45°$, edge at 135°.
- Differences: forward · backward · **central**. Derivative in one direction + smoothing in the other: **Prewitt** $[-1\ 0\ 1]\times[1\ 1\ 1]^T$ · **Sobel** $\times[1\ 2\ 1]^T$. Threshold T → binary edge map (manual, histogram, percentile, Otsu, adaptive).
- **Laplacian** $\nabla^2 I = I_{xx}+I_{yy}$, 4-nbr $[0\ 1\ 0;\ 1\ -4\ 1;\ 0\ 1\ 0]$. Edge at a **zero crossing**: $L_1L_2 < 0$ and $|L_1-L_2| \geq T$. Noise-sensitive, no orientation.
- **LoG** (Marr–Hildreth) = $\nabla^2(G_\sigma * I)$: smooth → Laplacian → zero crossings → threshold.
- **Canny:** Gaussian → Sobel gradient (M, θ) → **non-maximum suppression** → **double threshold** (T_L, T_H) → **hysteresis** (keep weak only if connected to strong).

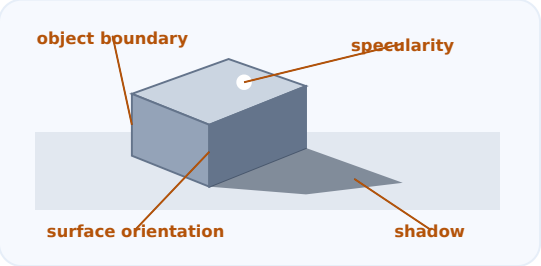
What an edge looks like to a derivative



Edges are **significant local changes in intensity**. The 1st derivative turns an edge into a **peak**; the 2nd derivative turns it into a **zero crossing**. Every method in this topic looks for one of the two.

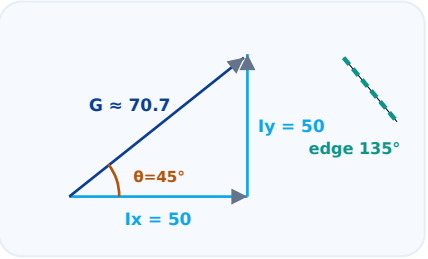
Discussion (following the lecture)

What is an edge, and what causes one?



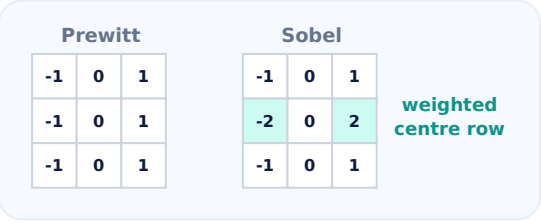
Edges are significant local changes of intensity, usually on the **boundary between two regions**. **Edge pixels** are pixels where the intensity changes abruptly. An **edge** is a set of *connected* edge pixels, and **edge detectors** are methods that find edge pixels. The goal is a **line drawing** of the scene, from which features (corners, lines, curves) are extracted for recognition and segmentation. **Causes:** *geometric* ones are object boundaries (a break in depth or in surface colour/texture) and surface boundaries (a break in surface orientation or colour/texture). *Non-geometric* ones are specularity (highlights), shadows and inter-reflections.

The gradient



Treat the image as a function I(x,y): $\mathbb{R}^2 \to \mathbb{R}$. The rate of intensity change is its first derivative, in both directions: $I_x = \partial I/\partial x$, $I_y = \partial I/\partial y$. Together they form the **gradient vector $\nabla I = [I_x, I_y]$**, with **magnitude $G = \sqrt{(I_x^2 + I_y^2)}$** and **direction $\theta = \tan^{-1}(I_y/I_x)$**. The gradient points in the direction of *fastest increase*, and the edge itself runs **perpendicular** to it. **Slide example**: $I_x = I_y = 50$ gives $G = \sqrt{5000} \approx$ **70.7** and a maximum-increase direction of $\theta =$ **45°**, so the local edge orientation is 45° + 90° = **135°** (equivalently −45°).

From derivatives to filters: Prewitt and Sobel



A digital image only has samples at whole pixel positions, so derivatives are approximated by differences: **forward** I(x+1) − I(x), **backward** I(x) − I(x−1), or **central** [I(x+1) − I(x−1)]/2. Differences amplify noise, so in practice a two-step procedure is used: **estimate the derivative in one direction and smooth in the perpendicular direction**. **Prewitt** uses the mask [−1 0 1] with the smoothing vector $[1\ 1\ 1]^T$. **Sobel** uses the same derivative with a *weighted* smoothing $[1\ 2\ 1]^T$, which suppresses noise a little better. Transpose for the y-direction. The gradient magnitude is then turned into a **binary edge map** with a threshold: `E[i,j] = 1 if G[i,j] ≥ T, else 0`.

Masks (x-direction for gradients; transpose for y)

Prewitt Gx

-1	0	1
-1	0	1
-1	0	1

Sobel Gx

-1	0	1
-2	0	2
-1	0	1

Laplacian (4-nbr)

0	1	0
1	-4	1
0	1	0

Laplacian (8-nbr)

1	1	1
1	-8	1
1	1	1

Prewitt / Sobel = derivative $[-1\ 0\ 1]$ × smoothing $[1\ 1\ 1]^T$ or $[1\ 2\ 1]^T$ · Laplacian = sum of second differences

The masks from the slides. Sobel gives the centre row double weight; that weighted smoothing is its only difference from Prewitt.

CHOOSING T	HOW
Manual	try values and inspect the result
Histogram analysis	look at the distribution of gradient magnitudes; pick a value that separates weak from strong responses
Percentile-based	take a chosen percentile of the gradient magnitudes
Otsu's method	automatically split the response histogram into two groups, when such a split makes sense
Adaptive	let T vary across the image when edge strength or noise varies

Second derivative: the Laplacian

	1	
1	-4	1
	1	

$L_1 \cdot L_2 < 0$ sign flips

$|L_1 - L_2| \geq T$ big enough

→ edge

The 1st derivative measures how fast intensity changes; the 2nd measures how *that rate* changes: $\nabla^2 I = \partial^2 I/\partial x^2 + \partial^2 I/\partial y^2$. With second differences $I_{xx} = I[i, j+1] - 2I[i,j] + I[i, j-1]$ (and similarly I_{yy}) this gives the **4-neighbour** mask [0 1 0; 1 −4 1; 0 1 0]; adding the diagonals gives the **8-neighbour** mask. Edges sit at **zero crossings**. To mark an edge between neighbouring responses L_1 and L_2 we need $L_1 \cdot L_2 < 0$ (the sign changes) and $|L_1 - L_2| \geq T$ (the change is large enough). Drawbacks: it is **very sensitive to noise** and is a **scalar**, so it gives no edge orientation.

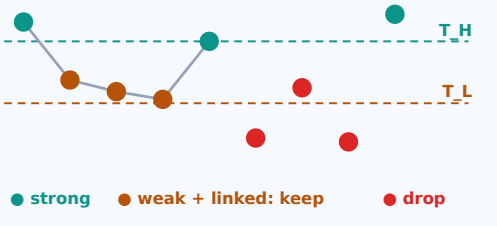
Laplacian of Gaussian (LoG, the Marr–Hildreth operator)



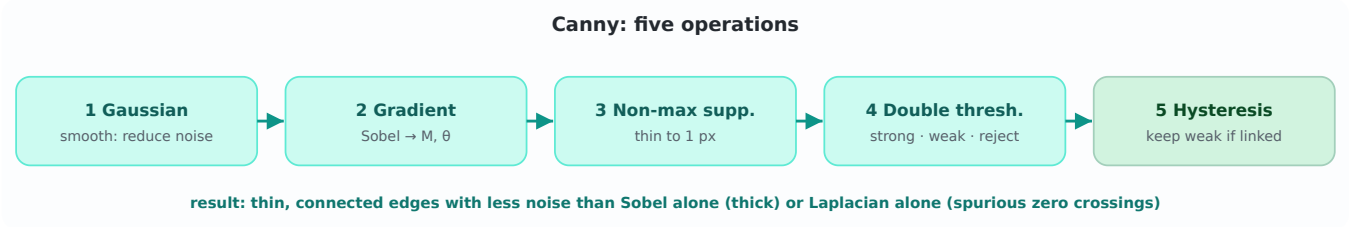
›G kernel (“Mexican hat”): Gaussian smooth + Laplacia

LoG(I) = $\nabla^2(G_\sigma * I)$, which fixes the Laplacian’s noise problem by smoothing first. Steps: (1) smooth with a Gaussian, (2) take the Laplacian of the smoothed image, (3) find **significant zero crossings**, (4) reject weak responses with a threshold. A bigger σ finds coarser edges and ignores more noise.

Canny edge detector



Why we need it: simple gradient operators such as Sobel give **thick edges and respond to noise**, and Laplacian methods give **multiple or spurious zero crossings**. Canny aims for **clear, thin, connected edges** with less noise, in five operations.



Step 2: $G_x = S_x * I_s$, $G_y = S_y * I_s$, then M and θ. Step 3: keep a pixel only if its M is a local maximum compared with its two neighbours along θ. Step 4: $M \geq T_H$ is strong, $T_L \leq M < T_H$ weak, $M < T_L$ rejected. Step 5: strong edges are kept, weak ones only if connected to a strong one.

