

Keypoint Detection

Keypoint Detection



Antonio Carta

2026-09-30

University of Pisa

antonio.cart@unipi.it

Motivation



- Learn how to find the **same physical point** in different images of a scene
- Two sub-problems:
 - **Detection**: where are the good points?
 - **Description**: how do we tell them apart/match them?

Matching and Correspondence



Figure 1: What kind of features can we use to recognize correspondences between two images? Why is this a hard problem? Image source: Szelisky

- Given two (or more) images of the same scene, find pairs of pixels (p_1, p_2) that are projections of the **same 3D point**
- Hard because of:
 - viewpoint change (perspective distortion)
 - scale change (zoom, distance)
 - illumination change
 - occlusion and noise
- Approach: instead of matching every pixel, select a sparse set of **distinctive keypoints**, then match those

- **Image stitching / panoramas**: align overlapping images via corresponding points
- **Structure from Motion (SfM)**: recover 3D structure and camera poses from correspondences across many views
- **Tracking**: follow the same point across video frames
- **Stereo / depth estimation**: match points across a calibrated camera pair

All of these tasks reduce to: detect repeatable points, describe them, match them reliably

To detect **stable keypoints** we need invariance or equivariance:

- **equivariance**: detected locations transform with the image
- **invariance**: descriptor value remains stable after normalization

we know already several tools:

- **scale**: multi-scale representation
- **shift**: LSI systems (convolutions)
- **orientation**: Today with SIFT

Note: the feature descriptor needs to be invariant/equivariant

Detector

- find keypoint locations (and scale/orientation)
- e.g. Harris, DoG extrema
- detectors need to be equivariant to view changes

Descriptor

- encode local appearance around a keypoint into a vector
- SIFT's 128-D histogram
- descriptors need to be invariant to view changes

Classical vs. learned

- hand-crafted: Harris, SIFT
- learned: SuperPoint, LightGlue

- What is a Keypoint
- Corner Detection
- SIFT: Blob Detection
- Scale-Invariance and Scale-Space
- SIFT: Detection and Localization
- SIFT: Orientation and Descriptor



Figure 2: source: opencv.org

What is a Keypoint



- **Repeatable**: the point is detected under different views/conditions
- **Invariant**: to scale, rotation, illumination, and (ideally) some viewpoint change
- **Local**: defined from a small image neighborhood (robust to occlusion/clutter elsewhere)
- **Distinctive**: the local appearance should be informative enough to disambiguate it from other points
- **Efficient**: detection and description should be cheap enough to run on many points/images

- **Repeatability**: if the same physical point is visible in two images, does the detector fire at the corresponding pixel in both?
- This is the **first** requirement, a perfectly distinctive but irreproducible point is useless for correspondence
- Flat regions and straight edges are **not** repeatable detections: many nearby locations look identical (1D or 2D ambiguity), small noise/viewpoint shifts move the “best” point around

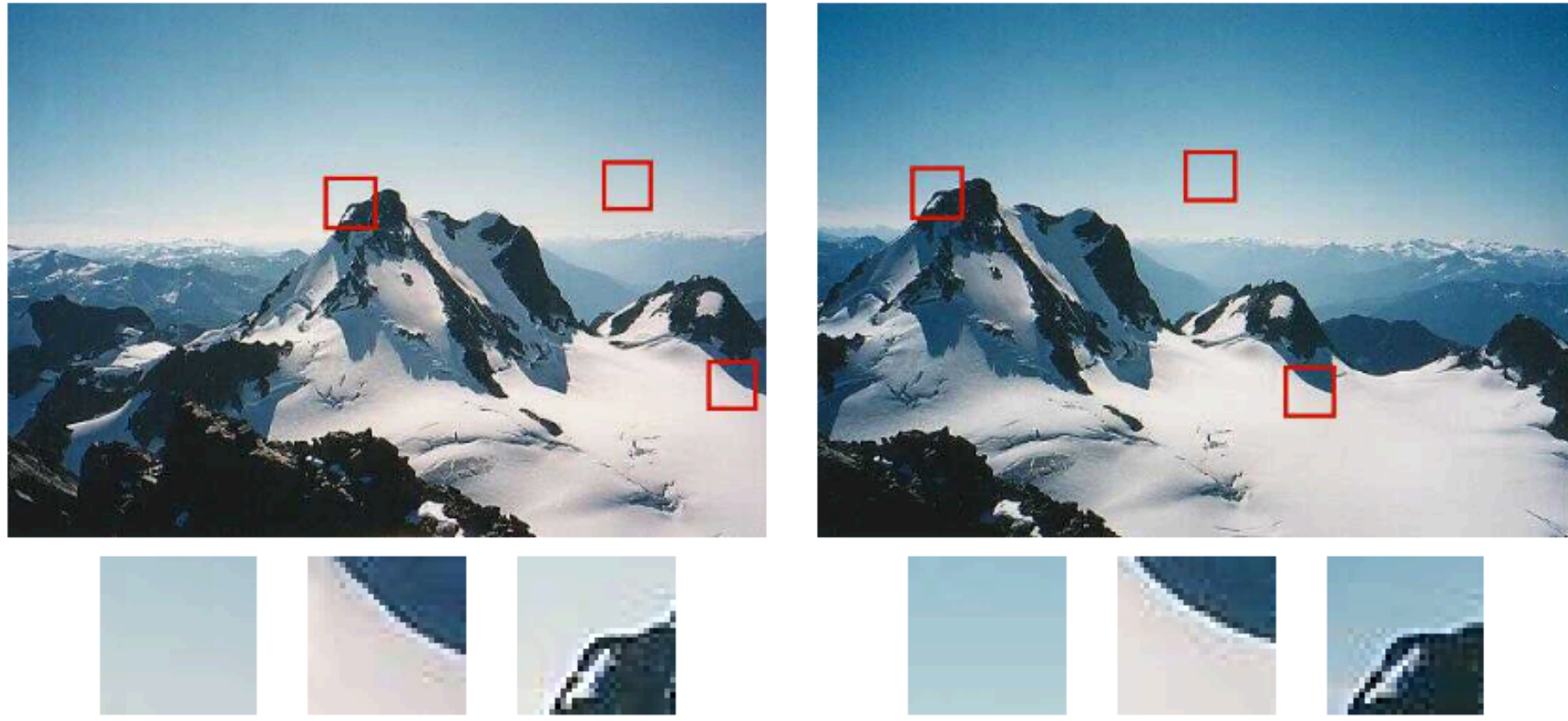


Figure 3: source: Szeliski

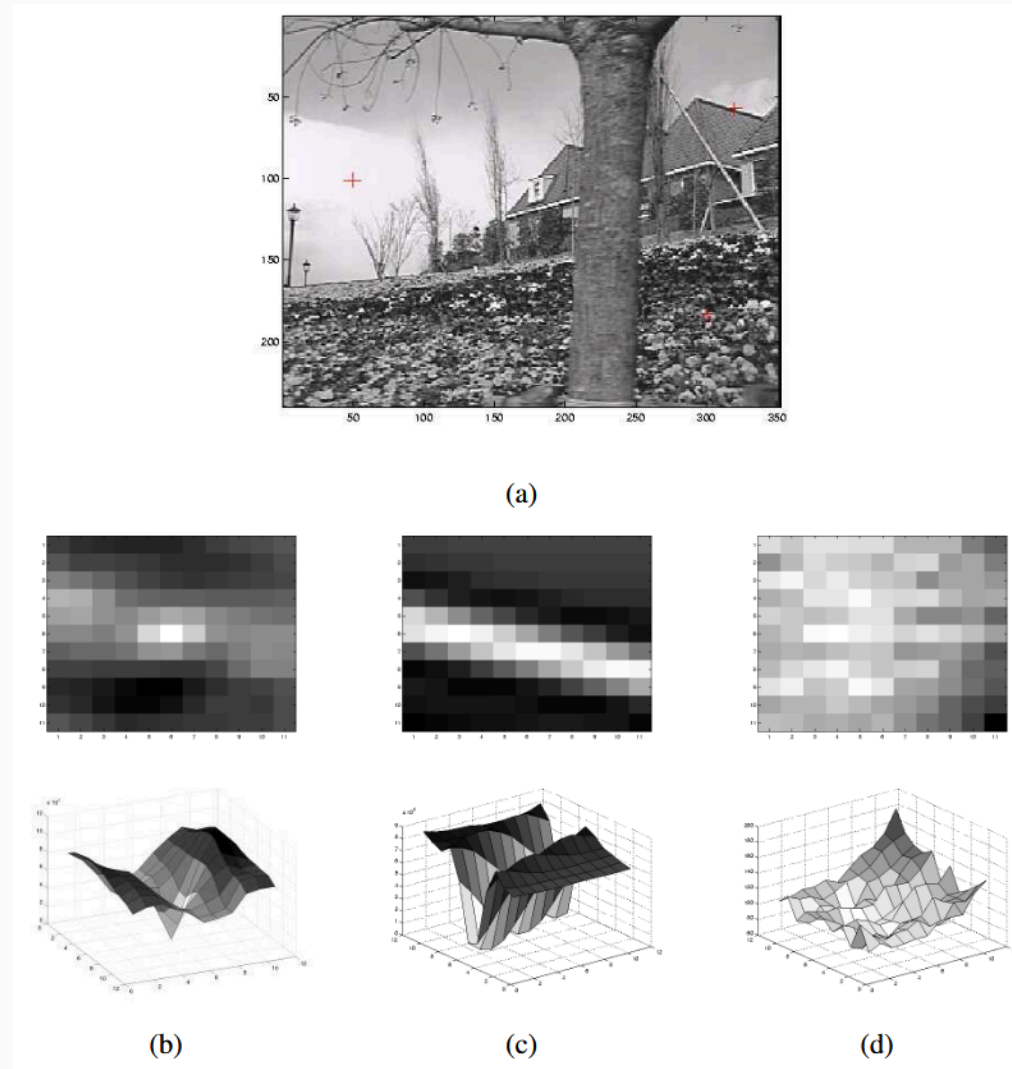


Figure 4: Autocorrelation at red crosses: (b) flower bed, (c) roof, (d) cloud. source: Szeliski

