Taif University
College of Computers and Information Technology

Image Processing & Computer Vision

(AI:501837-3) - (DS:501838-3)

Programs:
Master of AI
Master of Data Science

Department:
Computer Science

Year:
Spring 2025

Lecture 6:
Keypoints Points Detection & Descriptors

Prepared by: Maha A. Thafar

Lecture Outline

Feature Detection

- Intro to Image Matching
- Interest Points
  - Corners Detection
- Illustration Examples.

Feature Description

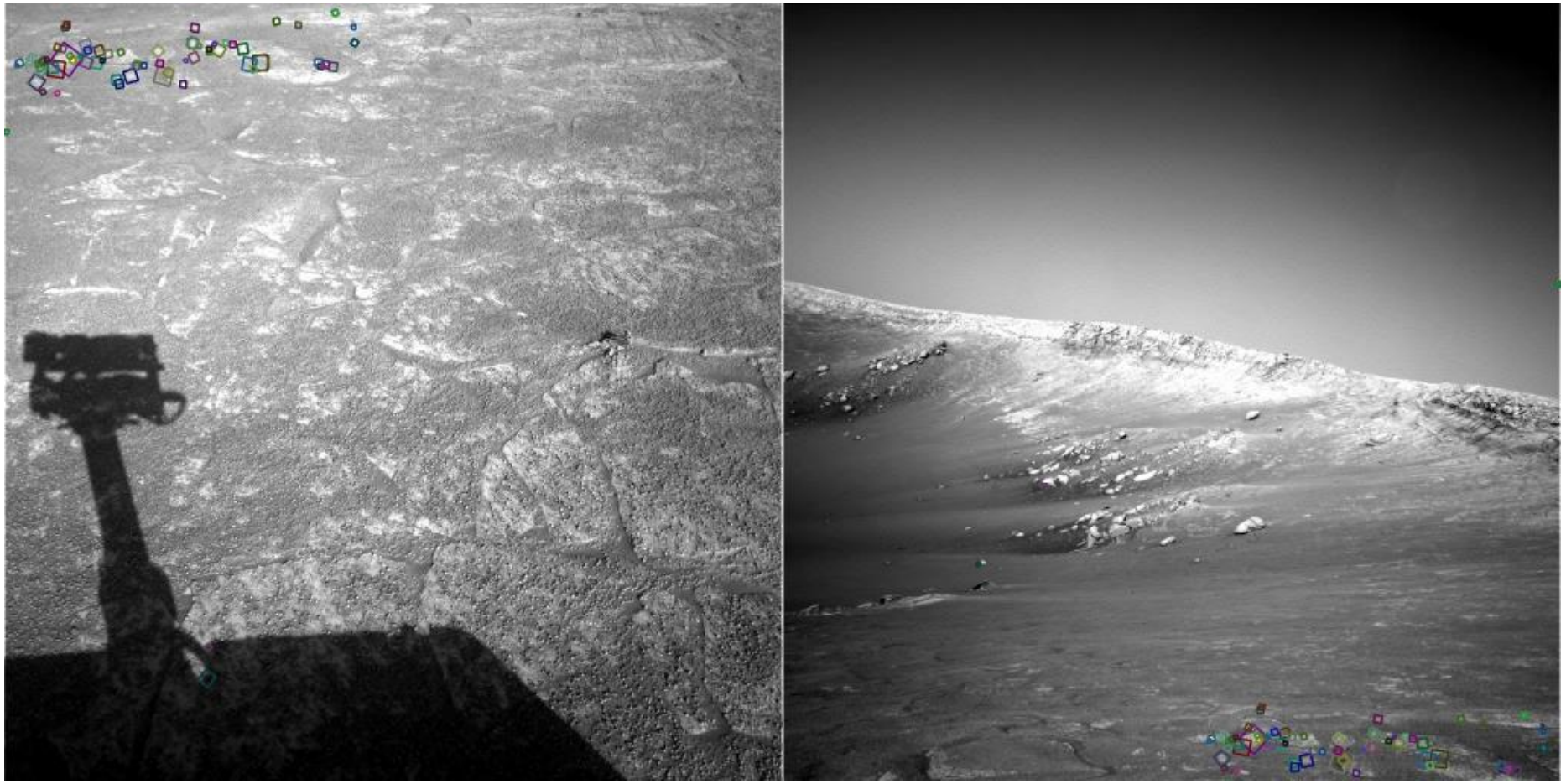
- Object Detection
  - SIFT Algorithm
  - HOG Algorithm

Feature Matching

- Local Feature Matching
- Feature Distance
- Evaluating the matching results

Image Matching: Intro

Hard Case?