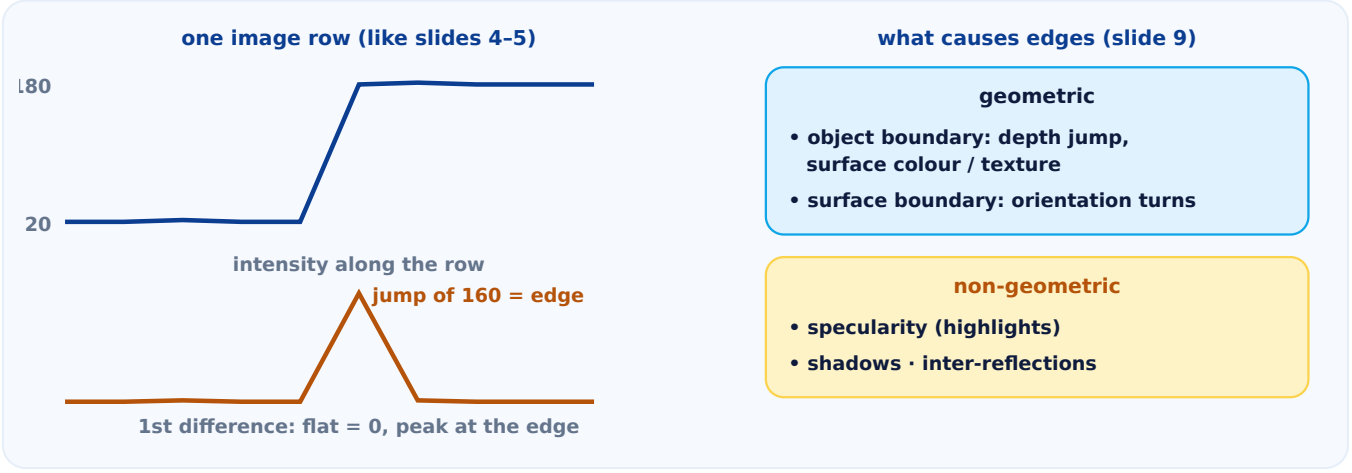
DETECTOR	DERIVATIVE	STRENGTH	WEAKNESS
Prewitt	1st	simple, cheap	noise-sensitive, thick edges
Sobel	1st (weighted smoothing)	a little more noise-robust than Prewitt	still thick edges
Laplacian	2nd	one isotropic mask, precise location	very noise-sensitive, no orientation, spurious zero crossings
LoG	2nd after Gaussian	noise handled by σ	can round corners; σ must be chosen
Canny	1st + post-processing	thin, connected, low-noise edges	more steps; two thresholds to tune

Step by step — deeper notes

Everything here follows the Topic 5 slides; each part shows its slide numbers. Parts marked **added** are study aids we wrote, not lecture content.

1 · An edge is a jump in a row of pixels

Topic 5 · slides 3–9



Slide 3: edges are **significant local changes of intensity**, usually on the boundary between two regions. Slides 4–5 draw one image row as a graph: flat stretches are regions, sudden jumps are edges.

Slide 8 separates three words: an **edge pixel** is where the intensity changes abruptly; an **edge** is a set of connected edge pixels; an **edge detector** finds edge pixels. The goal (slide 6) is a **line drawing** of the scene, from which corners, lines and curves are taken for recognition and segmentation.

In the toy row, the step from 20 to 180 gives a difference of **160**; the small wobble 20 → 22 gives only 2. Slide 7 shows the same idea on a photo: several detectors, several line drawings.

2 · Derivatives of the three slide-12 pictures

Topic 5 · slides 10-12, 15

slide 12: “what would be the derivatives?” (one row of each, forward differences)



uniform black

I	0	0	0	0	0
Ix	0	0	0	0	

Ix = 0, Iy = 0 everywhere



left-to-right ramp

I	0	50	100	150	200
Ix	50	50	50	50	

Ix = 50 everywhere, Iy = 0



white box on black

I	0	0	255	255	0
Ix	0	255	0	−255	

non-zero only at the box border

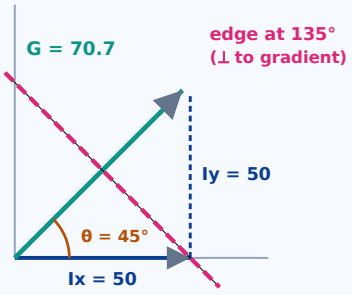
Slides 10–11: treat the image as a function I(x, y): $\mathbb{R}^2 \to \mathbb{R}$. Its rate of change is the first derivative, taken in both directions: $\mathbf{I_x = \partial I/\partial x}$ and $\mathbf{I_y = \partial I/\partial y}$.

Slide 12 asks for the derivatives of three pictures. **Uniform black:** nothing changes, so both are 0. **Ramp:** each step right adds 50, so $I_x = 50$ everywhere (constant, no single edge) and $I_y = 0$. **White box:** zero inside and outside, +255 entering the box and −255 leaving it; along the top and bottom sides I_y does the same.

added The rows of numbers and the forward difference I(x+1) − I(x) (slide 15) are ours; the slide shows only the three pictures.

3 · Gradient magnitude and direction: slide 14 worked

Topic 5 · slides 13-14



Slide 13 stacks the two derivatives into the **gradient** $\nabla I = [I_x, I_y]$, with magnitude $\mathbf{G = \sqrt{(I_x^2 + I_y^2)}}$ and direction $\boldsymbol{\theta = \tan^{-1}(I_y/I_x)}$.

Slide 14 worked: $I_x = I_y = 50$. ① $G = \sqrt{(2500 + 2500)} = \sqrt{5000} \approx \mathbf{70.7}$. ② The direction of maximum increase is $\theta = \tan^{-1}(50/50) = \tan^{-1}(1) = \mathbf{45°}$. ③ The edge runs **perpendicular** to that: 45° + 90° = **135°** (the same line as −45°).

The gradient points from dark to bright, *across* the edge; the edge line runs *along* the boundary.

4 · From differences to Prewitt and Sobel, then a threshold

Topic 5 · slides 15–22

row: three differences at x

10	20	40	70
	x−1	x	x+1

forward 70 − 40 = 30
backward 40 − 20 = 20
central (70 − 20)/2 = 25

patch

10	10	50
10	10	50
10	10	90

Prewitt Sx

−1	0	1
−1	0	1
−1	0	1

Gx = 40 + 40 + 80 = 160

Sobel Sx

−1	0	1
−2	0	2
−1	0	1

Gx = 40 + 2·40 + 80 = 200

Sobel Gy (transposed mask) = (10+20+90) − (10+20+50) = 40

G = √(200² + 40²) ≈ 204 ≥ T = 150 → E = 1 (edge pixel)

Slide 15: an image has only whole-pixel samples, so the derivative is approximated by a **forward**, **backward** or **central difference**. On the row 10 20 40 70 at x (value 40) they give 30, 20 and 25.

Slide 16: differences amplify noise, so take the derivative in one direction and **smooth in the perpendicular direction**. Slide 17: **Prewitt** = [−1 0 1] across × [1 1 1]^T down; **Sobel** = the same × weighted [1 2 1]^T. On the patch, Prewitt sums the three row differences 40 + 40 + 80 = **160**; Sobel doubles the middle one: **200**.

Slide 18: threshold the magnitude into a binary map, E[i, j] = 1 if G[i, j] ≥ T, else 0. Slide 22 lists ways to choose T: manual, histogram analysis, percentile, Otsu, adaptive. added T = 150 and the patch are our example; slides 19–21 show the two operators on a photo.

5 · Laplacian: zero crossings, the threshold, and LoG

Topic 5 · slides 23–29

clean step: Ixx = I(x+1) − 2I(x) + I(x−1)

I	10	10	10	90	90	90
Ixx		0	80	−80	0	

80 · (−80) < 0 and |80 − (−80)| = 160 ≥ T = 100 → edge

4-neighbour mask

`[0 1 0; 1 −4 1; 0 1 0]`

centre 10 with right column 90: 10+10+90+10−40 = 80

noise: Laplacian alone vs LoG

I	10	10	14	10	10
Ixx		4	−8	4	

crossings, but |4 − (−8)| = 12 < 100 → rejected

Gσ∗I	10	11	12	11	10
∇²		0	−2	0	

after smoothing ([1 2 1]/4): no sign change at all

Slide 23: the second derivative measures how the rate of change changes, $\nabla^2 I = \partial^2 I/\partial x^2 + \partial^2 I/\partial y^2$. Slide 24's approximation $I_{xx} = I[i, j+1] - 2I[i, j] + I[i, j-1]$ (and I_{yy} the same way down) gives the **4-neighbour mask**.

On a clean step the response goes 0, **80, −80**, 0: the edge sits at the **zero crossing** between 80 and −80. Slide 25 marks it only if **$L_1L_2 < 0$** and **$|L_1 - L_2| \geq T$**: here −6400 < 0 and 160 ≥ 100. A noise bump of 4 also flips sign (4, −8, 4) but its jump of 12 fails the threshold.

Slide 28: the Laplacian is noise-sensitive and a scalar, so it gives **no edge orientation**. Slide 29's **LoG = $\nabla^2(G_\sigma * I)$** (Marr–Hildreth) smooths first, then: Laplacian → significant zero crossings → threshold. added Smoothing with [1 2 1]/4 (the 1-D row of Topic 4's Gaussian) turns the bump into 10 11 12 11 10, and the crossings vanish.

6 · Canny steps 1–2 on a small image: smooth, then gradient

Topic 5 · slides 30–32 · Topic 4 · slide 29

I (5 × 7)

10	10	10	10	50	90	90
10	10	10	10	50	90	90
10	10	10	10	45	80	80
10	70	10	10	40	70	70
10	10	10	10	35	60	60

noise pixel 70

① $I_s = G_\sigma * I$

10	10	10	20	50	80	90
10	10	10	19.7	48.8	77.8	87.5
13.8	17.5	13.8	18.8	45	71.2	80
17.5	25	17.5	17.5	40	62.5	70
13.8	17.5	13.8	16.6	36.2	55.9	62.5

70 → 25 after smoothing

② $M = \sqrt{G_x^2 + G_y^2}$

0	0	40	159	238	159	41
19	23	34	149	230	156	54
40	45	28	124	211	150	76
23	0	15	100	185	134	74
27	23	19	90	164	111	40

thick ridge: 124 · 211 · 150

pixel (row 2, col 4): $I_s = (10+2\cdot50+90 + 2\cdot10+4\cdot45+2\cdot80 + 10+2\cdot40+70)/16 = 720/16 = 45$

$G_x = (77.8-19.7) + 2(71.25-18.75) + (62.5-17.5) = 208.1 \cdot G_y = 160 - 195 = -35$

$M = \sqrt{208.1^2 + 35^2} \approx 211 \cdot \theta = \tan^{-1}(-35/208.1) \approx -9.5°$ (gradient almost horizontal)

Slide 30: Sobel gives thick, noise-prone edges and the Laplacian spurious zero crossings; Canny aims for **clear, thin, connected** edges. Slide 31 lists five steps; here are steps 1–2 on a small image: a vertical edge that fades from 10|90 at the top to 10|60 at the bottom, plus one noise pixel (70).