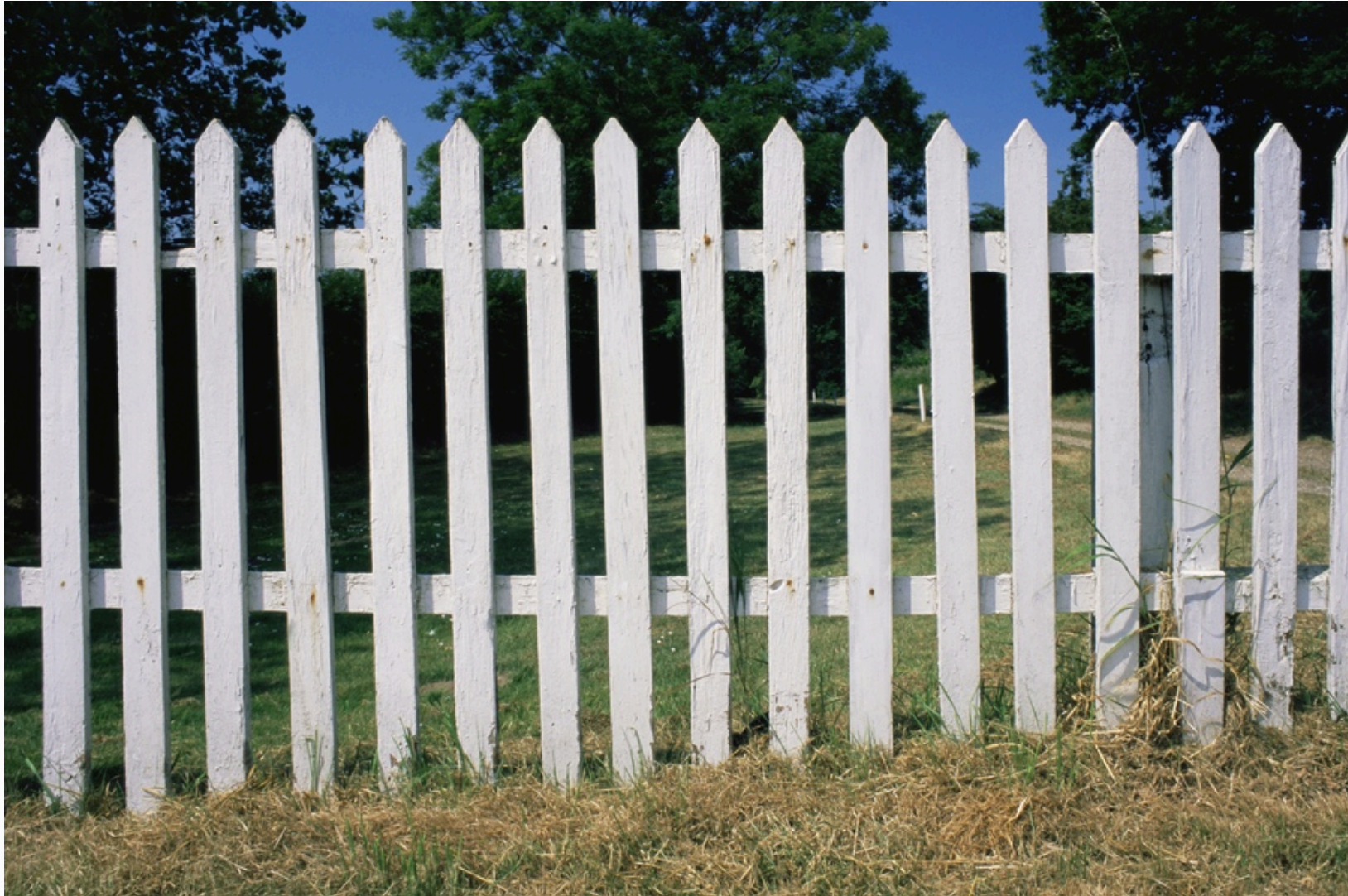


Figure 5: A point can be locally stable, but not have stable matches due to multiple similar points. source: GettyImages

Scale: an object at different distances/zoom appears at different sizes, need a detector/descriptor stable across scale changes

Rotation: in-plane camera rotation rotates local patches, the descriptor must either be invariant to rotation or compute a reference orientation to normalize against

Illumination: lighting changes shift and scale pixel intensities, gradients normalized appropriately remain comparatively stable

- **Locality**: a smaller support region is more robust to occlusion and background clutter, but contains less information
- **Distinctiveness**: a larger/more detailed descriptor is easier to match uniquely among thousands of candidates, but costs more to compute and compare
- **Computational cost**: detecting and describing thousands of points per image, then matching across image pairs, must remain tractable

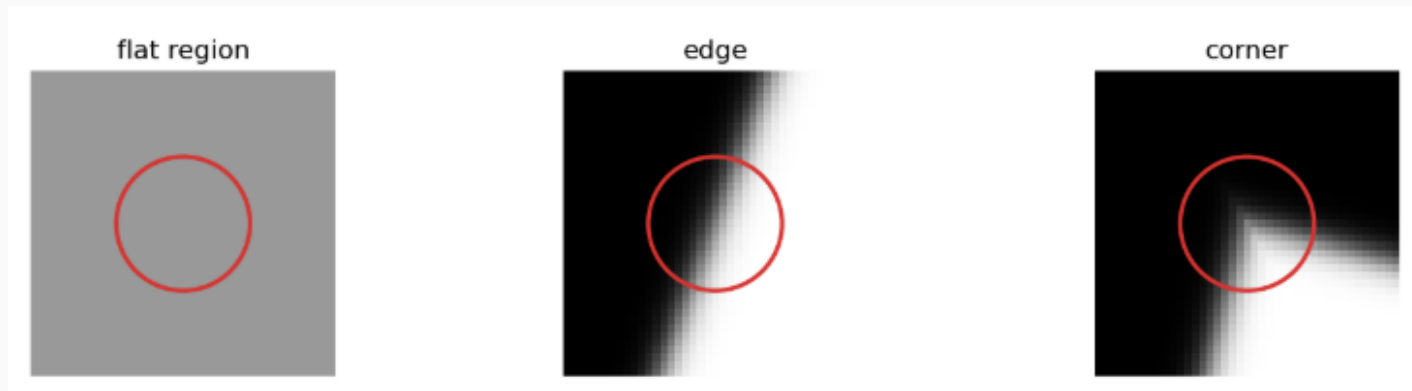
Harris Corner Detector



Why Corners: Edges vs. Corners vs. Flat Regions

Slide a small window (i.e. move the red circle in the figure) over the image and look at how the local patch changes:

- **flat region**: no change in any direction
- **edge**: no change along the edge direction, large change across it (1D ambiguity)
- **corner**: large change in **all** directions
- Corners are the only case fully constrained in 2D, hence well-localized and repeatable



1D Ambiguity: The Aperture Problem

- looking through a small “aperture” (window), a **pure edge** only constrains motion along its gradient direction, motion along the edge is invisible
- Same ambiguity in keypoint detection: an edge point looks locally identical to its neighbors along the edge direction
- **Corners** break the aperture problem by constraining on two (or more) non-parallel directions

“<https://upload.wikimedia.org/wikipedia/commons/a/ab/Barber-pole-01.gif>”



Figure 6: source:
wikimedia

A corner has

- large changes in intensity/color/texture
- this changes need to be in two different directions (if only one direction has a rapid change, we have an edge, which has 1D ambiguity)

Now, we need to find a cheap computational definition of edge using only local information.

- interpreting an image as a continuous function: large intensity shifts == large directional derivatives

the gradient

$$J(x) = \nabla I(x) = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right) (x)$$

gives us the steepest direction. Its direction is perpendicular to the local countour of the edge

- gradients can be found with any derivative kernels we have seen
- example: $[-2, -1, 0, 1, 2]$

The Sum-of-Squared-Differences Surface (SSD)

we want:

- high gradient in multiple directions
- a point must be distinguishable from its local neighbors

the sum of squared differences (SSD) measures the difference between an image I_0 displaced by u and a second image I_1

- notation: here x_i and u are 2D coordinates

when we **match** points between two images I_0 and I_1 we have

$$E_{\text{SSD}}(\Delta u) = \sum_i w(x_i) [I_1(x_i + \Delta u) - I_0(x_i)]^2$$

Annotations:

- sum over all pixel coordinates** (points to the summation symbol $\sum_i$)
- kernel window (a small gaussian or box kernel)** (points to the weight $w(x_i)$)
- two images** (points to I_1 and I_0)
- u : displacement vector** (points to Δu)

Matching two images:

$$E_{\text{SSD}(\Delta u)} = \sum_i w(x_i) [I_0(x_i + \Delta u) - I_1(x_i)]^2$$

compare two images

- problem: matching requires two images, when we select interesting points we only have one image I_0
- we use the **autocorrelation**: that is the SSD of the image with itself

Autocorrelation:

$$E_{\text{AC}(\Delta u)} = \sum_i w(x_i) [I_0(x_i + \Delta u) - I_0(x_i)]^2$$

compare two patches of I_0

Note: a corner/distinctive point has high E_{AC} for any direction Δu . We want to find an efficient criterion to evaluate this property.