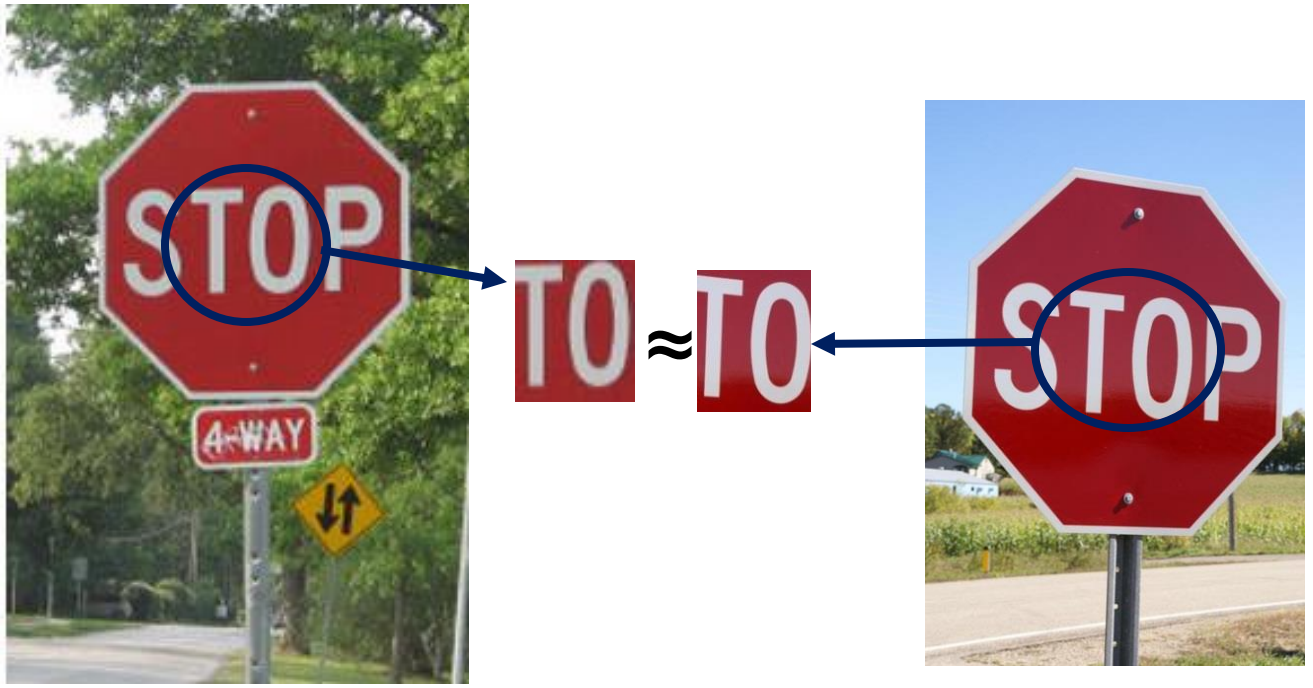


NASA Mars Rover images with SIFT feature matches
Figure by Noah Snavely

Correspondence across views

Matching points, patches, edges, or regions across images.

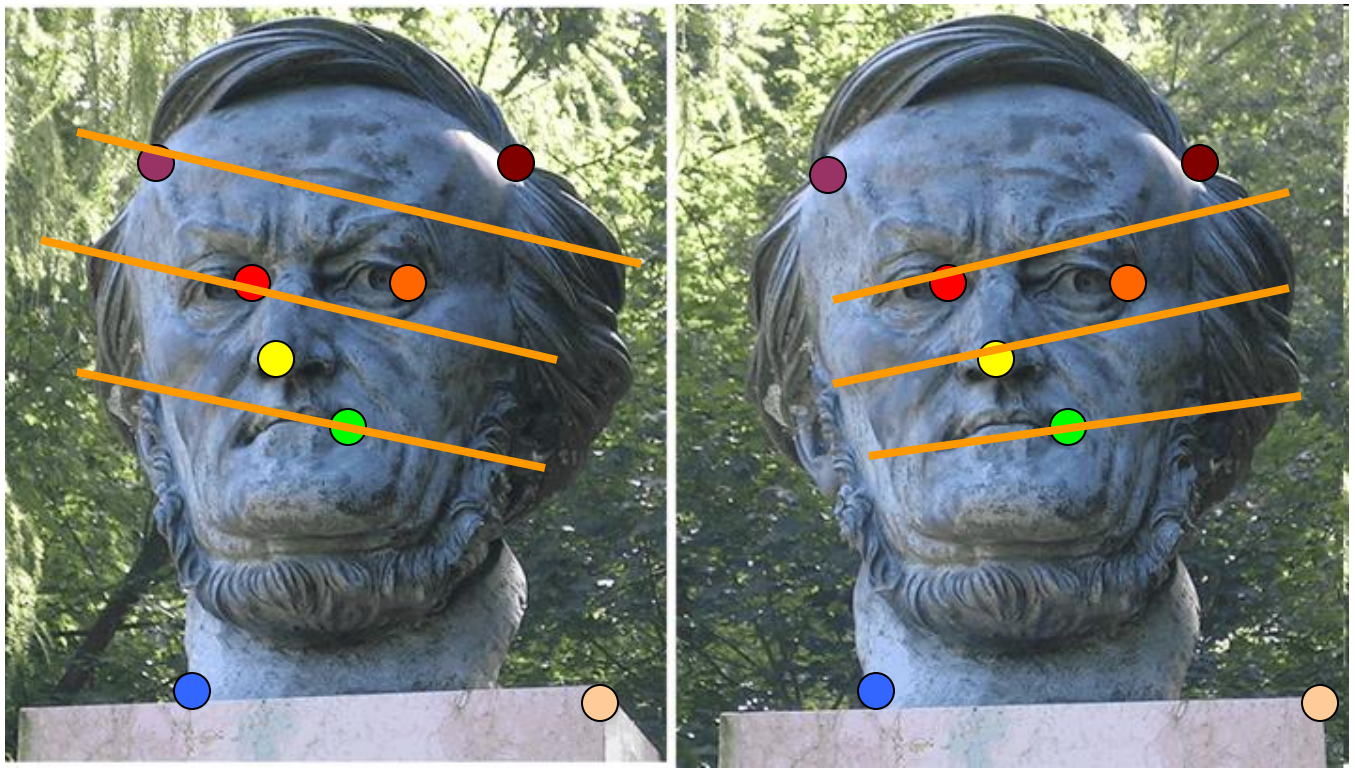
Sparse or local correspondence vs. dense (at every pixel)



Our goal is to find which parts of the images **correspond** to what "**template**" we have as a **stop sign**.

Correspondence in Reconstruction

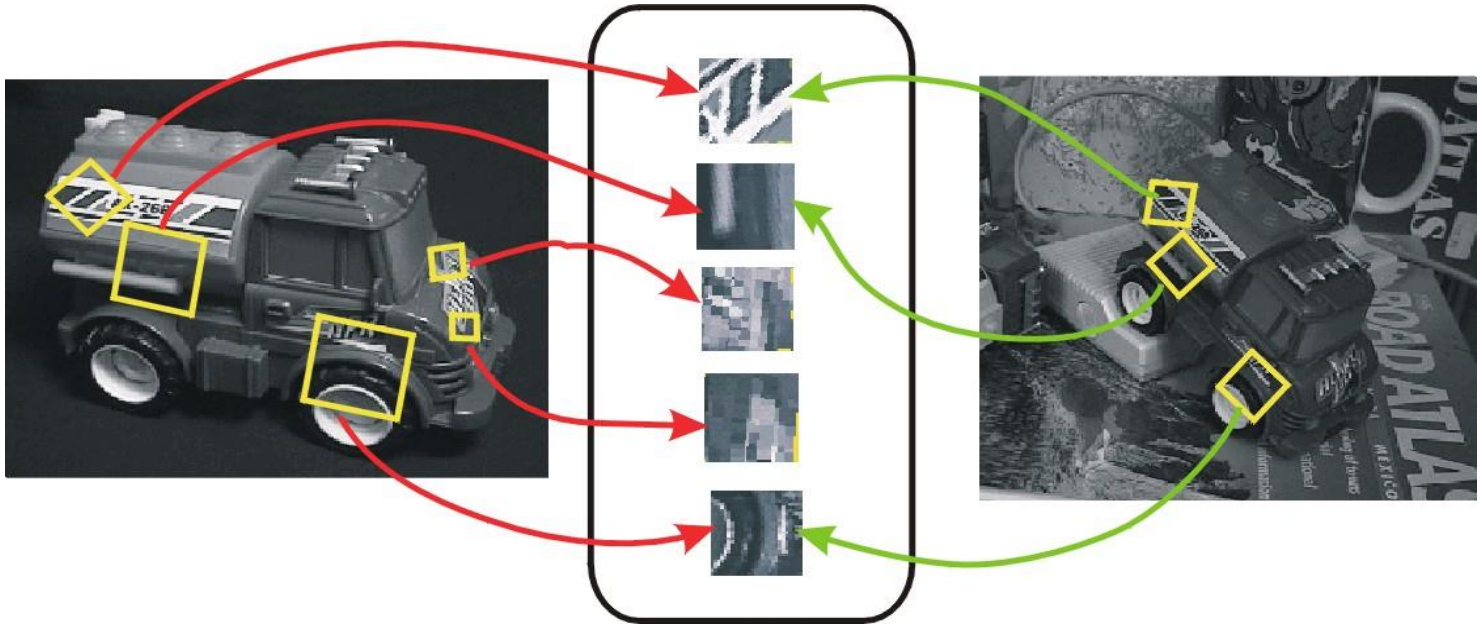
Example: estimating "**fundamental matrix**" that corresponds two views