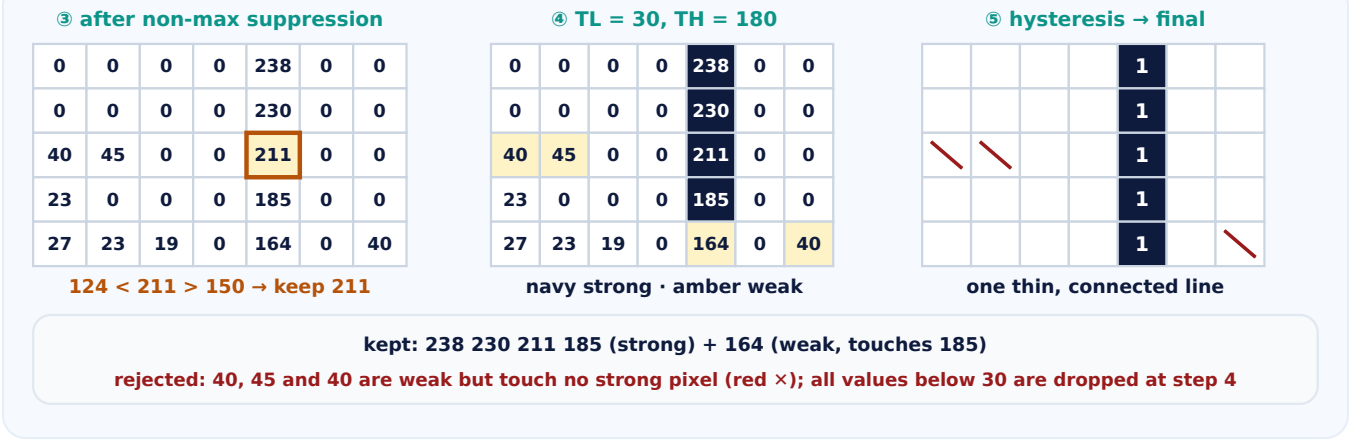
Step 1 (slide 32): $I_s = G_\sigma * I$ with the 3 × 3 Gaussian from Topic 4 slide 29, H = 1/16 [1 2 1; 2 4 2; 1 2 1]. The noise pixel drops from 70 to **25**.

Step 2: $G_x = S_x * I_s$, $G_y = S_y * I_s$ with the Sobel masks, then M and θ. The box shows every number for the highlighted pixel. The edge is now a **3-pixel-wide ridge** in M (124, 211, 150 in row 2).

added Border pixels copy their nearest neighbour, and masks are laid on the image without flipping; the slides don't specify either. Values are rounded for display.

7 · Canny steps 3–5: thin, classify, connect

Topic 5 · slides 33–36



kept: 238 230 211 185 (strong) + 164 (weak, touches 185)

rejected: 40, 45 and 40 are weak but touch no strong pixel (red ×); all values below 30 are dropped at step 4

Step 3 · non-maximum suppression (slide 33): for each pixel, find the gradient direction and compare M with the two neighbours *approximately along* it; keep it only if it is the local maximum. Row 2 has θ ≈ −9.5°, so compare left and right: 124 < **211** > 150. Only 211 survives; the ridge becomes one pixel wide.

Step 4 · double threshold (slide 34): $M \geq T_H = 180$ is **strong** (238, 230, 211, 185); $T_L = 30 \leq M < T_H$ is **weak** (164, and the noise leftovers 40, 45, 40); $M < T_L$ is rejected (23, 27, 19 …).

Step 5 · hysteresis (slide 35): strong pixels are accepted; a weak pixel is accepted only if **connected to a strong one**. 164 touches 185, so the faint bottom of the edge survives; the noise leftovers touch nothing strong and go. Slide 36 shows the result on a photo. **added** T_L, T_H and "connected" = the 8 surrounding pixels are our choices.

8 · Exam traps

Topic 5 · slides 13–14, 17, 25, 28, 31–35 **added**

- **Edge orientation is not θ:** θ is the gradient (maximum increase); the edge is perpendicular, θ + 90° (slides 13–14).
- **Prewitt vs Sobel** differ only in the smoothing vector, $[1\ 1\ 1]^T$ vs $[1\ 2\ 1]^T$; the derivative part $[-1\ 0\ 1]$ is the same (slide 17).
- **A sign change alone is not an edge:** a zero crossing needs $L_1L_2 < 0$ *and* $|L_1 - L_2| \geq T$ (slide 25).
- **The Laplacian gives no orientation**: it is a scalar (slide 28). Gradient methods give θ.
- **Canny order:** smooth → gradient → non-max suppression → double threshold → hysteresis (slide 31). Thinning comes *before* thresholding.
- **Weak ≠ rejected:** a weak pixel survives if it is connected to a strong one (slide 35).

9 · Quick check

Topic 5 · slides 13, 25, 34–35 **added**

▼ **Q1.** A pixel has $I_x = 0$ and $I_y = 80$. Give G, θ and the edge orientation.

G = √(0 + 6400) = **80**; θ = **90°** (the gradient points straight along y); the edge is perpendicular to it, a **horizontal** edge at 0° (slide 13).

▼ **Q2.** Neighbouring Laplacian responses are 30 and −10, with T = 50. Is there an edge?

30 · (−10) = −300 < 0, so the sign changes, but |30 − (−10)| = 40 < 50. **No edge**: both conditions must hold (slide 25).

▼ **Q3.** Canny with $T_L = 50$, $T_H = 150$: a pixel with M = 100 sits next to a pixel with M = 200. What happens to it?

100 is a **weak** candidate (slide 34); its neighbour is **strong**, so hysteresis **keeps** it (slide 35).

Key terms

edge | **edge pixel** | **edge detector** | **object / surface boundary** | **specularity · shadow · inter-reflection** | **partial derivatives I_x, I_y** | **gradient ∇I** | **magnitude G** | **direction θ** | **forward / backward / central difference** | **Prewitt** | **Sobel** | **threshold T** | **Otsu's method** | **Laplacian ∇²** | **zero crossing** | **LoG / Marr–Hildreth** | **Canny** | **non-maximum suppression** | **double thresholding** | **hysteresis**

Exam relevance

Try each question yourself first, then reveal the answer.

Q1. "What is an edge? List the causes of edges in an image."

Answer: Edges are **significant local changes of intensity**, typically on the boundary between two regions. Edge pixels are where intensity changes abruptly, and an edge is a set of connected edge pixels. Edge detection aims to produce a **line drawing** of the scene from which features such as corners, lines and curves are taken for recognition and segmentation. **Geometric causes:** object boundaries (a break in depth and/or surface colour or texture) and surface boundaries (a break in surface orientation and/or colour or texture). **Non-geometric causes:** specularity, shadows and inter-reflections.

Q2. "An edge pixel has $I_x = 50$ and $I_y = 50$. Find G, the direction of maximum increase, and the local edge orientation."

Answer: $G = \sqrt{(50^2 + 50^2)} = \sqrt{5000} \approx$ **70.7**. Direction of maximum increase $\theta = \tan^{-1}(50/50) =$ **45°**, which is where the gradient points. The edge runs **perpendicular** to the gradient, so its orientation is 45° + 90° = **135°** (equivalently −45°).

Q3. "Explain the Prewitt and Sobel operators and how a binary edge map is produced."

Answer: Both approximate the first derivative in one direction with [−1 0 1] and **smooth in the perpendicular direction** to reduce noise. **Prewitt** smooths with $[1\ 1\ 1]^T$, giving Gx = [−1 0 1; −1 0 1; −1 0 1]. **Sobel** uses a weighted smoothing $[1\ 2\ 1]^T$, giving Gx = [−1 0 1; −2 0 2; −1 0 1], which is a little more robust to noise. Transpose each for Gy. Compute $G = \sqrt{(Gx^2 + Gy^2)}$, then threshold: E[i,j] = 1 if G ≥ T, else 0. T can be chosen manually, from the histogram, as a percentile, by Otsu's method, or adaptively.

Q4. "How does the Laplacian detect edges? What are its drawbacks, and how does LoG address them?"

Answer: The Laplacian $\nabla^2 I = \partial^2 I/\partial x^2 + \partial^2 I/\partial y^2$ measures how the rate of change changes (4-neighbour mask [0 1 0; 1 −4 1; 0 1 0]). An edge lies at a **zero crossing**: neighbouring responses L_1 and L_2 must have $\mathbf{L_1 \cdot L_2 < 0}$ and $\mathbf{|L_1 - L_2| \geq T}$. **Drawbacks:** very noise-sensitive (spurious zero crossings) and a scalar response, so no edge orientation. **LoG (Marr–Hildreth)** $= \nabla^2(G_\sigma * I)$: smooth with a Gaussian first, then take the Laplacian, find significant zero crossings, and reject weak ones with a threshold.

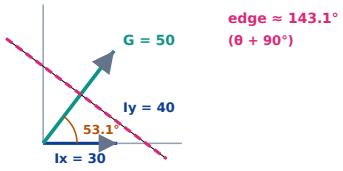
Q5. "Describe the five steps of the Canny edge detector and why it is needed."

Answer: Why: Sobel-type operators give **thick, noise-sensitive** edges, and Laplacian methods give **spurious zero crossings**. Canny produces thin, connected, low-noise edges. (1) **Gaussian smoothing**: $I_s = G_\sigma * I$. (2) **Gradient**: Sobel G_x, G_y, then magnitude M and angle θ. (3) **Non-maximum suppression**: keep a pixel only if its M is larger than its two neighbours along the gradient direction, which thins edges to one pixel. (4) **Double thresholding**: $M \geq T_H$ strong, $T_L \leq M < T_H$ weak, $M < T_L$ rejected. (5) **Hysteresis tracking**: accept strong pixels, accept weak pixels connected to strong ones, and reject isolated weak pixels.