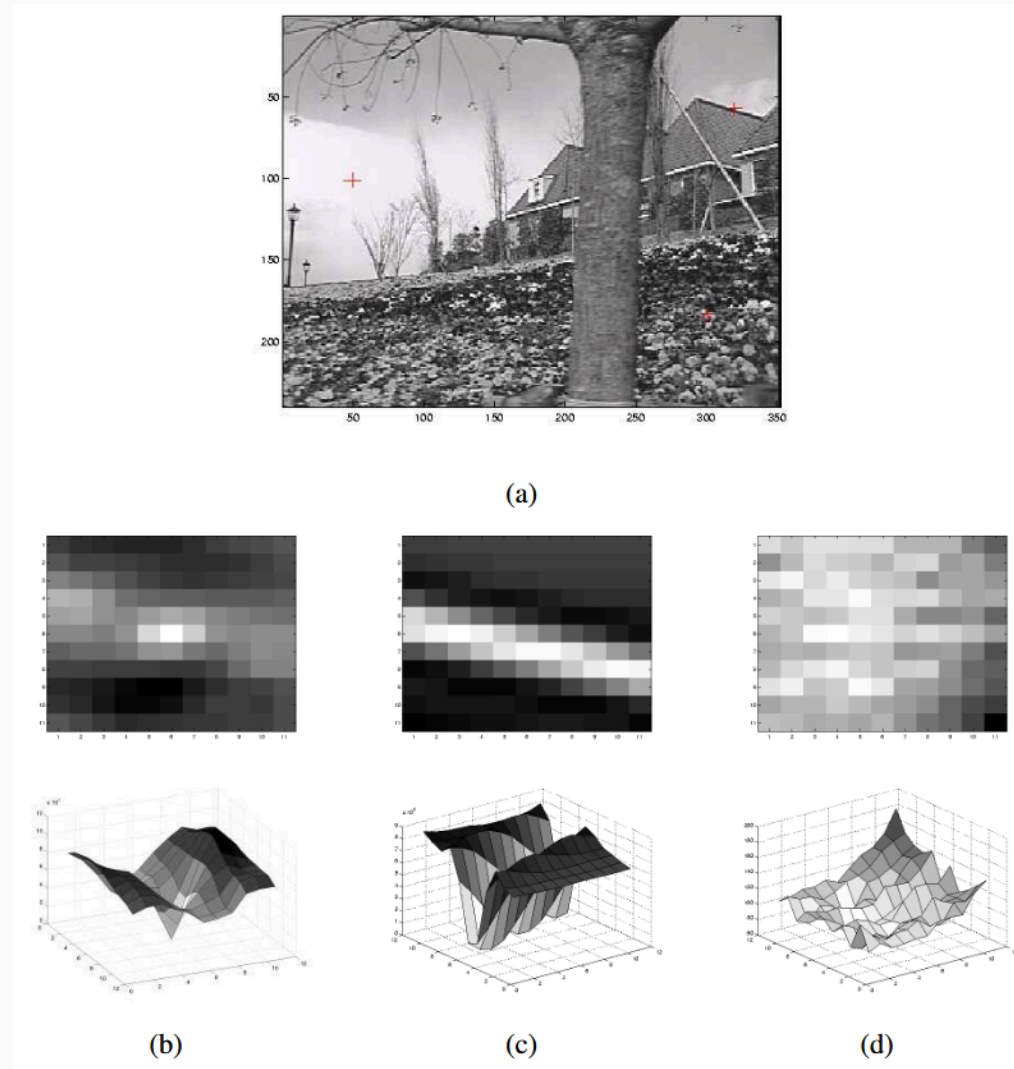


Figure 7: Autocorrelation at red crosses: (a) flower bed, (b) roof, (c) cloud. source: Szeliski

Note: a corner/distinctive point has high E_{AC} for any direction Δu . We want to find an efficient criterion to evaluate this property. *We start by taking the Taylor expansion of E_{AC}*

we can approximate $I_0(x_i + \Delta u) \approx I_0(x_i) + \nabla I(x_i)^\top \Delta u$

$$\begin{aligned} E_{AC}(\Delta u) &= \sum_i w(x_i) [I_0(x_i + \Delta u) - I_0(x_i)]^2 && \text{Taylor exp.} \\ &\approx \sum_i w(x_i) [I_0(x_i) + \nabla I(x_i)^\top \Delta u - I_0(x_i)]^2 && \text{cancel } I_0 \\ &= \sum_i w(x_i) [\nabla I(x_i)^\top \Delta u]^2 \end{aligned}$$

$$\begin{aligned} E_{AC}(\Delta u) &= \sum_i w(x_i) \left[\nabla I(x_i)^\top \Delta u \right]^2 \\ &= \Delta u^\top \mathbf{A} \Delta u \end{aligned}$$

where $\mathbf{A} = \sum_i w(x_i) \nabla I_0(x_i) \nabla I_0(x_i)^\top$

- $\mathbf{A}$ is called the 2nd moment matrix or structure tensor
- given $\nabla I_0 = \begin{pmatrix} I_{x(x_i)} \\ I_{y(x_i)} \end{pmatrix}$

$$\mathbf{A} = \sum_{x,y \in W} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

- $\mathbf{A}$ aggregates gradient information over the window w (a Gaussian or box w)
- $\mathbf{A}$ summarizes how $E_{AC}(\Delta u)$ grows for any shift direction, **without** having to try every Δu explicitly

Note: a corner/distinctive point has high E_{AC} for any direction Δu . We want to find an efficient criterion to evaluate this property. *Now we just have to figure out what properties of A guarantee this.*

Example: Building $\mathbf{A}$ from I_x, I_y on a Patch

- compute I_x, I_y with a derivative filter on a small patch
- form the three maps $I_x^2, I_y^2, I_x I_y$ pixel-wise
- sum (or Gaussian-weight) each map over the window to get the 3 distinct entries of $\mathbf{A}$
- create $\mathbf{A}$ and compute its eigenvalues and eigenvectors

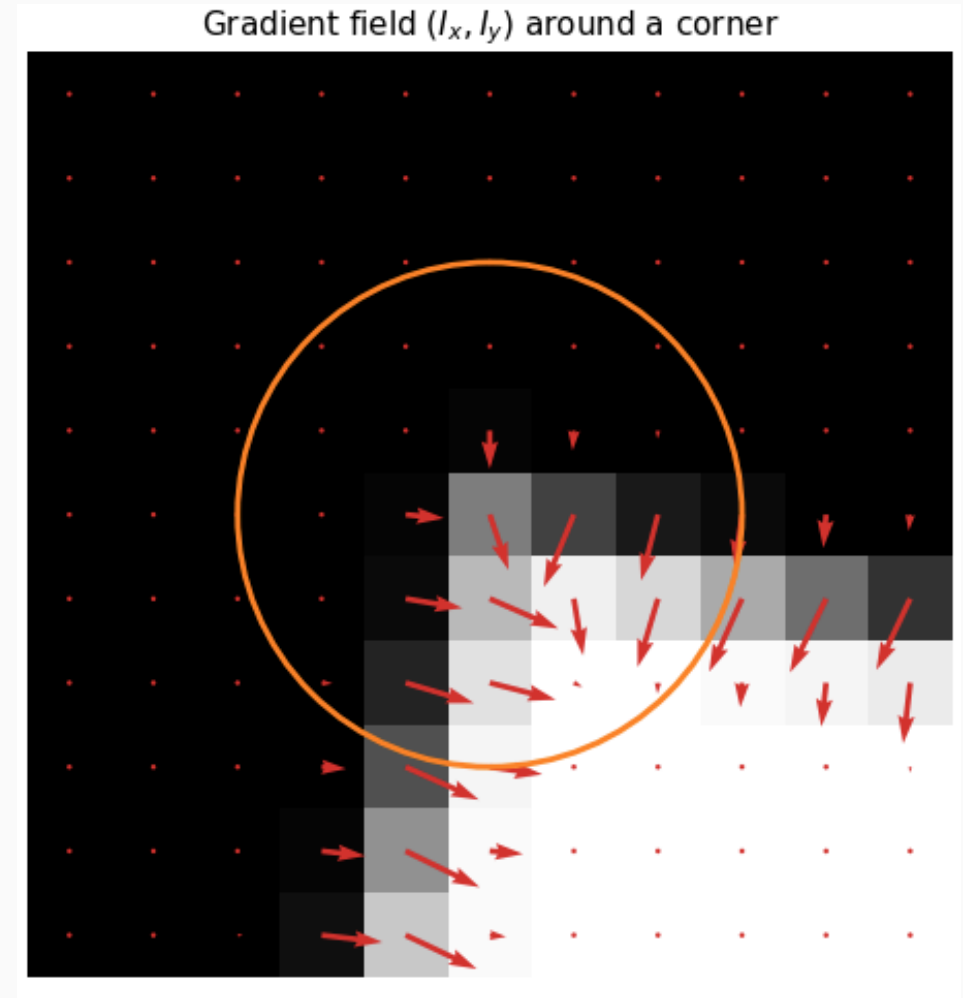


Figure 8: gradient field around a corner.

```
dI_drow, dI_dcol = np.gradient(corner_img)
Ix = dI_dcol
Iy = -dI_drow # Cartesian y points upward; array rows point downward

window_radius = 3
gauss_w = np.exp(-(xx**2 + yy**2) / (2 * (window_radius / 2) ** 2))
A = np.array([
    [np.sum(gauss_w * Ix**2), np.sum(gauss_w * Ix * Iy)],
    [np.sum(gauss_w * Ix * Iy), np.sum(gauss_w * Iy**2)],
])
```