Correspondence in Object Detection

Finding ***distinctive*** and ***repeatable*** **feature points** can be difficult when we want our features to be invariant to large transformations:

- **Geometric variation:** (translation, rotation, scale, perspective)
- **Appearance variation:** (reflectance, illumination)



Features (key points) Descriptors

Image Features

Features

Filtering → Edges → **Corners**

Corners are also called
interest points or **key points**
"Local features"

- **Features** may be specific structures in the image, such as points, corners, edges, or objects.
- **Features** may also be the result of feature detection applied to the image.
- **Local features** (i.e., corners) describe one component of the image, not the entire image.
- **Corners** are important because we can get spare correspondences, which allows for fast matching between images.

Slides from Rick Szeliski, Svetlana Lazebnik, Derek Hoiem and Grauman&Leibe 2008 AAAI Tutorial

Fundamental to Applications

More Motivation: Feature points are used for:

- **Image alignment**
- **3D reconstruction:** find correspondences across different views.
- **Motion tracking:** which points are good to track?
- **Object recognition:** find patches likely to tell us something about object category

Robust feature alignment/ Panorama stitching

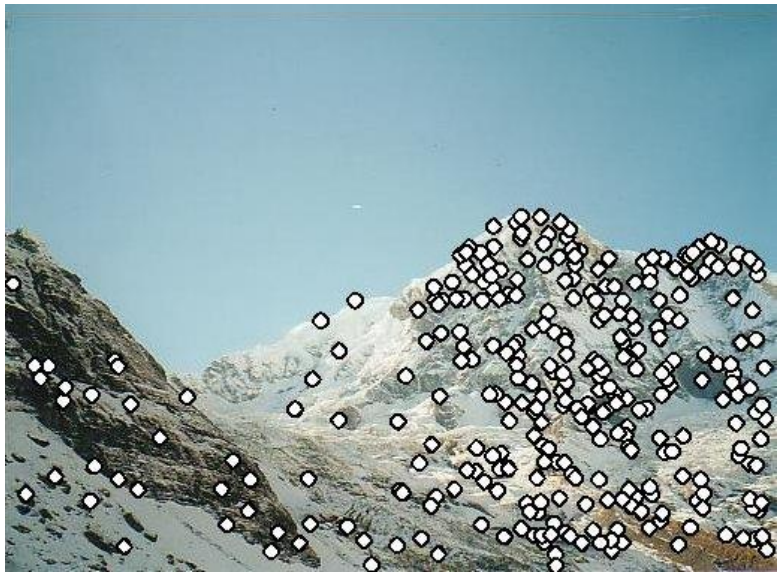
We have two images – how do we overlay them?

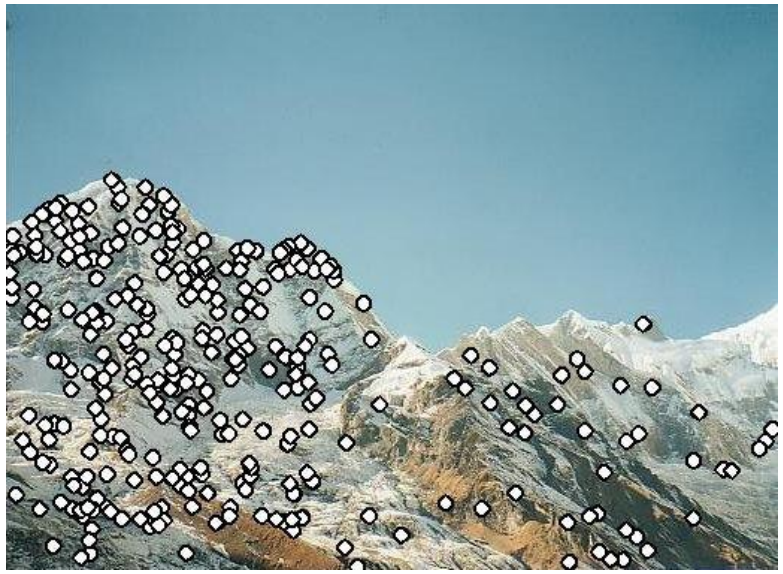
i.e., How to detect *which features* to match?

Using panorama (on your phone) requires correspondence. We have two images of a scene, and we can see that there is some overlay, but the images are not quite the same.

Source: L. Lazebnik

Robust feature alignment/ Panorama stitching





- Extract features

What makes a good feature?

What points would you choose?

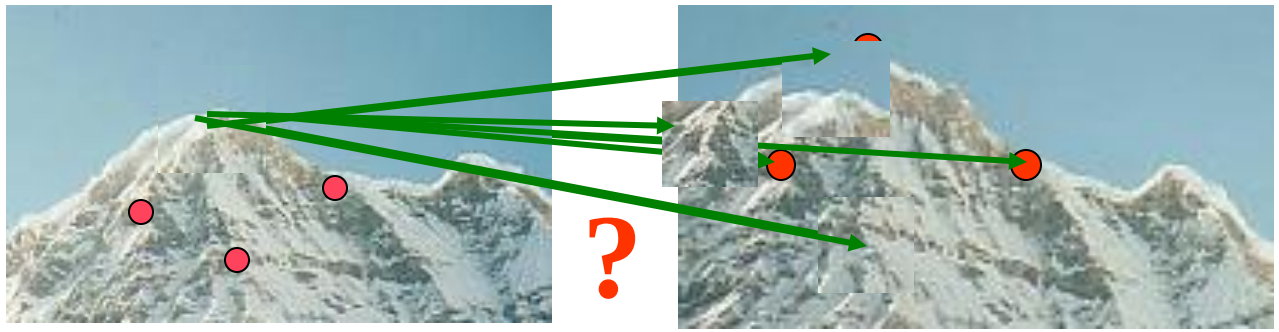
Source: L. Lazebnik

Goal: Distinctiveness

To solve our problem of panorama stitching, we need our features to have some properties:

1. **distinctiveness**, meaning the features we are able to extract from the image have to be unique in some way.