Practice: 10 questions

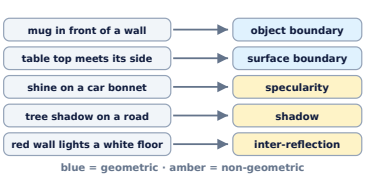
Exam-style questions on Topic 5: calculate, classify, compare, apply, explain, match, outline. Answer on paper first, then reveal. Each answer cites its slides; **added** marks anything that goes beyond the slides. "Show all answers" above opens these too.

P1. **CALCULATE** An edge pixel has $I_x = 30$ and $I_y = 40$. Find the gradient magnitude G, the direction of maximum increase θ, and the local edge orientation.



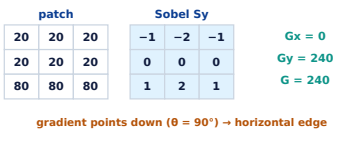
Answer · Topic 5 slides 13–14**:** $G = \sqrt{(30^2 + 40^2)} = \sqrt{(900 + 1600)} = \sqrt{2500} = \mathbf{50}$. $\theta = \tan^{-1}(40/30) = \tan^{-1}(1.333) \approx \mathbf{53.1°}$: the gradient points that way, from dark to bright. The edge runs perpendicular: 53.1° + 90° ≈ **143.1°** (the same line as −36.9°). Same method as the slide 14 example, with different numbers.

P2. **CLASSIFY** Classify the cause of each edge as geometric (object or surface boundary) or non-geometric (specularity, shadow, inter-reflection): (a) a mug against the wall behind it (b) where a table top meets its side (c) a bright shine on a car bonnet (d) a tree's shadow on a road (e) a red wall casting reddish light onto a white floor.



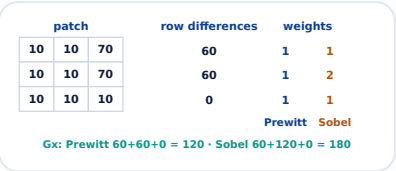
Answer · Topic 5 slide 9**:** (a) **geometric, object boundary**: a jump in depth between the mug and the wall. (b) **geometric, surface boundary**: the surface orientation changes at the corner. (c) **non-geometric, specularity**: a highlight, not a real boundary. (d) **non-geometric, shadow**: the road is the same, only the lighting changes. (e) **non-geometric, inter-reflection**: light bounced from one surface onto another. Non-geometric edges are real intensity changes, but they are not object outlines.

P3. **CALCULATE** Apply the Sobel masks to the 3 × 3 patch [20 20 20; 20 20 20; 80 80 80]. Find G_x, G_y, G and say which way the edge runs.



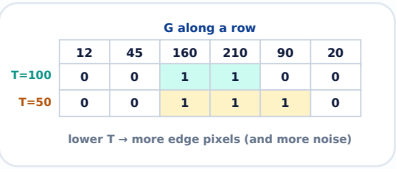
Answer · Topic 5 slides 13, 17**:** $S_x = [-1\ 0\ 1; -2\ 0\ 2; -1\ 0\ 1]$: each row is flat, so $\mathbf{G_x = 0}$. S_y (the transpose) = bottom row weighted [1 2 1] minus top row weighted [1 2 1]: (80 + 160 + 80) − (20 + 40 + 20) = 320 − 80 = **240**. $G = \sqrt{(0^2 + 240^2)} = \mathbf{240}$. $\theta = \tan^{-1}(240/0) = 90°$: the gradient points along y (down the rows), so the edge is **horizontal**, between the 20-rows and the 80-row.

P4. **COMPARE** Compute G_x for the patch [10 10 70; 10 10 70; 10 10 10] with Prewitt and with Sobel, and explain where the difference comes from.



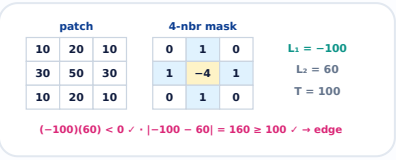
Answer · Topic 5 slides 17, 19–21**:** Both take the difference right − left in each row: 60, 60, 0. **Prewitt** adds them equally ($[1\ 1\ 1]^T$): 60 + 60 + 0 = **120**. **Sobel** weights the centre row double ($[1\ 2\ 1]^T$): 60 + 2·60 + 0 = **180**. The derivative part is identical; only the smoothing vector differs. Sobel trusts the pixel's own row more than its neighbours. Slides 19–21 compare the two on the cameraman photo.

P5. **APPLY** Gradient magnitudes along a row are 12, 45, 160, 210, 90, 20. Write the binary edge map for T = 100 and for T = 50, and say what lowering T does.



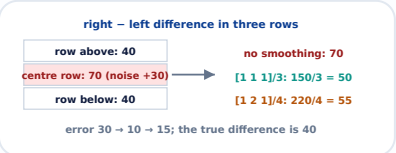
Answer · Topic 5 slides 18, 22**:** E = 1 when G ≥ T (slide 18). **T = 100:** 0 0 1 1 0 0. **T = 50:** 0 0 1 1 1 0, since 90 now passes. A lower T keeps weaker edges, but also more noise responses; a higher T gives cleaner maps with possible gaps. That is why slide 22 lists ways to pick T: manual, histogram, percentile, Otsu, adaptive. Neither map is thin here (two or three 1s side by side), which is the thick-edge problem Canny fixes.

P6. **CALCULATE** Apply the 4-neighbour Laplacian to the centre of [10 20 10; 30 50 30; 10 20 10]. Its neighbour's response is $L_2 = 60$. With T = 100, is there an edge between them?



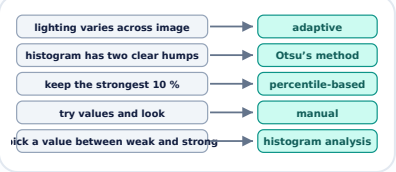
Answer · Topic 5 slides 24–25**:** Mask [0 1 0; 1 −4 1; 0 1 0]: $L_1 = 20 + 30 + 30 + 20 - 4 \cdot 50 = 100 - 200 =$ **−100**. Zero-crossing test (slide 25): $L_1 L_2 = (-100)(60) = -6000 < 0$ ✓, and $|L_1 - L_2| = |-160| = 160 \geq 100$ ✓. Both hold, so **an edge is marked** between the two pixels. A negative Laplacian at a bright centre is expected: it is a local peak.

P7. **EXPLAIN** Explain why Prewitt and Sobel smooth in the direction perpendicular to the derivative. Use three rows whose right − left differences are 40, 70 (noise +30) and 40.



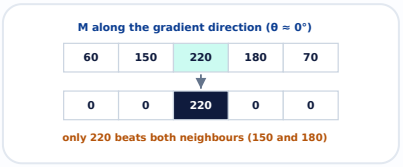
Answer · Topic 5 slides 15–17**:** Differences amplify noise (slide 15 approximations work on single samples). Slide 16's two-step procedure takes the derivative in one direction and **averages the neighbouring responses in the perpendicular direction**. Using the centre row alone gives 70, off by 30. **added** Averaging with [1 1 1] (Prewitt) gives (40 + 70 + 40)/3 = **50**; with [1 2 1] (Sobel) (40 + 140 + 40)/4 = **55**. The noise is spread over three rows, so a real edge (all rows ≈ 40) stays strong while a one-row blip is weakened. The normalising /3 and /4 are ours; the slide masks leave them out.

P8. **MATCH** Match each situation to the threshold-selection method from the slides: (a) the lighting and noise vary across the image (b) the magnitude histogram has two clear groups (c) you want to keep the strongest 10 % of responses (d) a quick one-off image you can inspect (e) you study the magnitude distribution to find a value separating weak from strong.



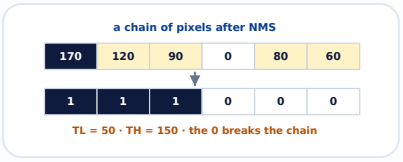
Answer · Topic 5 slide 22**:** (a) **adaptive thresholding**: T varies across the image. (b) **Otsu's method**: automatically splits the histogram into two groups, when that split is appropriate. (c) **percentile-based**: T = the 90th percentile of the magnitudes. (d) **manual**: experiment and inspect. (e) **histogram analysis**: examine the distribution and pick a separating value.

P9. APPLY After Canny step 2, five pixels across an edge (along the gradient direction) have M = 60, 150, 220, 180, 70. Apply non-maximum suppression. What is left, and why is this step needed?



Answer · Topic 5 slides 30, 33**:** Compare each pixel with its two neighbours along the gradient (slide 33). 150 < 220 and 180 < 220 are not maxima; 60 and 70 aren't either. Only **220** is a local maximum, so the row becomes **0 0 220 0 0**. Gradient responses spread over several pixels (here five), so plain Sobel gives thick edges (slide 30). NMS thins each edge to **one pixel**, before any threshold is applied.

P10. OUTLINE Outline Canny steps 4 and 5 on this chain of neighbouring pixels after NMS: M = 170, 120, 90, 0, 80, 60, with $T_L = 50$ and $T_H = 150$.



Answer · Topic 5 slides 34–35**: Step 4, double threshold** (slide 34): 170 ≥ 150 is **strong**; 120, 90, 80, 60 lie between 50 and 150, so they are **weak**; 0 < 50 is rejected. **Step 5, hysteresis** (slide 35): accept the strong 170; accept 120 (touches 170) and 90 (touches 120, connected through the chain). 80 and 60 are weak but cut off by the 0, so no strong pixel reaches them: **rejected**. Result: 1 1 1 0 0 0. added "Connected" through a chain of weak pixels is the usual reading of slide 35.