$$E_{AC}(\Delta u) \approx \Delta u^\top \mathbf{A} \Delta u$$

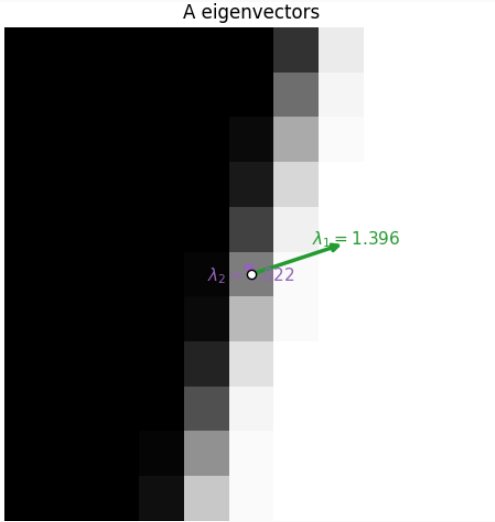
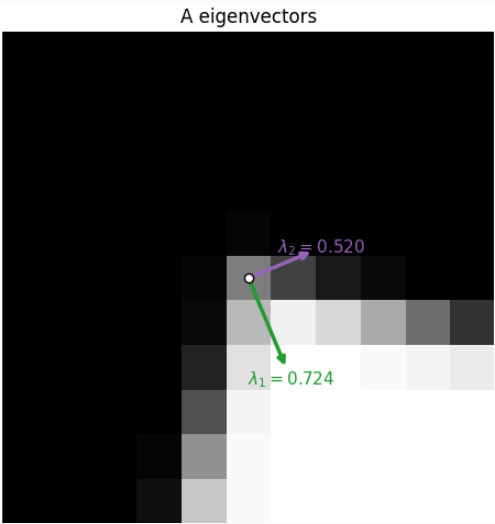
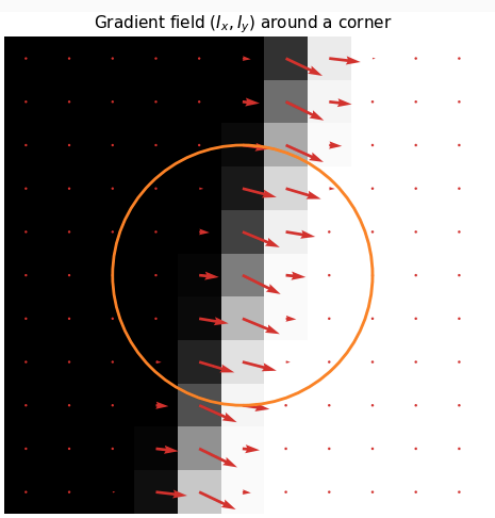
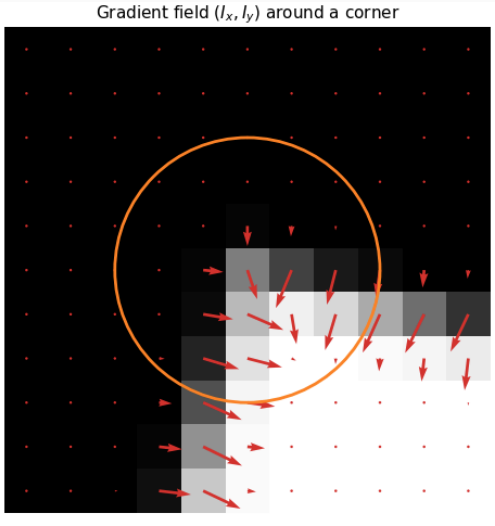
- For a real symmetric matrix such as our $\mathbf{A}$, eigenvalues λ_1, λ_2 are real, and eigenvectors $\mathbf{e}_1, \mathbf{e}_2$ are orthogonal
- $\mathbf{A}\mathbf{e}_i = \lambda_i\mathbf{e}_i$: along eigenvector directions, $\mathbf{A}$ acts as pure scaling by λ_i
- $\mathbf{A}$'s eigenvectors give the directions of **minimum** and **maximum** local intensity change

- $E_{AC}(\Delta u') \approx \lambda_1 u_x'^2 + \lambda_2 u_y'^2$ in the rotated frame aligned with the eigenvectors (u_x', u_y')
- Large λ_1, λ_2 (both directions): a sharp, well-localized 2D structure $\rightarrow$ **corner**
- One large, one small: structure changes in one direction only $\rightarrow$ edge
- Both small: no structure $\rightarrow$ flat region

- This classification only needs the **relative magnitudes** of λ_1, λ_2 , not their exact eigenvectors
- the “flat” region is a small square near the origin (both eigenvalues small)
- “edge” is the two thin wedges along the axes (one eigenvalue much larger than the other)
- “corner” is everything else (both eigenvalues comparably large)

Example: Corner vs Edge

FA 2018/19, SS 2019, SS 20



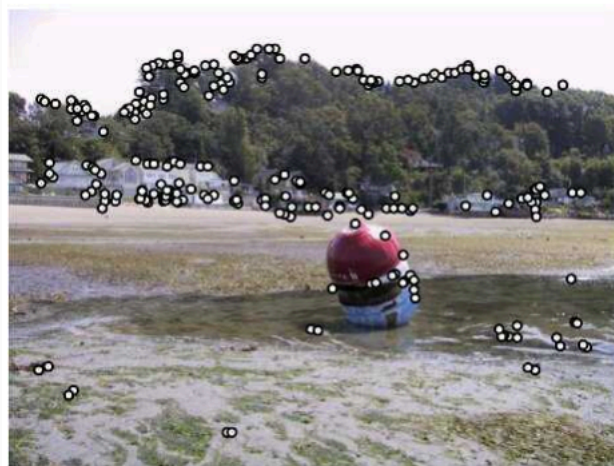
- Computing exact eigenvalues for every pixel is wasteful, Harris and Stephens (1988) propose a cheaper quantity:

$$R = \det(\mathbf{A}) - \alpha \operatorname{trace}(\mathbf{A})^2 = \lambda_1 \lambda_2 - \alpha (\lambda_1 + \lambda_2)^2$$

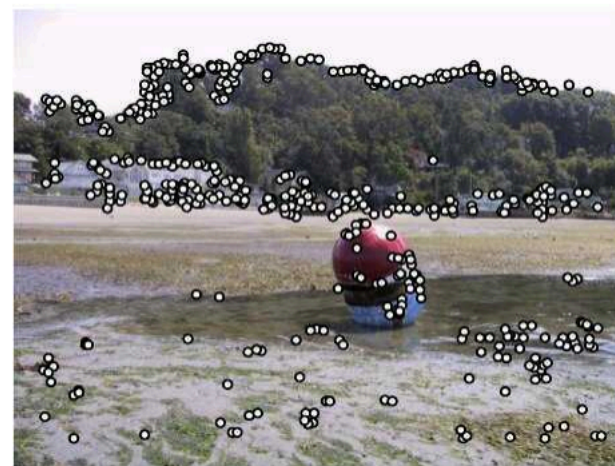
- trace and determinant are computed **directly** from the matrix entries $I_x^2, I_y^2, I_x I_y$, no eigendecomposition needed
 - $\det(\mathbf{A}) = \lambda_1 \lambda_2$ and $\operatorname{trace}(\mathbf{A}) = \lambda_1 + \lambda_2$
- R large and positive: both eigenvalues large → **corner**
- R large and negative: one eigenvalue much larger → **edge**
- $|R|$ small: both eigenvalues small → **flat**

- α is an empirical sensitivity parameter, check a few samples to set it
- α controls how strongly the response penalizes elongated (edge-like) structures relative to corner-like ones
- After computing R at every pixel, keep only points with $R > \tau$ (threshold for keypoint selection)
- τ trades off the number of detected keypoints against their quality
 - too low: many spurious/noisy corners
 - too high: miss weak but valid corners

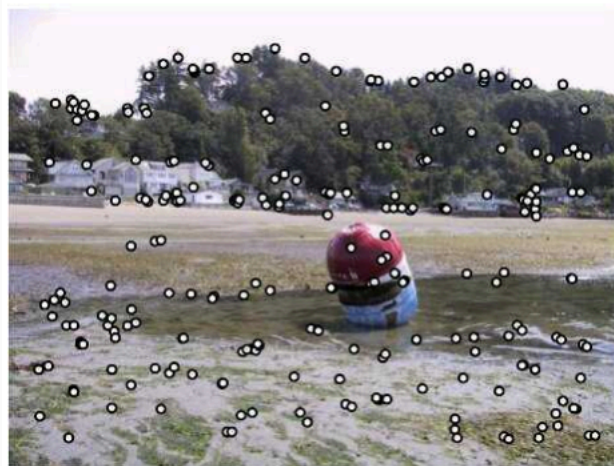
- Thresholding alone leaves **clusters** of nearby pixels all with high R (the same physical corner spans several pixels)
 - regions with high contrast will also have more keypoints
- **Adaptive Non-maximal suppression (ANMS)**: keep a point only if its R is significantly (10%) better than all the others in a small neighborhood
- Produces isolated, well-localized, and spatially distributed keypoints
- Similar ideas reappear in SIFT (scale-space extrema) and in modern detectors



(a) Strongest 250



(b) Strongest 500



(c) ANMS 250, $r = 24$



(d) ANMS 500, $r = 16$

Figure 9: source: Szeliski

Example: Harris Response

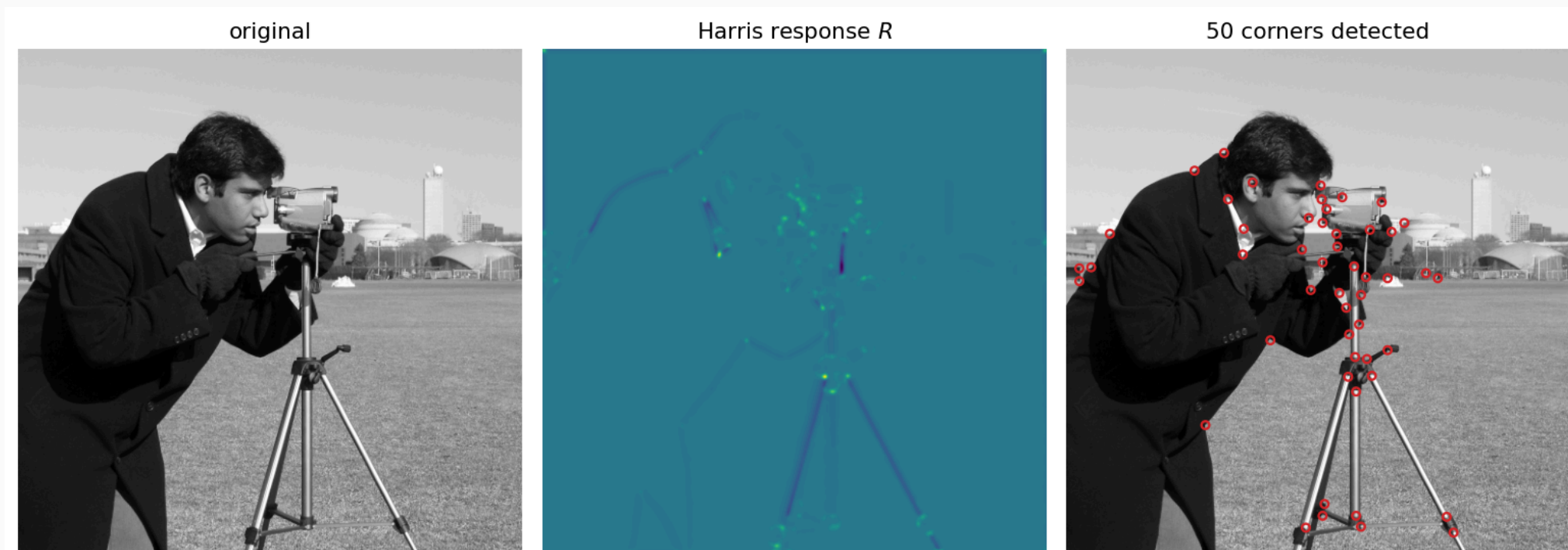


Figure 10: Harris R example

- Notice how corners concentrate at true geometric corners (camera/tripod edges, building corners, texture intersections)
- Flat regions (sky, grass) and pure edges (long straight lines) are correctly suppressed