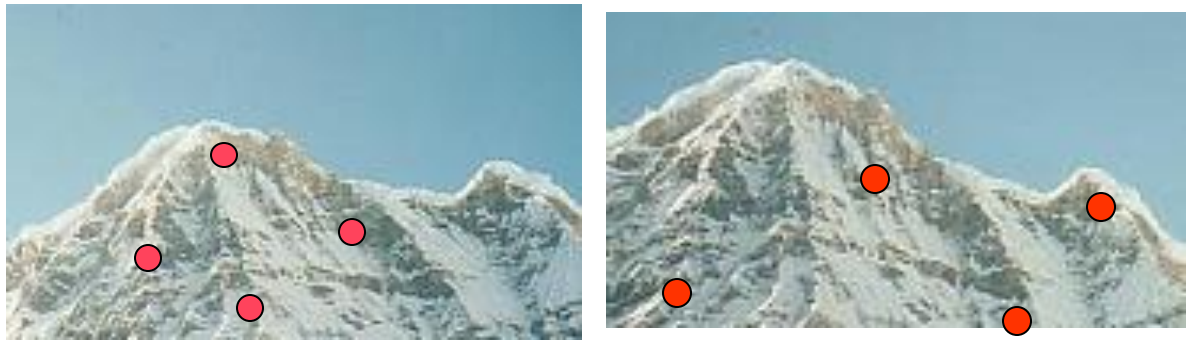
We want to be able to reliably determine which point goes with which.



May be difficult in structured environments with repeated elements

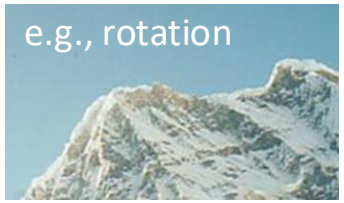
Goal: Repeatability

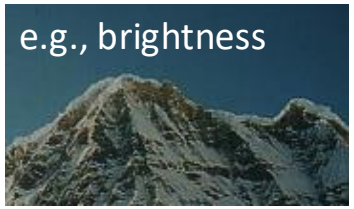
We want to detect (at least some of) the same points in both images.



With these points, there's no chance to find true matches!

Under geometric and photometric variations.



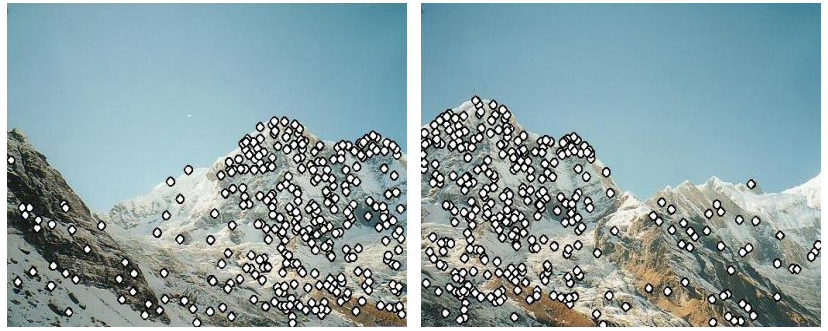


2. **Repeatability**: Even if we have two different views of the same scene, we want to be able to identity the same point in both images. We can only match features if both points show up in the same image!

Local Feature: Main Components

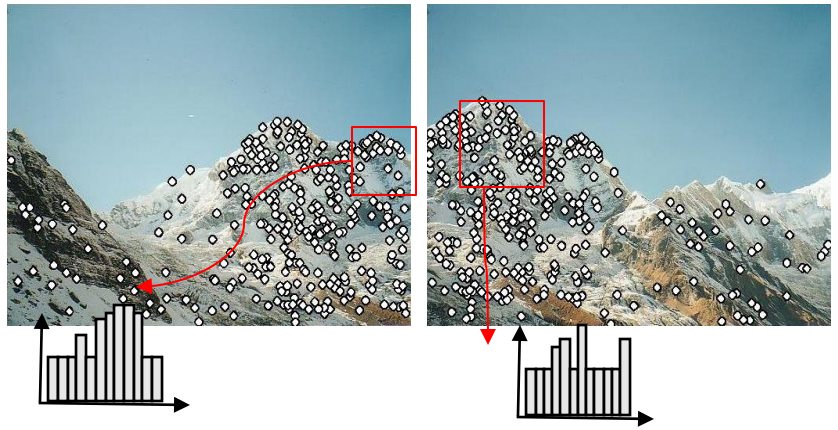
1) Detection:

Identify the interest points (Find a set of distinctive features) --> **Corners**.



2) Description:

Extract vector feature descriptor surrounding each interest point.



3) Matching:

Determine correspondence between descriptors in two views by Computing distance between feature vectors

Kristen Grauman

Corner Detection: Basic idea

We do not know which other image locations the feature will end up being matched against.

But we can compute how stable a location is in appearance with respect to small variations in its position.

Strategy:
Compare image patch against local neighbors.

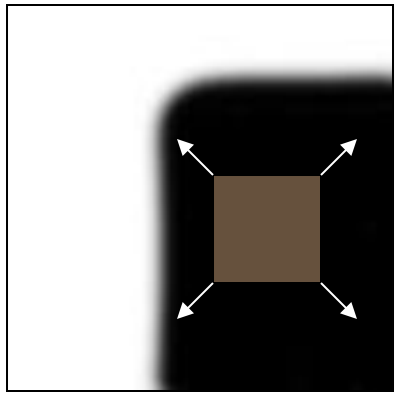
Slides from Rick Szeliski, Svetlana Lazebnik, and Kristin Grauman

Corners as distinctive interest points

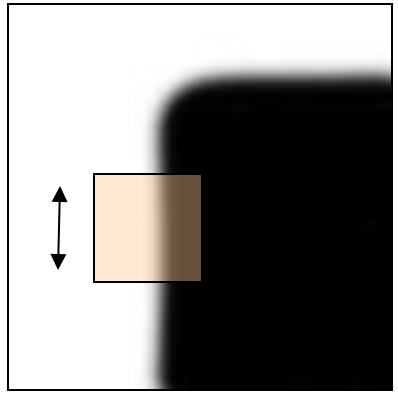
Local measure of feature **uniqueness:** Suppose we only consider a small window of pixels.