Quick MCQs: check your understanding

M1. According to the slides, edges are... CHOOSE 1

A. significant local changes of intensity in an image
B. the brightest pixels in an image
C. regions of constant intensity
D. the pixels on the image border

M2. Which are non-geometric causes of edges? CHOOSE ALL THAT APPLY

A. Specularity
B. Shadows
C. Inter-reflections
D. Discontinuity in depth

M3. An edge pixel has Ix = 50 and Iy = 50. What is G? CHOOSE 1

A. 100
B. 50
C. ≈ 70.7
D. 2500

M4. How is the local edge orientation related to the gradient direction θ? CHOOSE 1

A. It is the same as θ
B. It is perpendicular: θ + 90°
C. It is θ / 2
D. It is always 0°

M5. For a uniform black image (slide 12), what are Ix and Iy? CHOOSE 1

A. Both 0
B. Ix = 255, Iy = 0
C. Both 255
D. Undefined

M6. Which are ways the slides give to approximate the first derivative? **CHOOSE ALL THAT APPLY**

A. Forward difference
B. Backward difference
C. Central difference
D. The Laplacian

M7. What is the only difference between the Prewitt and Sobel operators? **CHOOSE 1**

A. Sobel uses $[-1\ 0\ 1]$, Prewitt uses $[1\ -2\ 1]$
B. Sobel smooths with $[1\ 2\ 1]^T$, Prewitt with $[1\ 1\ 1]^T$
C. Sobel is a second-derivative filter
D. Prewitt needs a Gaussian first

M8. In the binary edge map, E[i, j] = 1 when... **CHOOSE 1**

A. $G[i, j] < T$
B. $G[i, j] \geq T$
C. $G[i, j] = 0$
D. $\theta[i, j] \geq T$

M9. To mark an edge at a Laplacian zero crossing between L1 and L2, which conditions must hold? **CHOOSE ALL THAT APPLY**

A. $L1 \cdot L2 < 0$
B. $|L1 - L2| \geq T$
C. $L1 + L2 = 0$
D. L1 and L2 both positive

M10. Which are drawbacks of the Laplacian operator given in the slides? **CHOOSE ALL THAT APPLY**

A. It is sensitive to noise
B. It is a scalar, so it gives no edge orientation
C. It needs two thresholds TL and TH
D. It only works on colour images

M11. LoG is also known as the... **CHOOSE 1**

A. Canny operator
B. Sobel operator
C. Marr–Hildreth operator
D. Prewitt operator

M12. What is the correct order of the Canny steps? **CHOOSE 1**

A. Gradient → Gaussian → double threshold → NMS → hysteresis
B. Gaussian → gradient → non-max suppression → double threshold → hysteresis
C. Gaussian → double threshold → gradient → hysteresis → NMS
D. NMS → Gaussian → gradient → hysteresis → double threshold

M13. Non-maximum suppression keeps a pixel only if... **CHOOSE 1**

A. its M is above TH
B. its M is a local maximum compared with its two neighbours along the gradient direction
C. it is connected to a strong edge
D. its θ is 0°

M14. In hysteresis tracking, a weak edge pixel not connected to any strong pixel is... **CHOOSE 1**

A. accepted
B. turned into a strong pixel
C. rejected
D. smoothed again

Answer key

1. **M1: A**. Slide 3: edges are significant local intensity changes, usually on the boundary between two regions. **Topic 5 · slide 3**
2. **M2: A, B, C**. Specularity, shadows and inter-reflections are non-geometric. A depth discontinuity is a geometric cause (object boundary). **Topic 5 · slide 9**
3. **M3: C**. $G = \sqrt{(50^2 + 50^2)} = \sqrt{5000} \approx 70.7$. Adding the two (100) is the common slip. **Topic 5 · slides 13–14**
4. **M4: B**. The gradient points across the edge (maximum increase); the edge runs perpendicular, e.g. 45° → 135°. **Topic 5 · slide 14**
5. **M5: A**. Nothing changes in either direction, so both derivatives are zero everywhere. **Topic 5 · slide 12**
6. **M6: A, B, C**. Slide 15 lists forward, backward and central differences. The Laplacian is a second derivative. **Topic 5 · slides 15, 23**
7. **M7: B**. Both use the $[-1\ 0\ 1]$ derivative mask; Sobel's smoothing vector is weighted $[1\ 2\ 1]^T$. **Topic 5 · slide 17**
8. **M8: B**. Slide 18: pixels whose gradient magnitude reaches the threshold become edge pixels. **Topic 5 · slide 18**
9. **M9: A, B**. The sign must change (product negative) and the jump must be large enough. They need not sum to zero. **Topic 5 · slide 25**
10. **M10: A, B**. Slide 28: noise-sensitive and no orientation. Two thresholds belong to Canny. **Topic 5 · slide 28**
11. **M11: C**. $\text{LoG}(I) = \nabla^2(G\sigma * I)$ is the Marr–Hildreth operator. **Topic 5 · slide 29**
12. **M12: B**. Smooth first, then differentiate, thin, classify, and finally connect. **Topic 5 · slide 31**
13. **M13: B**. NMS thins broad responses by keeping only local maxima along the gradient. Thresholds and connectivity come in steps 4–5. **Topic 5 · slide 33**
14. **M14: C**. Strong pixels are accepted, weak ones only when connected to strong ones; isolated weak pixels are rejected. **Topic 5 · slide 35**

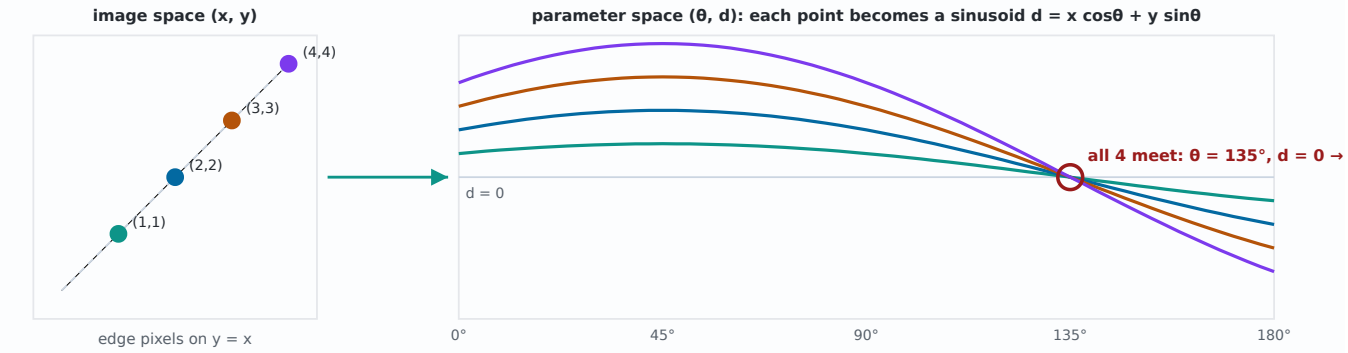
L06 Hough transform

How to get from scattered edge pixels to whole lines. Each point is turned into a curve in (d, θ) parameter space, curves that cross are counted in an accumulator, and the peak is read back as a line. Worked through with the slide’s 5 × 5 example. **slides**

Short Note — Quick Revision

- **Problem:** edges are local. They don’t say which pixels belong to the **same line**, or whether a line exists despite gaps, noise or occlusion.
- **y = mx + c fails** for vertical lines (infinite slope). Use **polar form x cosθ + y sinθ = d**: d = perpendicular distance from the origin, θ = angle of that perpendicular to the x-axis.
- **Core idea:** image space → parameter space. One point → one **sinusoid**. Points on the same line → curves that **intersect** at that line’s (d, θ).
- **Accumulator A(d, θ):** a 2-D array of vote counts. Each edge pixel votes for every (d, θ) line through it; a **peak** = a likely line.
- **Steps:** edge pixels → coordinates → sinusoids → vote → peak → (d, θ) → draw the line. Real images: use a threshold A ≥ T (not universal).
- **Extensions:** circles (a, b, r: a 3-D accumulator) · probabilistic Hough (random subset, faster, gives segments).

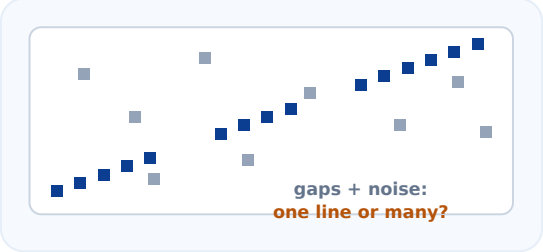
The core idea: image space → parameter space



Each edge point (x_0, y_0) stands for **every** line through it. In (θ, d) space that is one sinusoid. **Points on the same line give curves that meet at one place**, and that meeting point is the line’s (d, θ). These are the four pixels of the slide example.

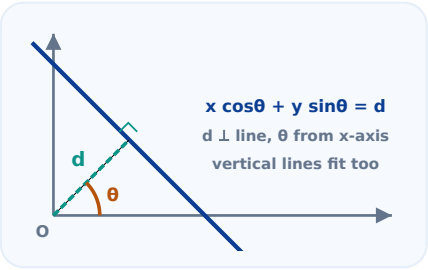
Discussion (following the lecture)

The problem



An edge detector tells us *where* intensity changes sharply, which pixels probably lie on boundaries, and where there is local structure. It does **not** say whether several edge pixels lie on the **same line**, whether they form a common geometric shape, whether a line is there **despite gaps, noise or partial occlusion**, or whether a broken boundary belongs to a **circle**. The question is: "*How can we find a global geometric structure from a set of local edge points?*" Example: P1 (1,2), P2 (2,3) and P3 (3,4) all lie on **y = x + 1**. Through any single point, infinitely many lines pass.

Why polar form instead of y = mx + c



Slope–intercept form has two degrees of freedom (m, c), and two points pin them down. It **fails for vertical lines**: x = 1 has an infinite slope and no intercept, and m has no upper bound, so it can't be put into a finite grid. **Polar (normal) form** describes a line by its **perpendicular distance d from the origin** and the **angle θ that this perpendicular makes with the x-axis**: **x cosθ + y sinθ = d**. Both are bounded (θ ∈ [0°, 180°), |d| ≤ the image diagonal), so every line, vertical ones included, fits in a finite table. (One slide writes it with −y sinθ. That is the same idea with the y-axis pointing down, as image rows do; use one convention throughout an answer.)