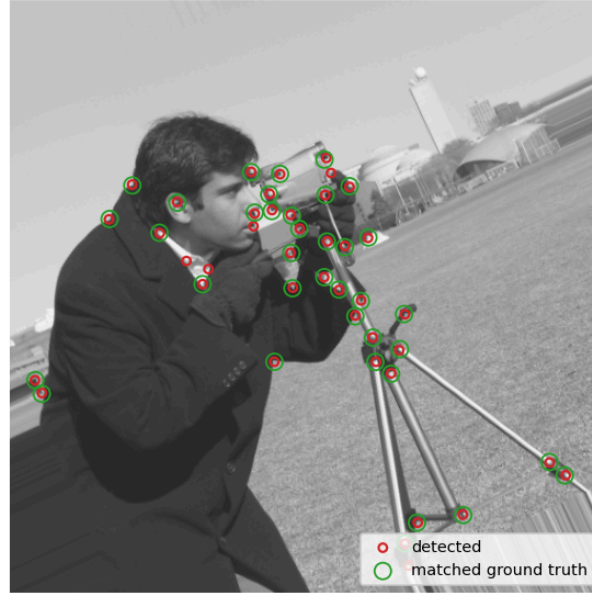
Example: Harris Under Rotation and Illumination Change

FA 17/18 17/18 17/18 17/18

original + Harris corners



rotated 25°, brightness/contrast changed
repeatability: 74% within 3px



- Same image rotated 25° around its center, with brightness and contrast changed
 - corners are re-detected from scratch on the transformed image
- Repeatability: we map the original corners through the known rotation. 74% of them have a re-detected corner

- Harris uses a **fixed** window size W , chosen once for the whole image
- **Scale**: If the same physical corner appears at a different scale (camera closer/farther), the same window size sees a different amount of structure, the response and even whether a corner is detected at all can change
- A naive fix (re-run Harris at several fixed scales) doesn't tell us which scale is the right one for a given point

We need a principled way to achieve **rotation-invariance** and select, per keypoint, a **characteristic scale** (next section)

Scale Invariance and Scale-Space Detectors

Blob Detection

- we switch from corner detection to **blob detection**
- a **blob** is an image region that has some uniform properties (color, brightness, texture), often with an elliptical shape



Fig6. Input Image

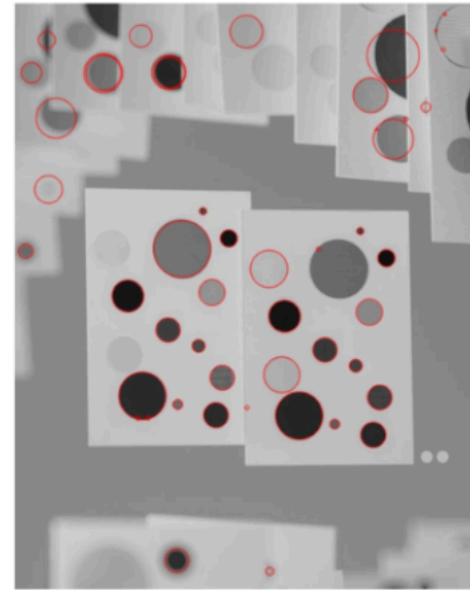


Fig7. Blob Detection

Figure 11: source: opencv.org

- **Scale-space**: the image convolved with Gaussians of increasing σ , $L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$ forms a continuous family of increasingly blurred versions of itself
 - **Gaussian pyramid**: a discretized, efficient version of scale-space representation.
Repeatedly blur and subsample by a factor of 2
 - Larger σ = coarser scale = larger structures emphasized, smaller details suppressed
- Goal of this section**: use scale-space to select the right scale per keypoint

- A “blob” or corner-like structure has an **intrinsic size** (e.g. the radius of a circular dark spot)
- If we measure response at the wrong scale (window much smaller/larger than the structure), the response is weak or unstable
 - we have the same issue with corner detection when using Harris R
 - ideal scale: window size matched to the structure size
- **Find characteristic scale**: for each keypoint, automatically find the σ^* that **maximizes a scale-dependent response**, this σ^* becomes part of the keypoint’s identity
- Crucially, σ^* should scale with the image scale: zooming in should rescale the detected σ^* proportionally

Laplacian of Gaussian (LoG)

The **Laplacian of Gaussian (LoG)**:

$$\nabla^2 G(x, y, \sigma) = \frac{\partial^2 G}{\partial x^2} + \frac{\partial^2 G}{\partial y^2}$$

- The Laplacian ($\frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$) is a measure of the 2nd order spatial derivatives
- as with any derivative-based operation for images, it suffers from noise
- so, we apply the Laplacian to the smoothed version of the image
- due to associativity, this is equivalent to applying the LoG directly

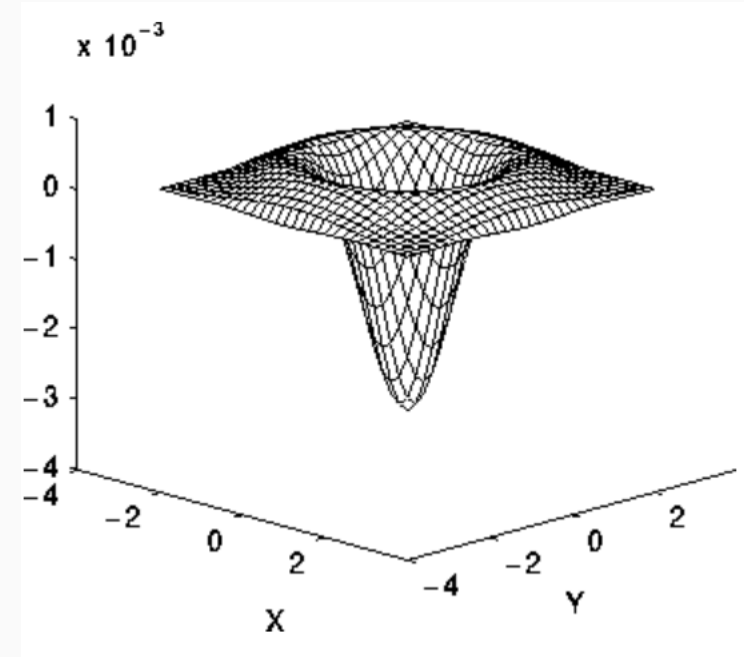


Figure 12: Laplacian of Gaussian.

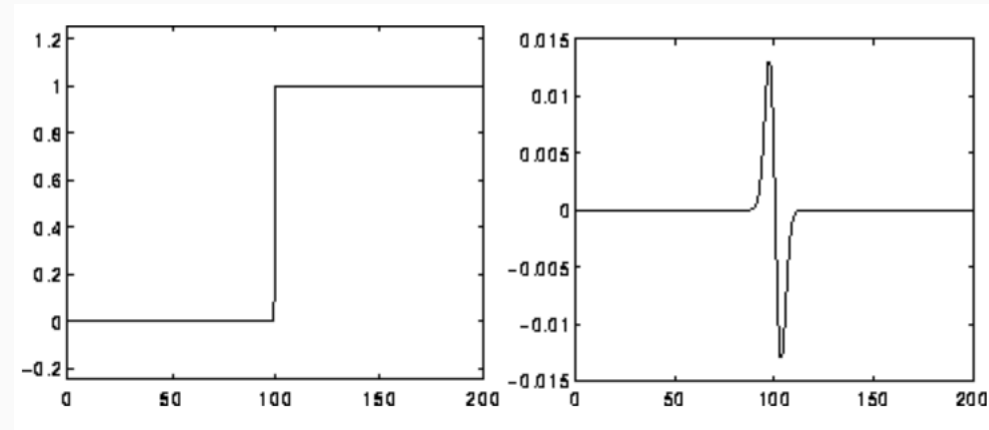


Figure 13: LoG will be around zero for constant regions. It has high response around edges.

Laplacian of Gaussian (LoG) for Scale Selection

The **Laplacian of Gaussian (LoG)**:

$$\nabla^2 G(x, y, \sigma) = \frac{\partial^2 G}{\partial x^2} + \frac{\partial^2 G}{\partial y^2}$$

$$\text{LoG}(x, y) = -\frac{1}{\pi\sigma^4} \left[1 - \frac{x^2 + y^2}{2\sigma^2} \right] e^{-\frac{x^2 + y^2}{2\sigma^2}}$$

- $L_{\text{LoG}}(x, y, \sigma) = \sigma^2 \nabla^2 (G(\cdot, \cdot, \sigma) * I)$
 - σ^2 normalization to make responses comparable across scales

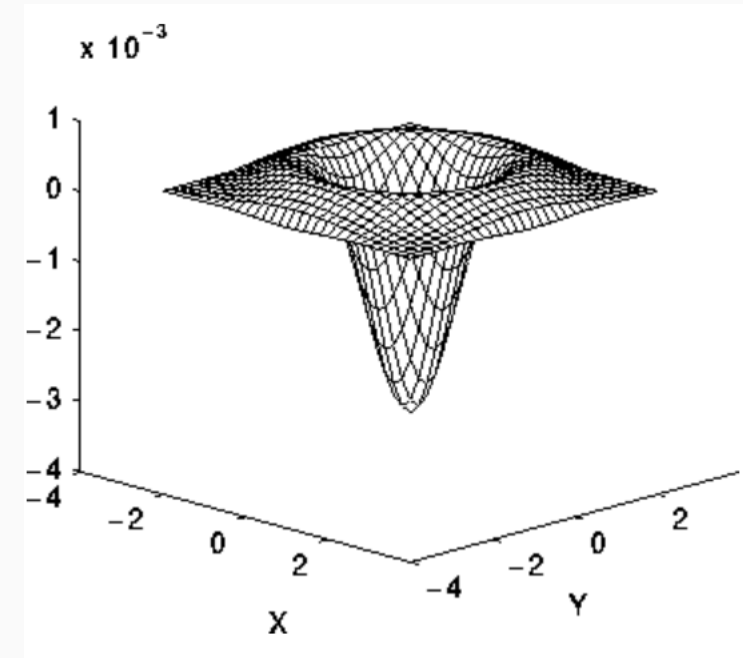


Figure 14: Laplacian of Gaussian.