- **What defines whether a feature is a good or bad candidate?**
  - We should easily recognize the corner by looking through a **small window → Shifting** a window in *any direction* should give **a large change in intensity**

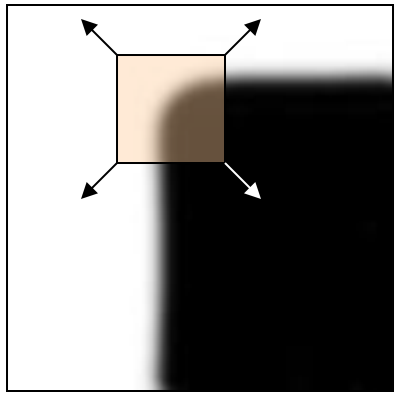
- Corner: **Point where Two Edges Meet**. *i.e., Rapid Changes of Image Intensity in **Two Directions** within a Small Region.*
- Harris Corner Detector is commonly used in computer vision algorithms to extract corners and infer features of an image based on the assumption: *"corners have high variation in intensity in all directions, while edges have high variation in intensity in only one direction".* It was first introduced by Chris Harris and Mike Stephens in 1988.



"Flat" region:
no change in all directions



"Edge":
no change along the edge direction



"Corner":
significant change in **all** directions

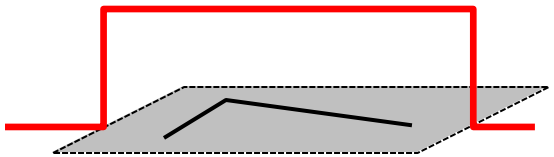
Corner Detection: Mathematics

Change in appearance of window $w(x,y)$ for the shift $[u,v]$:

$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u,y+v) - I(x,y)\right]^2$$

Window function — **Shifted intensity** — **Intensity**

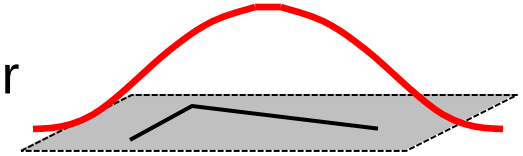
Window function $w(x,y)$ =



1 in window, 0 outside

Uniform

or



Gaussian

Source: R. Szeliski

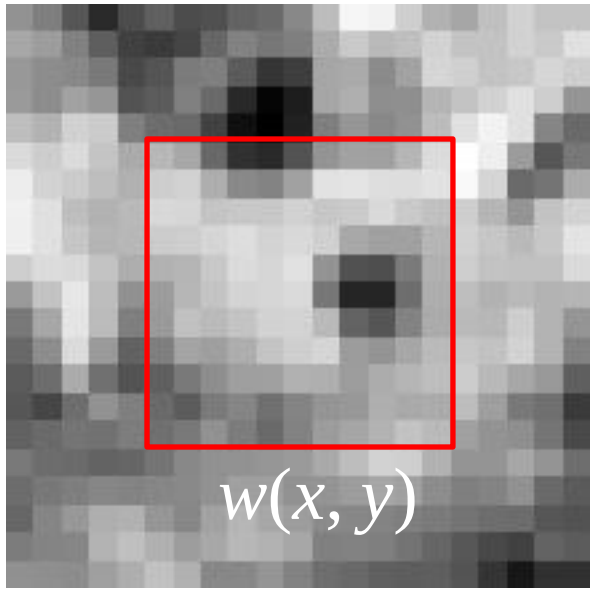
Corner Detection: Mathematics

Change in appearance of window $w(x,y)$ for the shift $[u,v]$:

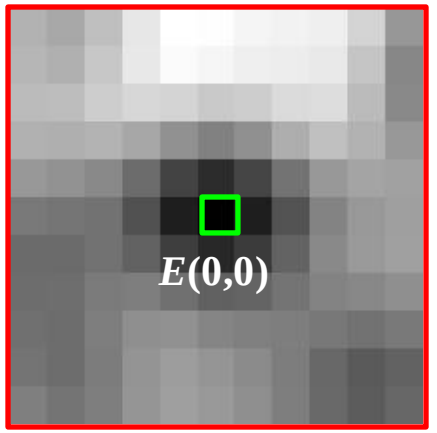
$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u,y+v) - I(x,y)\right]^2$$

We want to find out how this function behaves for small shifts

$I(x,y)$



$E(u,v)$



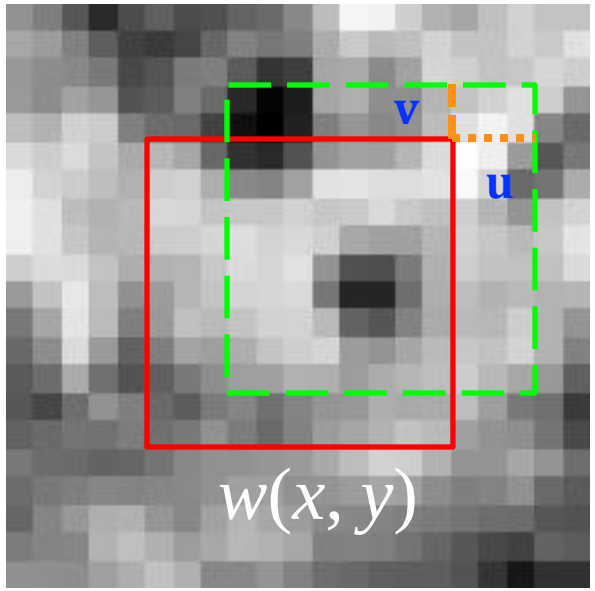
Corner Detection: Mathematics

Change in appearance of window $w(x,y)$ for the shift $[u,v]$:

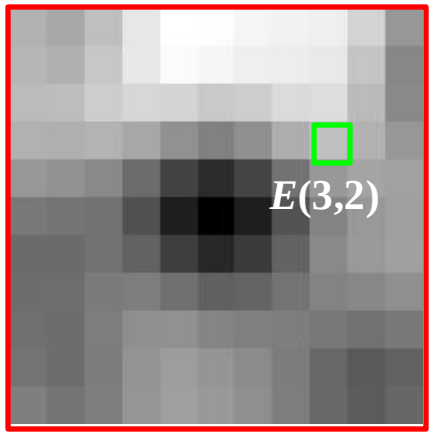
$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u,y+v) - I(x,y)\right]^2$$