LINE	POLAR FORM	WHY
x = 3	**θ = 0°, d = 3**	the perpendicular from the origin lies along the x-axis and is 3 long
y = x + 1	**θ = 135°, d = 1/√2 ≈ 0.71**	rewrite as −x + y = 1, divide by √2: x·cos135° + y·sin135° = 0.71

The Hough accumulator and voting



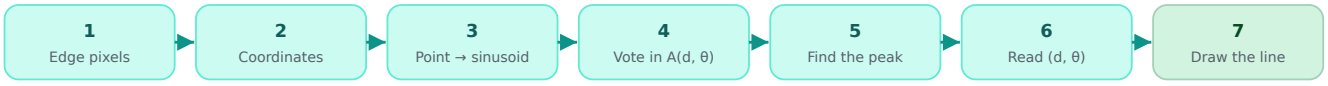
One edge point gives one sinusoid, and points on a common line give sinusoids that **intersect**. We can't draw infinitely many curves and search for exact intersections, so we **discretise**. The **accumulator A(d, θ)** is a 2-D array that counts how many edge points support each (d, θ) pair. **Voting**: each edge pixel casts one vote per θ value (one candidate line through it), and pairs with many votes are probably real lines.

Slide example: a 5 × 5 binary image with edge pixels on the diagonal, (1,1), (2,2), (3,3), (4,4), and θ ∈ {0°, 45°, 90°, 135°} (real systems use a much finer step). Compute d = x cosθ + y sinθ for each:

PIXEL (X, Y)	Θ = 0°	Θ = 45°	Θ = 90°	Θ = 135°
(1,1)	1	1.41	1	**0**
(2,2)	2	2.83	2	**0**
(3,3)	3	4.24	3	**0**
(4,4)	4	5.66	4	**0**

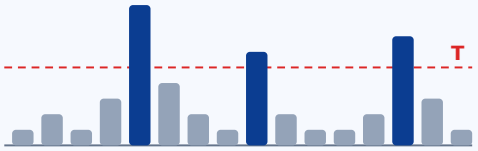
Every other (d, θ) cell gets one vote, but **(d = 0, θ = 135°) gets 4 votes**, which is the peak. Back in image space: x cos135° + y sin135° = 0 → −x + y = 0 → **y = x**, the diagonal through all four pixels.

Hough transform: the seven steps



1 get edge pixels · 2 extract their coordinates · 3 turn each into a sinusoid in parameter space · 4 let the curves vote in the accumulator · 5 find the peak · 6 read off (d, θ) · 7 rebuild the line in image space.

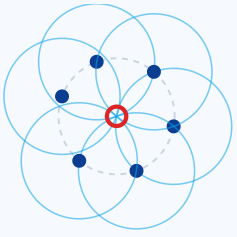
Finding peaks in real images



cells with A(d,θ) ≥ T = candidate lines

max(A) is the strongest line candidate (in the slide matrix A the single 3 is the winner). Real images are not that clean: **noise, several lines, broken edges, lines close together and false edges** all add votes. So a **minimum threshold T** is used, **A(d, θ) ≥ T**, and every cell above it is a candidate line. **T is not universal**: it depends on the image and on how long the lines are.

Extensions



circle: each point votes for centre (a, b) and r

probabilistic: random subset

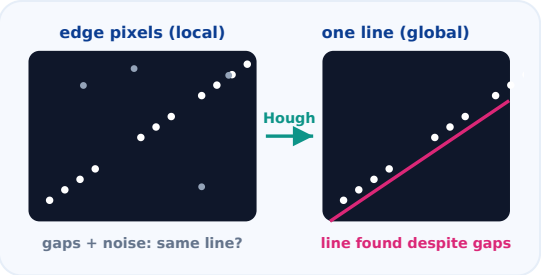
Circles: the same voting idea with parameters (a, b, r) for the centre and radius, so the accumulator becomes 3-D. **Probabilistic Hough transform**: votes with a random subset of edge points, which is much faster and returns line *segments* with end points.

Step by step — deeper notes

Everything here follows the Topic 6 slides; each part shows its slide numbers. Parts marked **added** are study aids we wrote, not lecture content.

1 · The problem: local edges, global shapes

Topic 6 · slides 3–6



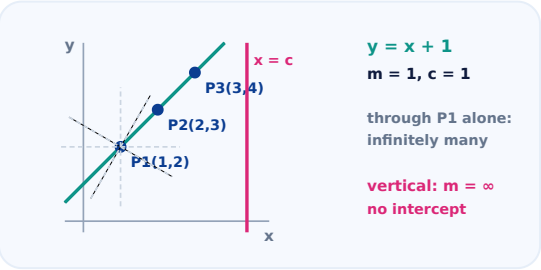
An **edge detector** tells us where significant intensity changes occur, which pixels probably belong to object boundaries, and where the image has local structure (slide 3; slide 4 shows a car and its edge map).

What edges **don't** say (slide 5): whether several edge pixels belong to the **same line**; whether a set of them forms a common geometric structure; whether a line is present **despite gaps, noise or partial occlusion**; whether an incomplete boundary belongs to a **circle**.

So the question of the topic (slide 6): **"How can we identify a global geometric structure from a collection of local edge points?"**

2 · Three points, and why y = mx + c breaks

Topic 6 · slides 7–9

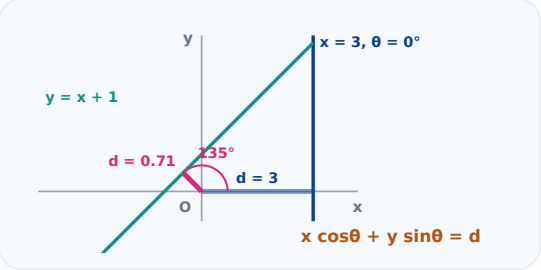


Slide 7 gives three edge pixels P1 (1, 2), P2 (2, 3), P3 (3, 4). **Step by step** (slide 8): 1. slope $m = (3 - 2)/(2 - 1) = \mathbf{1}$. 2. intercept $c = y - mx = 2 - 1 = \mathbf{1}$. 3. check P3: $3 + 1 = 4$ ✓. So all three lie on **y = x + 1**. Observation: they share one line. But through any **single** point, infinitely many lines pass (the grey dashed ones).

y = mx + c has **two degrees of freedom** (m and c); two points pin them down (slide 9). Now take **x = 1**: it is vertical, so the slope is **infinite** and there is no y-intercept. Slope–intercept form cannot write this line, and m has no upper limit. That is the "infinite slope problem" that slide 10 sets out to avoid.

3 · Polar form: d and θ, worked for two lines

Topic 6 · slides 10–12



To avoid the infinite slope, describe a line by its **perpendicular distance d** from the origin and the **angle θ** that perpendicular makes with the x-axis (slide 10): **$x\cos\theta + y\sin\theta = d$**. Both stay finite for every line, vertical ones included.

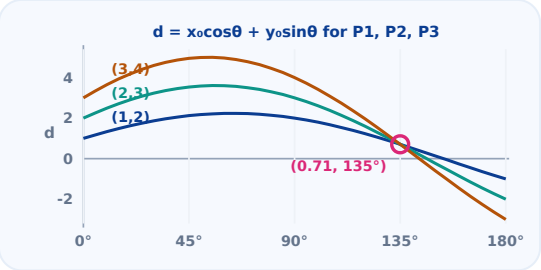
x = 3 (slide 12): the perpendicular from O runs along the x-axis, so **θ = 0°** and **d = 3**. Check: $x\cdot\cos0° + y\cdot\sin0° = x = 3$ ✓.

y = x + 1: 1. move everything to one side: $-x + y = 1$. 2. divide by $\sqrt{(-1)^2 + 1^2} = \sqrt{2}$: $-0.707x + 0.707y = 0.707$. 3. match $\cos\theta = -0.707$, $\sin\theta = 0.707$ → **θ = 135°**, **$d = 1/\sqrt{2} \approx 0.71$**. The perpendicular points up-left to (−0.5, 0.5), which is 0.71 from O.

Slide 11 writes the form as $x\cos\theta - y\sin\theta = d$ (image rows grow downwards in its picture); slides 13 and 21 use $+ y\sin\theta$, which we follow here.

4 · Image space → parameter space: one point, one sinusoid

Topic 6 · slides 13–16



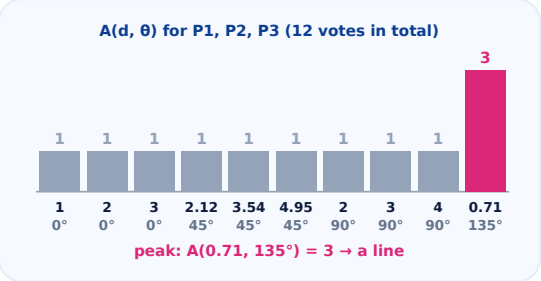
The core idea (slide 13): convert **image space** into **parameter space** (d, θ). For a fixed point (x_0, y_0), every line through it satisfies $\mathbf{x_0 \cos\theta + y_0 \sin\theta = d}$. Letting θ run gives one curve, a **sinusoid** (slides 14–15).

Worked for slide 7's points at θ = 0°, 45°, 90°, 135°: P1 (1, 2) → d = 1, 2.12, 2, **0.71**. P2 (2, 3) → 2, 3.54, 3, **0.71**. P3 (3, 4) → 3, 4.95, 4, **0.71**. At θ = 0°–90° the three values differ; at θ = 135° they agree.

So the three curves **intersect** at (d, θ) = (0.71, 135°), which is exactly the polar form of y = x + 1 from part 3. That is slide 16: points on the same straight line give curves that meet at a common location in parameter space.

5 · The accumulator: counting votes instead of finding curves

Topic 6 · slides 17–19



We know one point gives a sine curve and points on a line give curves that intersect, but it is **infeasible** to draw infinitely many curves and look for exact intersections (slide 17).

Instead keep an **accumulator**: a **2-D array A(d, θ)** that records how many edge points support each (d, θ) combination, the number of **votes** it has received (slide 18). **Voting** (slide 19): each edge pixel casts votes; each vote stands for one line through that pixel; combinations with many votes are likely real lines.

Worked with P1–P3 and 4 angles: 3 points × 4 angles = 12 votes. Nine (d, θ) cells get 1 vote each; the cell (0.71, 135°) gets **3**. The peak is the line y = x + 1.

added In practice d is also cut into bins (e.g. whole numbers), so near-equal d values share a cell.