Laplacian of Gaussian (LoG) for Scale Selection

The **Laplacian of Gaussian (LoG)**:

$$\nabla^2 G(x, y, \sigma) = \frac{\partial^2 G}{\partial x^2} + \frac{\partial^2 G}{\partial y^2}$$

- LoG responds strongly to blob-like structures whose size matches σ (a disk of radius $\approx \sqrt{2}\sigma$ gives the maximal response)
- Scale selection: for a candidate point, evaluate LoG across a range of σ , the σ that **maximizes** the (normalized) response is the characteristic scale

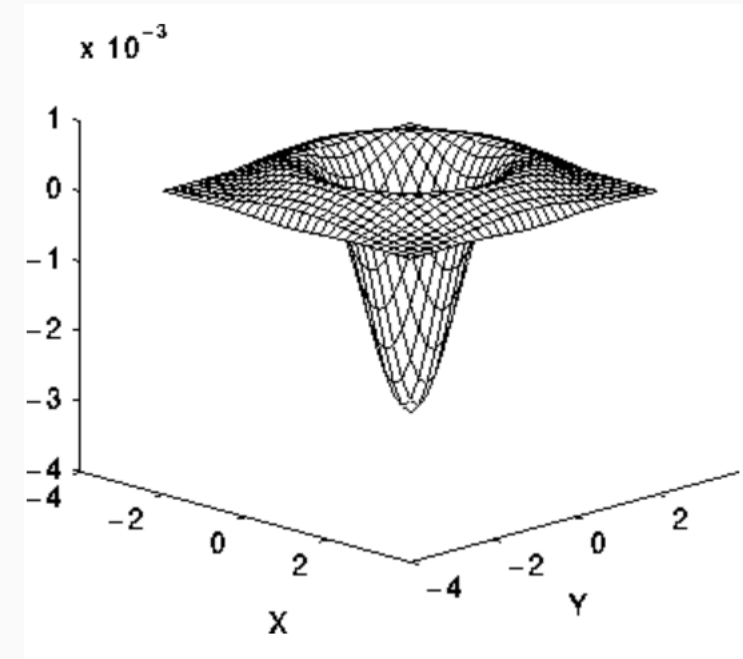


Figure 15: Laplacian of Gaussian.

- LoG is computationally expensive

Difference of Gaussians:

$$D(x, y, \sigma) = G(x, y, k\sigma) * I - G(x, y, \sigma) * I$$

(for a small scale k)

- $D(x, y, \sigma) \approx (k - 1)\sigma^2 \nabla^2 G$: DoG is proportional to the (normalized) LoG
- DoG is a subtraction of two Gaussian-blurred images (already computed for the Gaussian pyramid)

- Build a stack of DoG images $D(x, y, \sigma_1), D(x, y, \sigma_2), \dots$ at closely spaced scales, organized in octaves
 - DoG pyramid, as in the Gaussian pyramid
 - the pyramid is a 3D representation $\text{SCALE} \times \text{WIDTH} \times \text{HEIGHT}$
- A candidate keypoint is a pixel that is a **local extremum** (max or min) in a $3 \times 3 \times 3$ neighborhood: 8 neighbors in the same scale, 9 above, 9 below

We get spatial localization (x, y , like ANMS in Harris) and scale selection (σ) in one search over a 3D volume

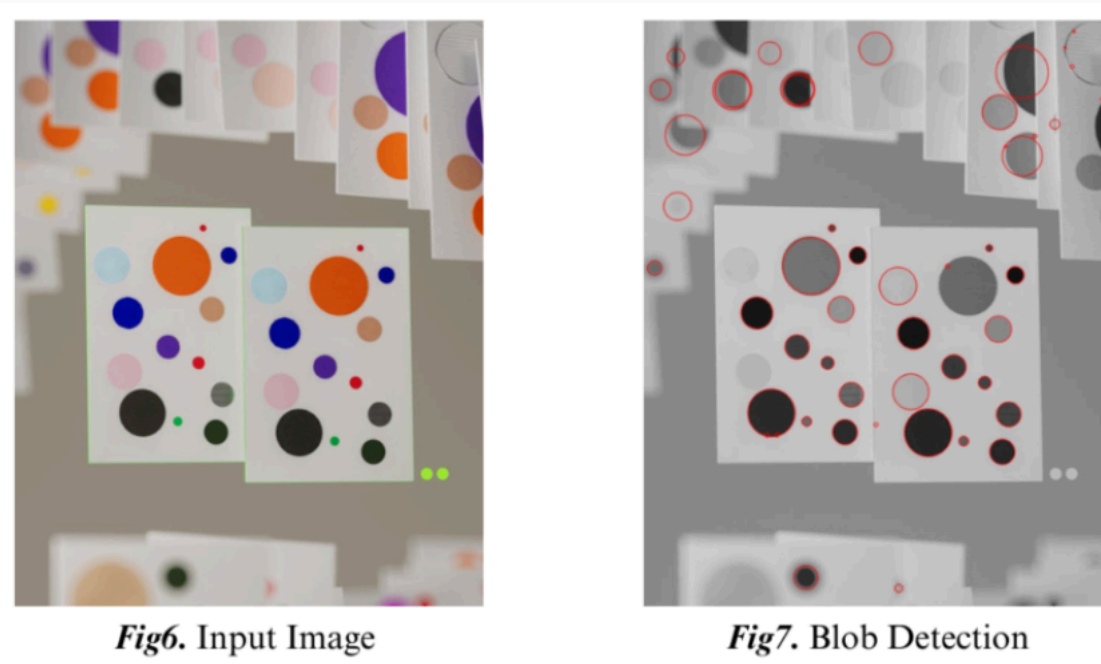


Figure 16: source: opencv.org

- Each keypoint will have its characteristic size
- We have gained scale-equivariance: rescaling the image we can find the same keypoints (at different scales)

Example: DoG Pyramid

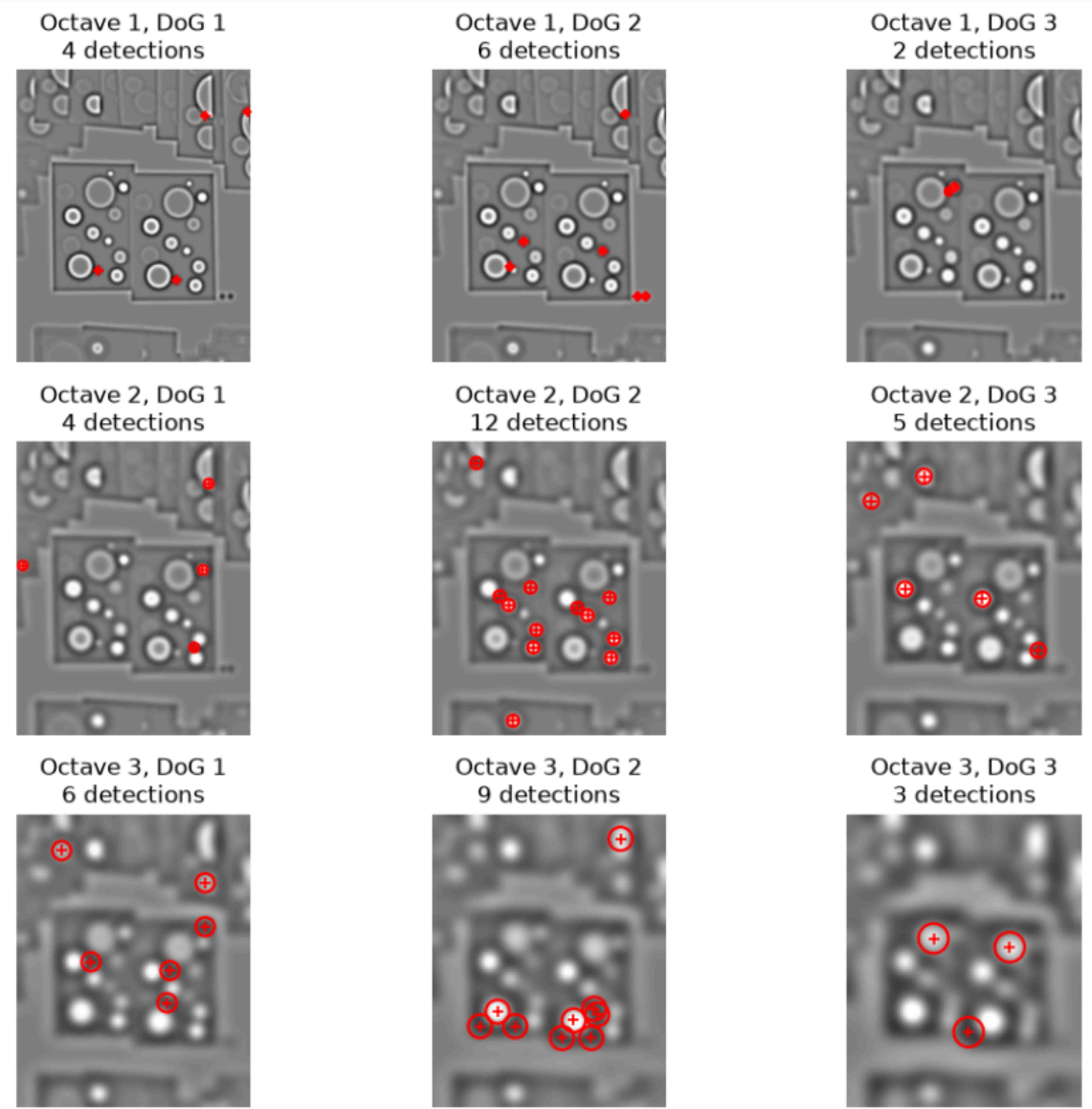


Figure 17: DoG pyramid

- We now have a “blob-like” detector
- We have scale-space detection: a selected keypoint will be associated to its characteristic scale
- note again that the output of the detector is very noisy and it needs to be cleaned (e.g. ANMS, thresholding, ...)
 - manual tuning is also necessary