We want to find out how this function behaves for small shifts

$I(x,y)$



$E(u,v)$



Corner Detection: Mathematics

We want to discover how E behaves for small shifts

Change in appearance of window $w(x,y)$ for the shift $[u,v]$:

$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u, y+v) - I(x,y)\right]^2$$

Window function

Shifted intensity

Intensity

- For nearly constant patches, this will be close to '0'.
- For very distinctive patches, this will be large

Hence, we want the patches where the error E(u,v) is LARGE (strong peak)

Source: R. Szeliski

Taylor Series Mathematics

$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u,y+v) - I(x,y)\right]^2$$

- Taylor Series says that the function: f(x) can be represented at point a in terms of its derivatives
- We want to find out **how this function behaves for small shifts**

$$E(u,v) = \sum w(x,y)\left[I(x,y) + uI_x + vI_y - I(x,y)\right]^2$$

Taylor Series

$$E(u,v) = \sum w(x,y)\left[uI_x + vI_y\right]^2$$

$$E(u,v) = \sum w(x,y)\left[(u\ \ v)\begin{pmatrix} I_x \\ I_y \end{pmatrix}\right]^2$$

$$E(u,v) = \sum w(x,y)\ (u\ \ v)\begin{pmatrix} I_x \\ I_y \end{pmatrix}(I_x\ \ I_y)\begin{pmatrix} u \\ v \end{pmatrix}$$

Corner Detection: Mathematics

$$E(u,v) = \sum w(x,y) \; (u \;\; v) \begin{pmatrix} I_x \\ I_y \end{pmatrix} (I_x \;\; I_y) \begin{pmatrix} u \\ v \end{pmatrix} \quad \Rightarrow \quad E(u,v) = (u \;\; v) [\sum w(x,y) \begin{pmatrix} I_x \\ I_y \end{pmatrix} (I_x \; I_y)] \begin{pmatrix} u \\ v \end{pmatrix}$$

The quadratic approximation simplifies to

$$E(u,v) \approx [u \;\; v] \begin{bmatrix} E_{uu}(0,0) & E_{uv}(0,0) \\ E_{uv}(0,0) & E_{vv}(0,0) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$E(u,v) \approx [u \;\; v] \; M \begin{bmatrix} u \\ v \end{bmatrix}$$

E(u,v) is an equation of ellipse, where *M* is a ***second moment matrix*** computed from image derivatives:

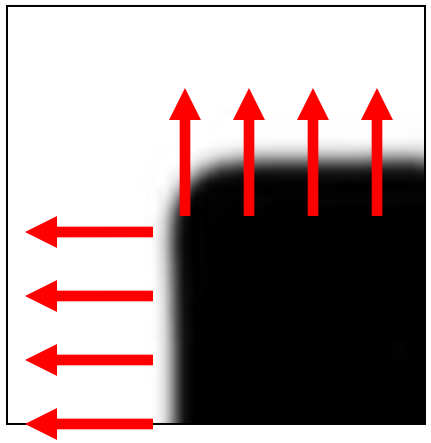
$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

The second moment matrix, is a 2x2 matrix that describes the distribution of intensity gradients in an image. It is calculated by taking the **gradient** of the image, which describes how the brightness of the image changes in different directions. The matrix provides information about the strength and orientation of edges in the image and can be used for corner detection task.

$$M = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} = \sum \begin{bmatrix} I_x \\ I_y \end{bmatrix} [I_x \; I_y] = \sum \nabla I (\nabla I)^T$$

What does this matrix reveal?

consider an axis-aligned corner:



$$M = \sum \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$$

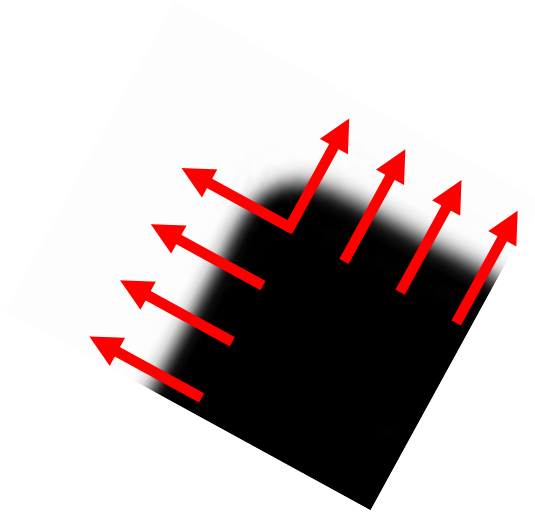
- This means dominant gradient directions align with x or y axis
- Look for locations where **both** λ's are large.
- If either λ is close to 0, then this is **not** corner-like.

What if we have a corner that is not aligned with the image axes?

What does this matrix reveal?