6 · The slide example, step by step

Topic 6 · slides 20–24

slide 20 image

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

edge pixels (1,1)…(4,4)

(x, y)	θ = 0°	45°	90°	135°
(1,1)	1	1.41	1	0
(2,2)	2	2.83	2	0
(3,3)	3	4.24	3	0
(4,4)	4	5.66	4	0

every other cell: 1 vote

A(0, 135°) = 4

back to image:

x cos135° + y sin135° = 0

−0.707x + 0.707y = 0

y = x

Slide 20: a 5 × 5 binary edge image, 1 = edge pixel, 0 = background, with edges at (1,1), (2,2), (3,3), (4,4). Use **x cosθ + y sinθ = d** with θ = 0°, 45°, 90°, 135° (slide 21; real systems use a much finer angle step).

Step by step (slides 22–24): 1. take the edge pixels and their coordinates. 2. for each pixel and each θ compute d: (2,2) at 45° → 2(0.707) + 2(0.707) = **2.83**; (2,2) at 135° → 2(−0.707) + 2(0.707) = **0**. 3. vote: 16 votes in all. 4. find the peak: d = 0 at 135° for all four pixels, so **A(0, 135°) = 4**; every other cell has 1. 5. retrieve (d, θ) = (0, 135°). 6. rebuild the line (slide 23): −0.707x + 0.707y = 0 → **y = x**, the diagonal through every edge pixel.

Slide 24 lists the same seven steps: edge pixels → coordinates → sinusoids → vote → peak → (d, θ) → line in image space.

7 · Reading peaks: max(A) and the threshold T

Topic 6 · slides 25–28

slide 25: A

0	1	0	0
1	0	1	0
0	0	0	3
0	1	0	0

max(A) = 3

threshold: A(d, θ) ≥ T

T = 1 → 5 candidates

T = 2 → 1 candidate (the 3)

T = 4 → none

real images add votes from: noise · several lines · broken edges · nearby lines · false edges

The accumulator is a matrix of vote counts and **max(A)** is the strongest line candidate (slide 25). In slide 25's A, the single **3** (row 3, column 4) wins; every other cell holds 0 or 1.

Real images are not that clean: **noise**, **multiple lines**, **broken edges**, **nearby lines** and **false edges** all add or split votes (slide 26). So set a minimum threshold **T** and keep every cell with **A(d, θ) ≥ T** as a candidate line (slide 27). On slide 25's A: T = 1 keeps 5 cells, T = 2 keeps only the 3, T = 4 keeps nothing. **T is not universally appropriate**: too low gives false lines, too high misses real ones.

Extensions (slide 28): extending the transformation to **circles**, and the **Probabilistic Hough Transformation**. The slides only name these.

8 · Exam traps

Topic 6 · slides 9–13, 16, 18, 25, 27 added

- **Why polar?** Because y = mx + c cannot hold a vertical line (infinite slope), not because polar is "more accurate" (slides 9–10).
- **d and θ are about the perpendicular** from the origin to the line, not the line's own slope angle (slide 10). x = 3 has θ = 0°, not 90°.
- **Sign convention:** slide 11 has x cosθ − y sinθ = d; slides 13 and 21 use + y sinθ. Pick one and keep it for the whole answer.
- **One point → one curve** (all lines through it); **one line → one point** (d, θ) where the curves meet (slides 13, 16).
- **A(d, θ) holds vote counts**, not pixel intensities (slide 18). The answer is the (d, θ) of the peak, then turned back into a line.
- **Threshold T** is a minimum vote count, A ≥ T, and no single T suits every image (slide 27).

9 · Quick check

Topic 6 · slides 12–13, 21, 25 added

▼ **Q1.** Give θ and d for the line y = 4.

The perpendicular from the origin is along the y-axis: θ = 90°, d = 4. Check: x·cos90° + y·sin90° = y = 4 (slides 12–13).

▼ **Q2.** Point (3, 3): what is d at θ = 45°?

d = 3 cos45° + 3 sin45° = 3(0.707) + 3(0.707) = 4.24 (slide 21 formula).

▼ **Q3.** An accumulator has values 0, 2, 5, 1, 4, 2. With T = 3, how many candidate lines?

Two: the cells with 5 and 4 are ≥ 3 (slide 27). The strongest is 5 = max(A) (slide 25).

Key terms

local vs global structure · slope–intercept problem · polar / normal form · x cosθ + y sinθ = d · perpendicular distance d · angle θ · image space · parameter (Hough) space · sinusoid · intersection · accumulator A(d, θ) · voting · peak · threshold T · angle resolution · circle Hough · probabilistic Hough

Exam relevance

Try each question yourself first, then reveal the answer.

Q1. "What problem does the Hough transform solve that an edge detector does not?"

Answer: An edge detector finds **local** intensity changes: which pixels are likely to lie on boundaries. It does not say whether several edge pixels belong to the **same line** or shape, whether a line is present **despite gaps, noise or partial occlusion**, or whether an incomplete boundary belongs to a circle. The Hough transform finds **global geometric structure** (lines, circles) from a collection of local edge points, by voting.

Q2. "Why does the Hough transform use the polar representation of a line? Give the polar form of x = 3 and y = x + 1."

Answer: Slope–intercept y = mx + c cannot represent **vertical lines** (m → ∞), and m is unbounded, so it can't be put into a finite accumulator. The **polar form x cosθ + y sinθ = d** uses the perpendicular distance d from the origin and the angle θ of that perpendicular to the x-axis. Both are bounded, so every line fits. **x = 3** → θ = 0°, d = 3. **y = x + 1** → −x + y = 1; dividing by √2 gives x cos135° + y sin135° = 1/√2, so **θ = 135°, d ≈ 0.71**.

Q3. "Explain the mapping from image space to parameter space and why points on a line meet at one point."

Answer: For a fixed edge point (x_0, y_0), every line through it satisfies $x_0 \cos\theta + y_0 \sin\theta = d$. As θ varies, this traces a **sinusoid** in (θ, d) space, which stands for *all* lines through that point. A particular line (d*, θ*) passes through all its points, so every one of their sinusoids goes through (θ*, d*). **Points on the same line give curves that intersect at a common location**, and that location is the line's parameters.

Q4. "Given edge pixels (1,1), (2,2), (3,3), (4,4) and θ ∈ {0°, 45°, 90°, 135°}, use a Hough accumulator to find the line."

Answer: d = x cosθ + y sinθ. θ = 0°: d = 1, 2, 3, 4 (one vote each). θ = 45°: d = 1.41, 2.83, 4.24, 5.66 (one each). θ = 90°: d = 1, 2, 3, 4 (one each). θ = 135°: d = 0 for all four, so **A(0, 135°) = 4**, the peak. The line is x cos135° + y sin135° = 0 → −x + y = 0 → **y = x**.

Q5. "What is the Hough accumulator? How are peaks chosen in real images, and what extensions exist?"

Answer: The accumulator **A(d, θ)** is a 2-D array that counts how many edge points support each discretised (d, θ) pair. Each edge pixel votes once per θ (one candidate line through it). max(A) is the strongest line, but real images have **noise, several lines, broken edges, nearby lines and false edges**, so a **threshold T** is applied (A ≥ T → candidate line). T is not universally right and depends on the image and line length. **Extensions:** circles (parameters a, b, r, a 3-D accumulator) and the **probabilistic Hough transform** (a random subset of points: faster, and gives segments).

Practice: 10 questions

Exam-style questions on Topic 6: calculate, apply, explain, classify, outline, describe, compare. Answer on paper first, then reveal. Each answer cites its slides; **added** marks anything that goes beyond the slides. "Show all answers" above opens these too.

P1. CALCULATE For the edge point (2, 1), compute d = x cosθ + y sinθ at θ = 0°, 45°, 90° and 135°.

θ	0°	45°	90°	135°
d	2	2.12	1	−0.71

point (2, 1): x cosθ + y sinθ
d < 0 at 135°: the perpendicular points the other way

Answer · Topic 6 slides 13, 21**:** θ = 0°: 2(1) + 1(0) = **2**. θ = 45°: 2(0.707) + 1(0.707) = **2.12**. θ = 90°: 2(0) + 1(1) = **1**. θ = 135°: 2(−0.707) + 1(0.707) = **−0.71**. These four (d, θ) pairs are the four votes this pixel casts (slide 19). **added** A negative d just means the perpendicular from the origin points in the opposite direction.

P2. **APPLY** Rebuild the image-space line from each accumulator peak: (a) θ = 0°, d = 4 (b) θ = 90°, d = 2 (c) θ = 45°, d = 1.41.



Answer · Topic 6 slides 23–24**:** Put the values into x cosθ + y sinθ = d. (a) cos0° = 1, sin0° = 0 → **x = 4** (vertical). (b) cos90° = 0, sin90° = 1 → **y = 2** (horizontal). (c) 0.707x + 0.707y = 1.41 → divide by 0.707 → **x + y = 2**. This is step 7 of slide 24: reconstruct the line in image space.

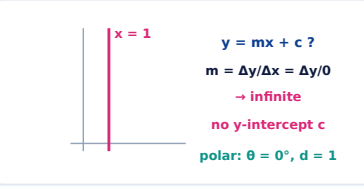
P3. **CALCULATE** Edge pixels (2, 0), (2, 1) and (2, 3). Using θ = 0°, 45°, 90°, 135°, fill the d table, find the accumulator peak and the line.

(x, y)	0°	45°	90°	135°
(2,0)	2	1.41	0	−1.41
(2,1)	2	2.12	1	−0.71
(2,3)	2	3.54	3	0.71

peak A(2, 0°) = 3 → x = 2

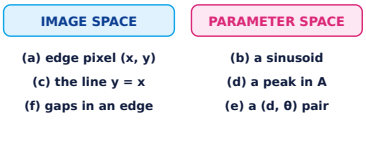
Answer · Topic 6 slides 20–23**:** θ = 0°: d = 2, 2, 2. θ = 45°: 1.41, 2.12, 3.54. θ = 90°: 0, 1, 3. θ = 135°: −1.41, −0.71, 0.71. Only (d = 2, θ = 0°) appears three times, so **A(2, 0°) = 3** is the peak. Line: x cos0° + y sin0° = 2 → **x = 2**, a vertical line, which y = mx + c could not even write (slide 9).