SIFT: Detection and Localization

- **SIFT** (Scale-Invariant Feature Transform, Lowe 1999/2004): the first widely adopted pipeline combining scale-invariant detection with a robust, distinctive descriptor
- Historically dominant in image stitching, SfM, object recognition for over a decade. De facto baseline that learned methods are still compared against
- **Full pipeline:**
 - DoG detection (scale-invariance)
 - sub-pixel localization (precise localization)
 - orientation assignment (rotation-invariance)
 - gradient-histogram descriptor (normalization for illumination invariance)

SIFT Pipeline Stage 1: DoG Pyramid Construction

- Build a Gaussian pyramid with several scales per octave (typically $s = 3$ intervals)
- Build the DoG pyramid (By subtracting adjacent Gaussian-blurred images)
- Downsample by 2 at the start of each new octave (same construction as the Gaussian/Laplacian pyramids)

Result: a 3D scale-space volume per octave, ready for extrema search

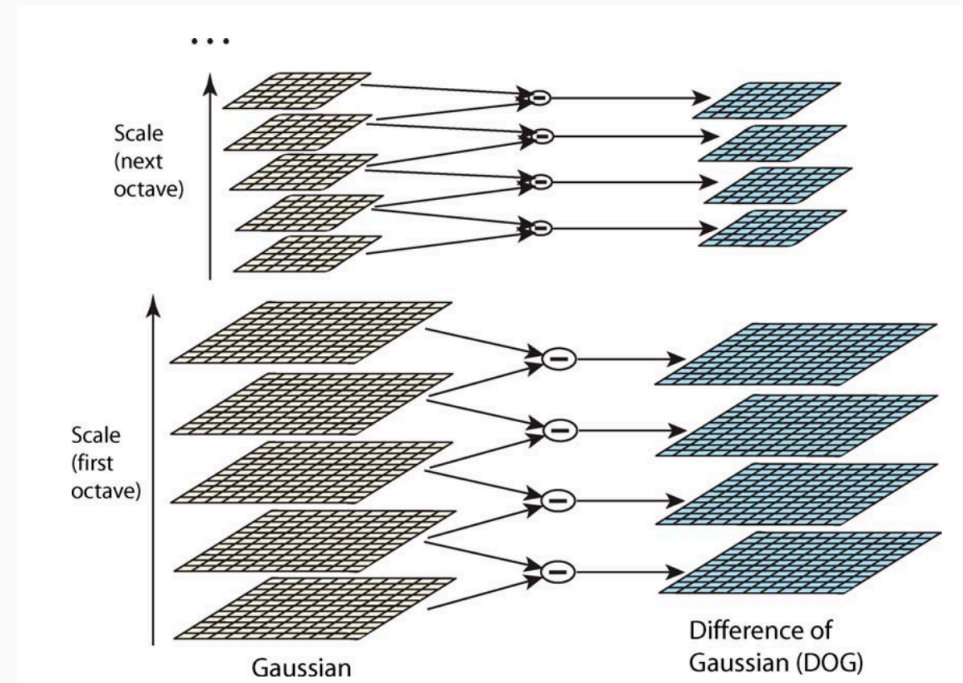


Figure 18: DoG pyramid. source: SIFT paper

SIFT Pipeline Stage 2: Keypoint Localization via Extrema Detection

- Search every DoG image for pixels that are a local extremum among their neighbors
 - 26 neighbors (8 at same scale + 9 above + 9 below)
 - top/bottom scale per octave are excluded
- This produces a large set of **candidate keypoints**
 - many are unstable (low contrast, on edges)
 - we need to filter them in the next steps

Result: Candidate keypoints (x, y, σ) . We have both the keypoint location and its scale.

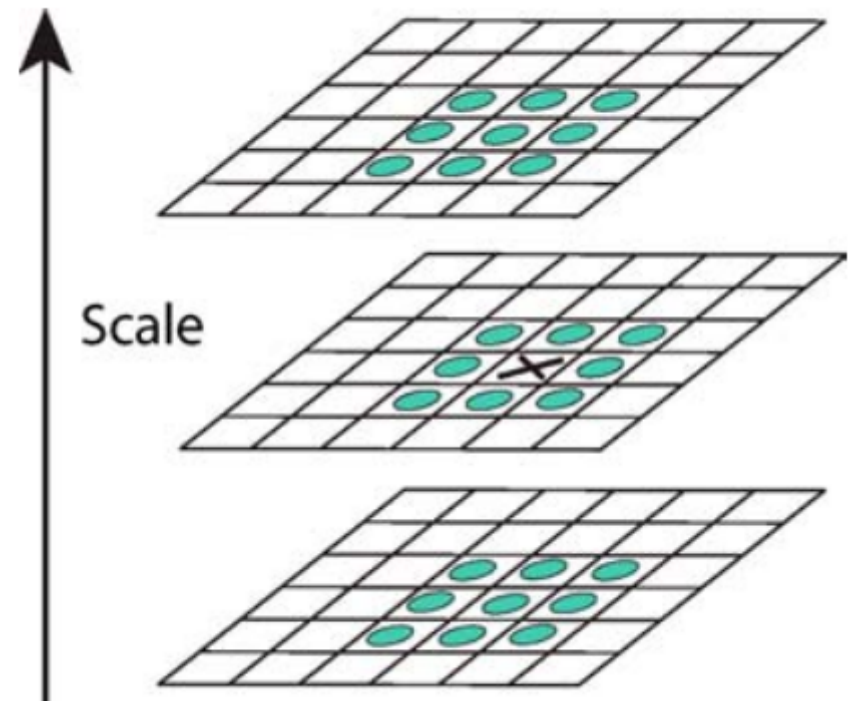


Figure 19: SSED. source: SIFT paper.

Problem: the accuracy of the position of the keypoint is limited by the discrete pixels.

Solution: since keypoints are extrema of the scale-space DoG $D(x, y, \sigma)$, we can do a Taylor expansion of D and search for minima iteratively with a Newton-like method (equations in the paper).

- if the closest point changed after this update (i.e. we moved more than 0.5 pixels in any direction), we repeat it using the derivatives computed at the new closest point

Result: sub-pixel, sub-scale precision beyond the discrete sampling grid

Problem: we have to remove noisy keypoints.

- If $|D(p)| < \tau$ (threshold hyperparameter), discard the keypoint
 - a weak DoG response means low local contrast, sensitivity to noise and unlikely to be repeatable

Result: This removes a large fraction of candidates that are extrema but not robust

Problem: A DoG extremum can lie along an edge: poorly localized along the edge direction (the same aperture problem as Harris)

- Reuse the Harris idea: compute the 2×2 Hessian of D in x, y at the keypoint:

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

- Ratio of eigenvalues $r = \frac{\lambda_{\max}}{\lambda_{\min}}$ is estimated cheaply without eigendecomposition:

$$\frac{\text{trace}(\mathbf{H})^2}{\det(\mathbf{H})} = \frac{(\lambda_{\max} + \lambda_{\min})^2}{\lambda_{\max} \lambda_{\min}} < \frac{(r + 1)^2}{r}$$

- Reject if this ratio exceeds a threshold (Lowe uses $r = 10$): same det/trace trick as the Harris response

Result: we removed problematic edge-like keypoints

- we now have a set of candidate keypoints (x, y, σ) localized in space and scale
- keypoints have been filtered to remove noisy ones (low-contrast, edge-like)
- the resulting keypoints are robust to some small transformations (translation, rotations, scaling, small illumination changes, ...)
- **Next step**: we need to design a **descriptor** for each keypoint

SIFT: Orientation and Descriptor

Goal: assign an orientation to keypoints to achieve rotation-invariance

- For each keypoint $\mathbf{p} = (x, y, \sigma)$, consider the Gaussian smoothed image L (already computed) closest in scale to σ
 - this will give a scale-invariant orientation
- for a point x, y in L , we can compute its gradient magnitude and orientation (finite difference approx. of gradient)
 - **gradient components:** $dx = L(x + 1, y) - L(x - 1, y), \quad dy = L(x, y + 1) - L(x, y - 1)$
 - **magnitude:** $m(x, y) = \sqrt{dx^2 + dy^2}$
 - **orientation:** $\theta(x, y) = \tan^{-1}\left(\frac{dy}{dx}\right)$

Histogram of gradient orientations:

- Compute $m(x, y)$, $\theta(x, y)$ in a neighborhood of p , weighted by a Gaussian (1.5σ) centered on p
- Build a 36-bin histogram of orientations (10 degrees/bin), each pixel votes with weight = gradient magnitude x Gaussian weight
- **Peaks** in the histogram give the keypoint's dominant orientation

Result: we can achieve **rotation invariance** by rotating the patch to align with the peak orientation

- If the orientation histogram has more than one peak within 80% of the maximum, the keypoint is **split** into multiple keypoints, one per dominant orientation
- Examples: corners where two strong edge directions meet; textured regions with ambiguous dominant direction
- Each copy is treated independently downstream (own descriptor, own matches), improving robustness for points that don't have a single clear orientation

Goal: create a keypoint descriptor (vector) that is invariant to transformations to ensure robust matching across different viewpoints

- we now have a robust coordinate systems for each keypoint:
 - location provides translation-invariance
 - scale provides scale-invariance
 - orientation provides orientation-invariance

Naive idea: normalized correlation is not robust enough to non-rigid transformations

Solution: we re-use the gradient histogram as a descriptor

Descriptor: Gradient Histograms in a Local Patch

Using the gaussian-blurred $L(\sigma)$ (closest to keypoint scale), rotated according to the dominant orientation:

- extract a patch around the keypoint
- Compute the gradient magnitude/orientation at every pixel
- Divide the patch into a regular grid of sub-regions, and accumulate a **local orientation histogram** per sub-region (gaussian-weighted)