Since *M* is symmetric, we have

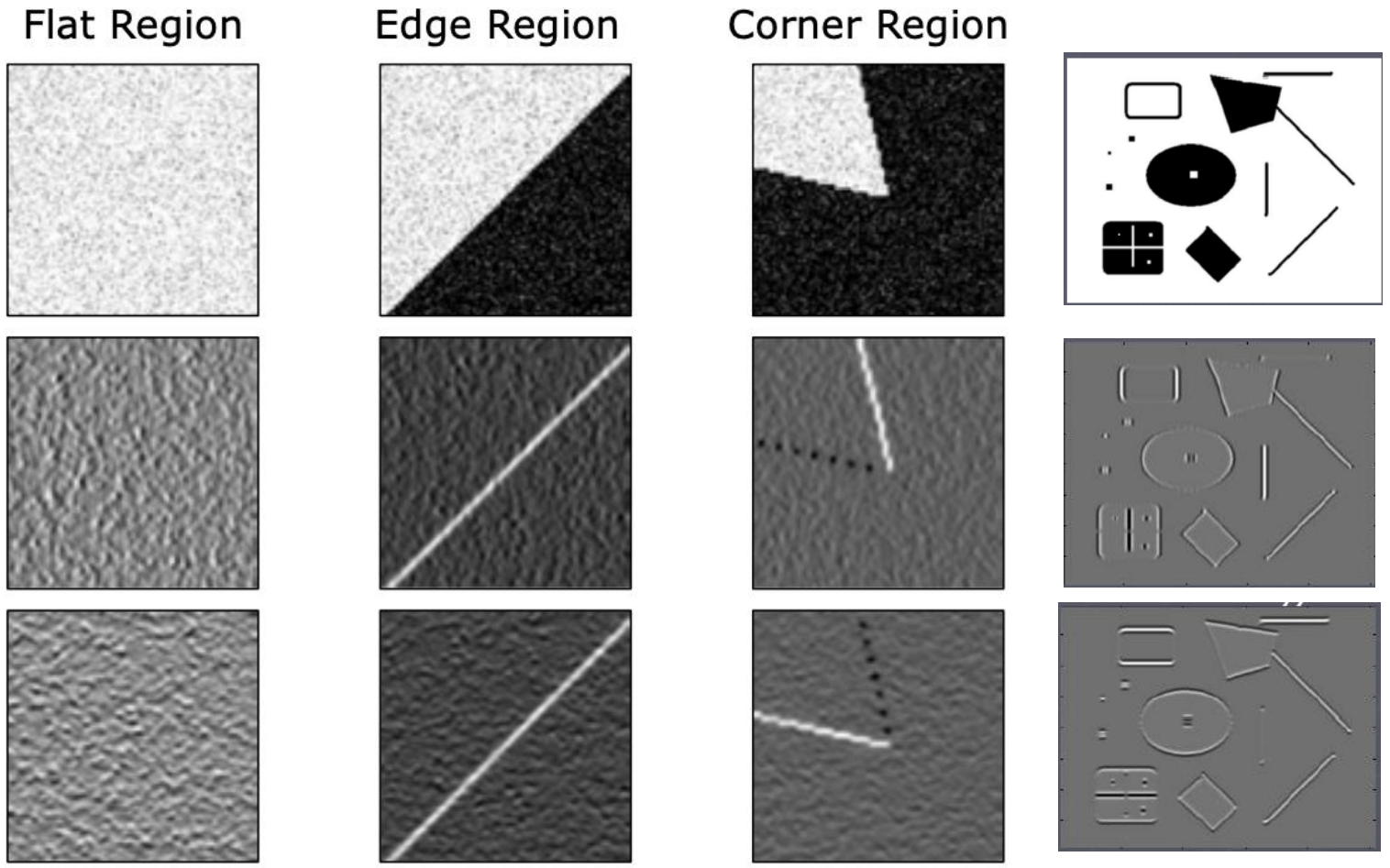
$$M = X \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} X^T$$

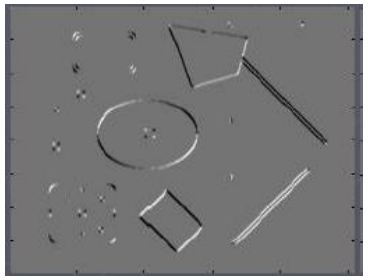


$$Mx_i = \lambda_i x_i$$

The ***eigenvalues*** of *M* reveal the amount of intensity change in the two principal orthogonal gradient directions in the window.