P4. **EXPLAIN** Explain the problem with slope–intercept form for the line x = 1, and how the polar representation removes it.



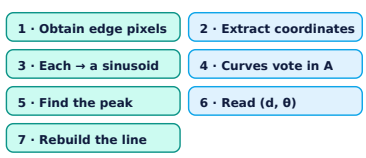
Answer · Topic 6 slides 9–11**:** Slope–intercept y = mx + c uses two degrees of freedom, m and c. On x = 1 every point has the same x, so m = Δy/Δx divides by zero: the slope is **infinite**, and the line never crosses the y-axis, so there is **no c** (slide 9). The **polar form** uses the perpendicular distance d from the origin and its angle θ to the x-axis (slide 10), which are finite for every line: x = 1 is simply **θ = 0°, d = 1**.

P5. **CLASSIFY** Image space or parameter space? (a) an edge pixel (x, y) (b) the sinusoid traced by one point (c) the line y = x (d) a peak in the accumulator (e) a (d, θ) pair (f) the gaps in a broken edge.



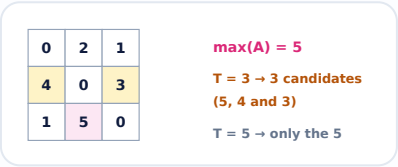
Answer · Topic 6 slides 13–18**:** **Image space:** (a), (c), (f): things you can see in the picture. **Parameter space:** (b), (d), (e): one point becomes a curve of all (d, θ) lines through it, and the accumulator A(d, θ) lives here too. The Hough transform goes image → parameter (vote) and then back to image (draw the line) (slides 13, 24).

P6. **OUTLINE** These steps are shuffled: find the peak; transform each coordinate into a sinusoid; obtain edge pixels; reconstruct the line; extract coordinates; retrieve (d, θ); let the curves vote. Put them in order.



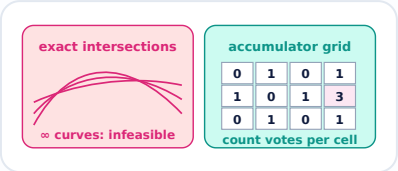
Answer · Topic 6 slide 24**:** 1. **Obtain edge pixels**. 2. **Extract their coordinates**. 3. **Transform each coordinate into a sinusoidal curve** in parameter space. 4. **Let the curves vote** in an accumulator. 5. **Identify the accumulator peak**. 6. **Retrieve the corresponding (d, θ)**. 7. **Reconstruct the line** in image space. Memory aid: edges → numbers → curves → votes → peak → parameters → line.

P7. **CALCULATE** An accumulator is A = (0, 2, 1 / 4, 0, 3 / 1, 5, 0). What is the strongest candidate? How many candidate lines are there with T = 3, and with T = 5?



Answer · Topic 6 slides 25, 27**:** The strongest candidate is **max(A) = 5** (row 3, column 2) (slide 25). With **T = 3**, keep cells with A ≥ 3: 4, 3 and 5, so **3 candidates** (slide 27). With **T = 5** only the 5 is left: **1 candidate**. Lowering T admits weaker lines (and more false ones); raising it may drop real lines.

P8. **EXPLAIN** Explain why the Hough transform uses an accumulator of votes rather than searching for the exact intersections of the curves, and what the angle step on slide 21 has to do with it.



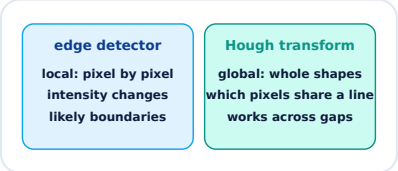
Answer · Topic 6 slides 17–19, 21**:** Every edge point gives a whole curve, and a line is where curves cross. Generating **infinitely many curves** and finding their **exact intersections** is infeasible (slide 17). So (d, θ) is turned into a finite **2-D array A(d, θ)**: each edge pixel votes for the cells of the lines through it, and cells with many votes are likely lines (slides 18–19). The θ values checked are the columns of that array: the example uses 0°, 45°, 90°, 135°, while real systems use a **much finer** angle resolution (slide 21).

P9. **DESCRIBE** Describe why picking just max(A) is not enough on real images, and how the threshold T is used.



Answer · Topic 6 slides 25–27**:** max(A) is the strongest single candidate (slide 25), but a real image may hold **several lines**, and **noise**, **broken edges**, **nearby lines** and **false edges** add stray votes or split a line's votes (slide 26). So a minimum threshold is set: every cell with **A(d, θ) ≥ T** is a candidate line (slide 27). **T is not universally appropriate**: it has to be chosen for the image.

P10. **COMPARE** Compare what an edge detector gives you with what the Hough transform gives you. Why do you need both?



Answer · Topic 6 slides 3–6, 13, 24**:** An **edge detector** is **local**: it marks where intensity changes sharply and which pixels are likely on boundaries (slide 3). It cannot say which pixels belong to the **same line** or whether a line continues across **gaps, noise or occlusion** (slide 5). The **Hough transform** is **global**: it maps those edge pixels into parameter space and votes, so a line supported by many pixels wins even if it is broken (slides 13, 19). You need both: Hough step 1 is "obtain edge pixels" (slide 24).

Quick MCQs: check your understanding

M1. Which of these does an edge detector tell us? **CHOOSE ALL THAT APPLY**

A. Where significant intensity changes occur
B. Which pixels are likely to belong to object boundaries
C. Whether several edge pixels belong to the same line
D. Where the image contains local structural information

M2. P1 (1, 2), P2 (2, 3), P3 (3, 4) all lie on... **CHOOSE 1**

A. $y = x$
B. $y = x + 1$
C. $y = 2x$
D. $x = 1$

M3. Why is $y = mx + c$ a problem for the line $x = 1$? **CHOOSE 1**

A. It has a slope of 0
B. Its slope is infinite
C. It passes through the origin
D. It needs three parameters

M4. In the polar form, d and θ describe... **CHOOSE 1**

A. the line's slope and intercept
B. the perpendicular distance from the origin and the angle of that perpendicular to the x-axis
C. the length of the line and its angle
D. the pixel's intensity and orientation

M5. The polar form of $x = 3$ is... **CHOOSE 1**

A. $\theta = 90°, d = 3$
B. $\theta = 0°, d = 3$
C. $\theta = 45°, d = 3$
D. $\theta = 0°, d = 0$

M6. In parameter space, a single edge point (x0, y0) becomes... **CHOOSE 1**

A. a single point
B. a straight line
C. a sinusoidal curve
D. a circle

M7. Points on the same straight line give curves that... **CHOOSE 1**

A. are parallel
B. intersect at a common location in parameter space
C. never meet
D. all have $d = 0$

M8. Which are true of the Hough accumulator? **CHOOSE ALL THAT APPLY**

A. It is a two-dimensional array
B. $A(d, \theta)$ is the number of votes for that combination
C. Each edge pixel casts votes for lines through it
D. It stores the grey level of each pixel

M9. Why not just find the exact intersections of the curves? **CHOOSE 1**

A. The curves never intersect
B. Generating infinitely many curves and searching for exact intersections is infeasible
C. Intersections only exist for circles
D. The polar form forbids it

M10. Point (1, 1) at $\theta = 135°$: $d = x\cos\theta + y\sin\theta =$ **CHOOSE 1**

A. 1.41
B. 0
C. 1
D. -1.41

M11. In the slide example the peak A(0, 135°) = 4 gives which line? **CHOOSE 1**

A. $y = x$
B. $x = 0$
C. $y = 0$
D. $x + y = 4$

M12. In slide 25's matrix A, max(A) is... **CHOOSE 1**

A. 1
B. 3
C. 4
D. the sum of all votes

M13. Which can make real accumulators messy, so a threshold is needed? **CHOOSE ALL THAT APPLY**

A. Noise
B. Broken edges
C. Nearby lines
D. Using the polar form

M14. Which extensions of the Hough transform do the slides name? **CHOOSE ALL THAT APPLY**

A. Extending it to circles
B. The Probabilistic Hough Transformation
C. Histogram equalisation
D. The median filter

Answer key

1. **M1: A, B, D**. Edges are local. Whether pixels share one line is exactly what they do NOT say, which is the problem Hough solves. **Topic 6 · slides 3, 5**
2. **M2: B**. $m = (3 - 2)/(2 - 1) = 1$ and $c = 2 - 1 = 1$, so $y = x + 1$. **Topic 6 · slides 7–8**
3. **M3: B**. A vertical line has infinite slope and no y-intercept, so slope–intercept form cannot represent it. **Topic 6 · slides 9–10**
4. **M4: B**. d = perpendicular distance from the origin to the line; θ = angle the perpendicular makes with the x-axis. **Topic 6 · slides 10–11**
5. **M5: B**. The perpendicular from the origin lies along the x-axis (θ = 0°) and is 3 long. **Topic 6 · slide 12**
6. **M6: C**. $x0 \cos\theta + y0 \sin\theta = d$ traces a sine-shaped curve as θ varies: all lines through the point. **Topic 6 · slides 13–14, 17**
7. **M7: B**. They meet at one (d, θ), and that location is the line's parameters. **Topic 6 · slide 16**
8. **M8: A, B, C**. It counts votes per (d, θ); it does not store intensities. **Topic 6 · slides 18–19**
9. **M9: B**. So the parameter space is discretised into an accumulator and votes are counted. **Topic 6 · slide 17**
10. **M10: B**. $1(-0.707) + 1(0.707) = 0$. That is why all diagonal pixels in the slide example vote for (0, 135°). **Topic 6 · slides 20–22**
11. **M11: A**. $x \cos135° + y \sin135° = 0 \rightarrow -x + y = 0 \rightarrow y = x$. **Topic 6 · slides 22–23**
12. **M12: B**. The single 3 is the largest count, so it is the strongest line candidate. **Topic 6 · slide 25**
13. **M13: A, B, C**. Slide 26 lists noise, multiple lines, broken edges, nearby lines and false edges. The polar form is the fix for vertical lines, not a cause of mess. **Topic 6 · slides 26–27**
14. **M14: A, B**. Slide 28 names circles and the probabilistic Hough transformation. The other two are from earlier topics. **Topic 6 · slide 28**