Result: the final descriptor vector concatenates the histograms

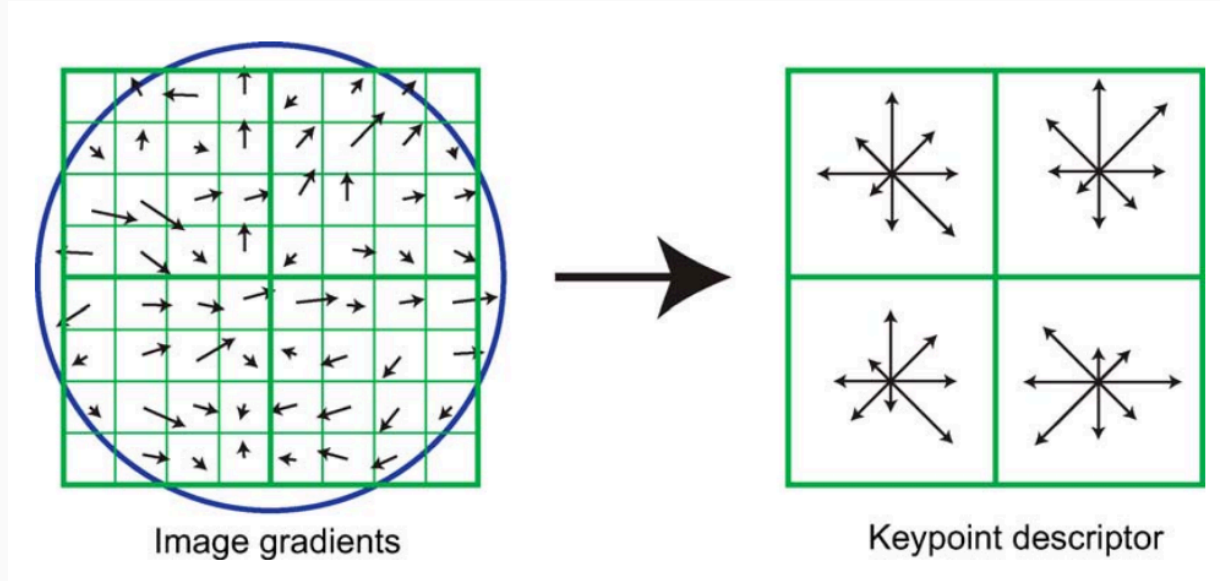


Figure 20: source: SIFT paper.

SIFT 128-D Descriptor

- patch divided into a 4×4 grid of cells
- each cell: an 8-bin orientation histogram (45 degrees/bin)
- $4 \times 4 \times 8 = 128$ numbers total, the SIFT descriptor
- trilinear interpolation of each gradient vote across neighboring bins/cells to avoid hard boundary artifacts

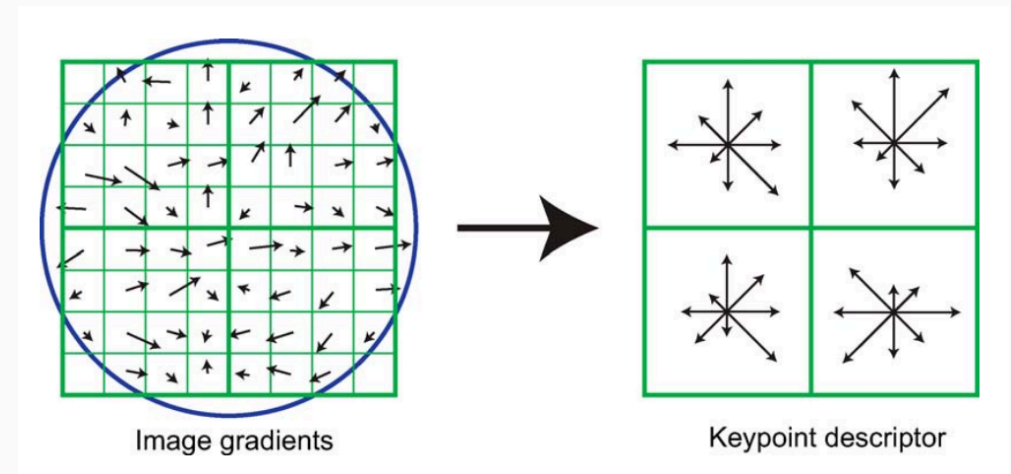


Figure 21: source: SIFT paper.

- Linear (additive) **brightness** changes do not affect gradients (a constant offset has zero derivative)
- Linear **contrast** changes (multiplicative) scale all gradient magnitudes uniformly. SIFT normalizes the 128-D vector.
- **Saturation/non-linear** effects (e.g. camera saturation in bright spots) can create a large gradients: max-clip (0.2 after normalization), then re-normalize

Result: robustness to affine intensity changes and to localized saturation artifacts



Figure 22: source: opencv.org

Invariance	Mechanism
Scale	DoG scale-space extrema, σ stored per keypoint
Rotation	Dominant orientation estimated and used to align the patch
Illumination (additive)	Gradients are insensitive to constant offsets
Illumination (contrast)	Unit-norm descriptor normalization
Localization noise	Sub-pixel quadratic refinement
Edge instability	Hessian-based edge rejection

Toward Matching

Viewpoint change: camera panned between the two shots

image A



image B



Figure 23: Same scene, different viewpoint

Harris corners detected independently on each image

image A: 79 corners



image B: 70 corners



Figure 24: Harris edge detector.

SIFT keypoints (circle = scale, line = orientation)

image A: 500 SIFT keypoints

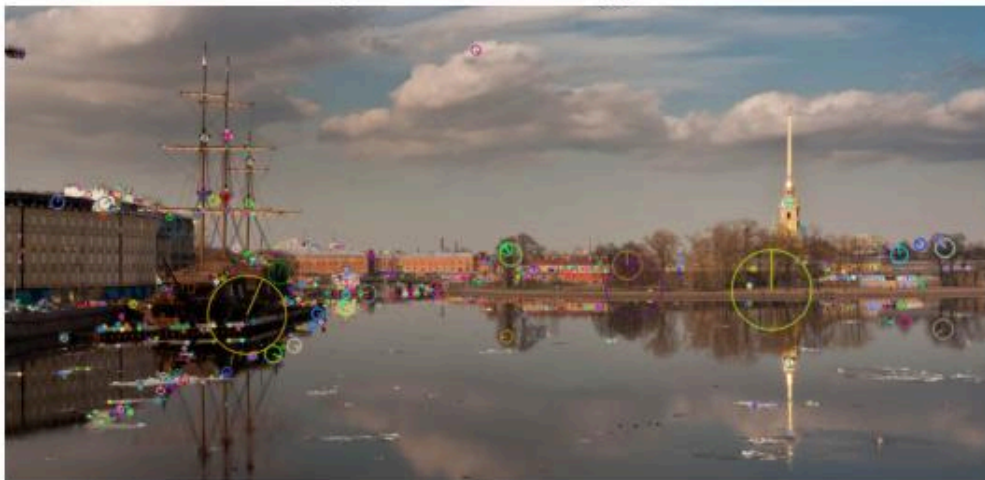


image B: 500 SIFT keypoints

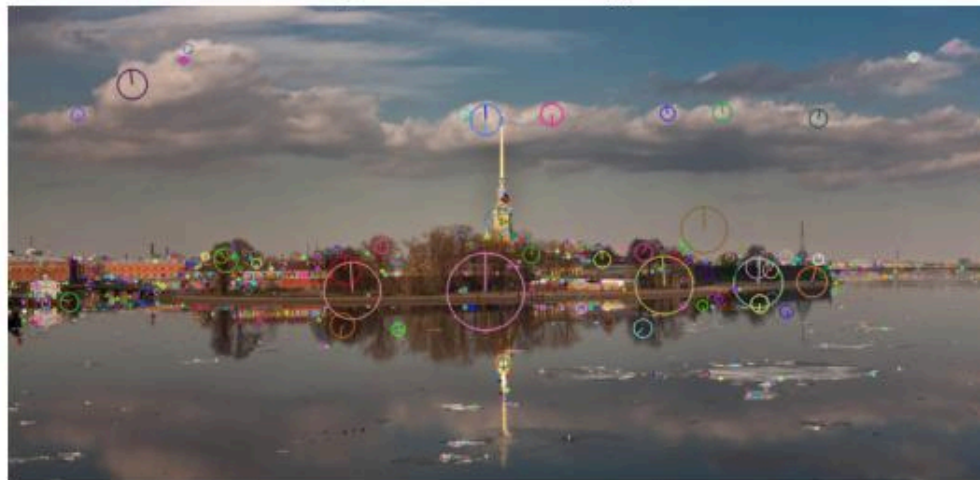


Figure 25: SIFT keypoints.

- 500 SIFT keypoints per image, circle=scale, line=orientation
- SIFT keypoints appear at many different scales (small circles on fine texture, large circles on the spire/boat)
- Harris corners are all detected at one fixed window size

- **Repeatability** = fraction of keypoints in image 1 that have a correct corresponding keypoint detected near the right location in image 2
- SIFT shows higher repeatability than plain Harris under scale changes, and comparable or better under moderate viewpoint change due to the orientation/scale normalization
- Both degrade under extreme viewpoint change and repetitive textures

Conclusions

SIFT:

- interpretable and very easy to use
- a clean model of invariance through normalization
- a strong baseline for sparse geometry tasks

Modern learned feature pipelines often reuse the same modular abstraction: detect, describe, match, verify.

A full pipeline to find correspondences given an image pair:

- detect keypoints and assign location, scale, and orientation
- compute descriptors
- **(MISSING) matching**: nearest-neighbor matching and Lowe's ratio test
 - we will need to remove some keypoints after matching (e.g. occlusions, poor matches, ...)
- **(MISSING) geometric estimation + verification**: DLT and RANSAC

- Harris detects corners via local gradients (A), classified via its eigenvalues, approximated by det/trace
- Scale invariance comes from searching for extrema across a scale-space (LoG/DoG)
- SIFT = DoG detection + sub-pixel refinement + edge/low-contrast rejection + orientation-normalized gradient-histogram descriptor
- With stable keypoints detection and description, many problems are reduced to finding correspondences: detect repeatable keypoints, describe them distinctively, then match

- Lowe, “Distinctive Image Features from Scale-Invariant Keypoints” (2004)
- Szeliski Sec. 7.1

- Given keypoints + descriptors from two (or more) images, how do we find **correct** correspondences?
- Nearest-neighbor descriptor matching, and why a distance threshold alone is not enough (ambiguous matches, repetitive texture)
- **RANSAC**: robustly fitting a geometric model (e.g. a homography) while ignoring outlier matches
- use case: image stitching and panoramas