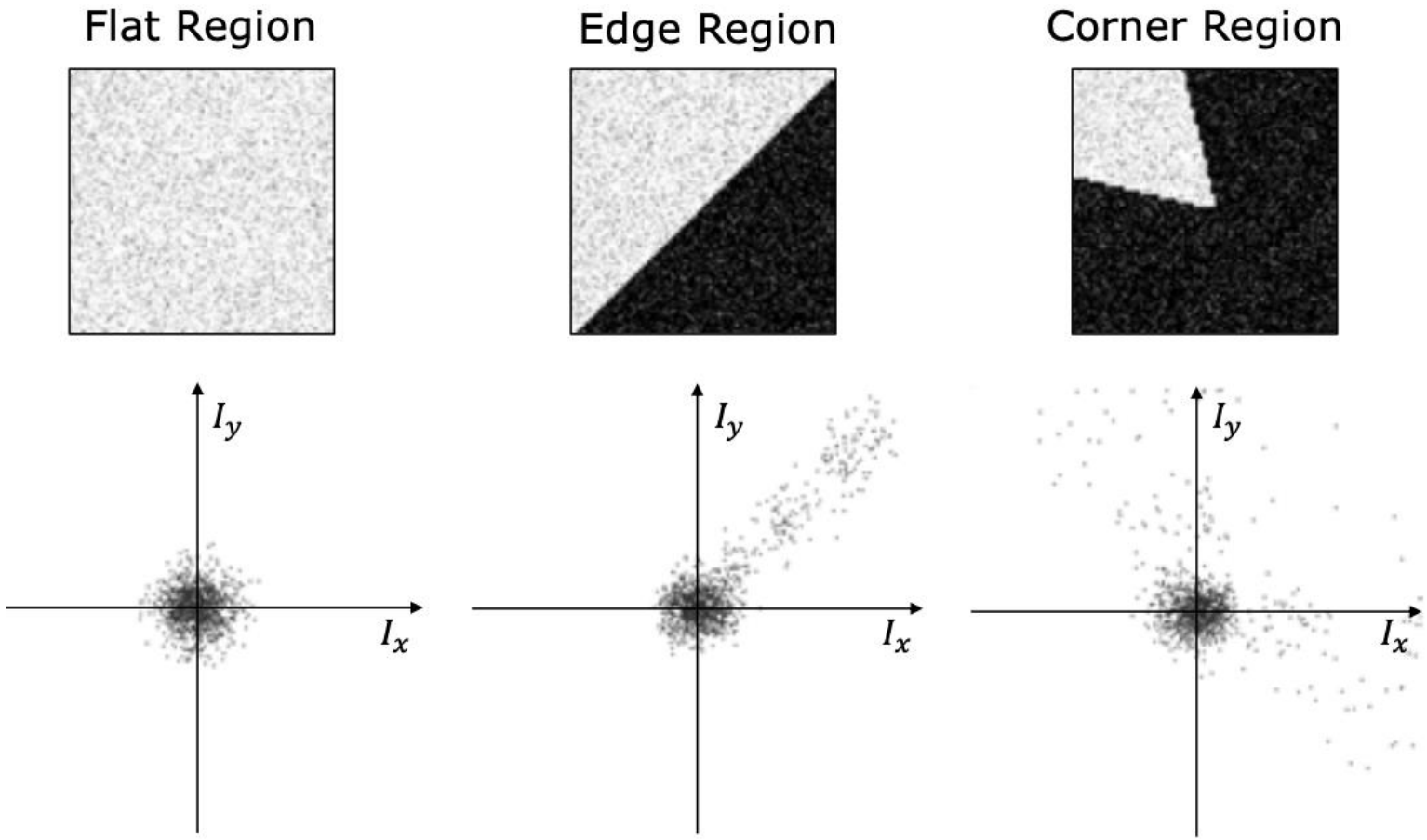
Image Gradient





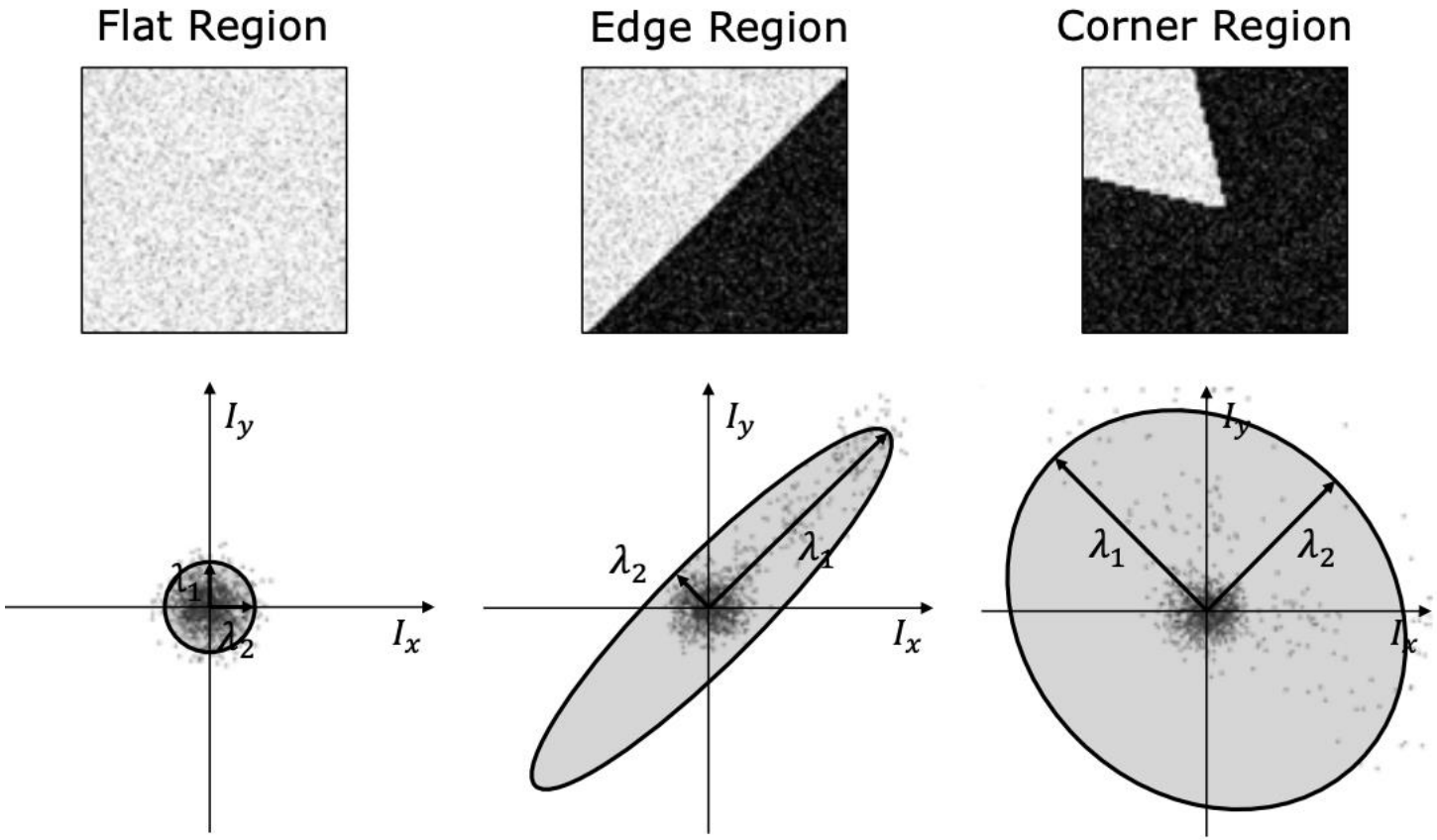
$$I_x I_y \Leftrightarrow \frac{\partial I}{\partial x}\frac{\partial I}{\partial y}$$

Distribution of Image Gradient



Distribution of I_x and I_y is different for all three regions.

Fitting Elliptical Disk to Distribution



λ_1: Length of Semi-Major Axis λ_2: Length of Semi-Minor Axis

Second Moment Matrix & Ellipse Fitting Approach

- The **Ellipse Fitting approach** is one of several methods that can be used to estimate the **Second Moment Matrix** from an image.
- To **estimate the second moment matrix**, we **fit an ellipse to the distribution** of intensity gradients in the image.
- **To fit an ellipse**, we first compute the gradient of the image using a gradient operator.
- We then compute the second moment matrix of the gradient image for each pixel in the image. When computing the second moment matrix, we are interested in calculating the distribution of intensity gradients in the image, which can be represented by **a 2D Gaussian function**.

Harris Corner Detection :
Interpreting the Second Moment Matrix

Consider a horizontal "slice" of $E(u, v)$: $[u \;\; v]\; M \begin{bmatrix} u \\ v \end{bmatrix} = \text{const}$

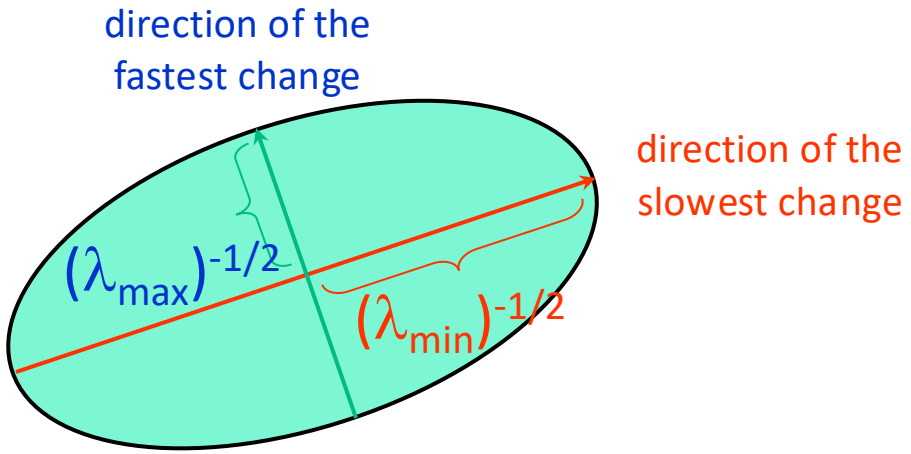
This defines an ellipse.

Diagonalization of M: $M = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R$ (A)

The direction is determined by a rotation matrix R (eigenvectors), The axis lengths of the ellipse are determined by the eigenvalues *represent the variance of the distribution (strength) in the corresponding directions.*

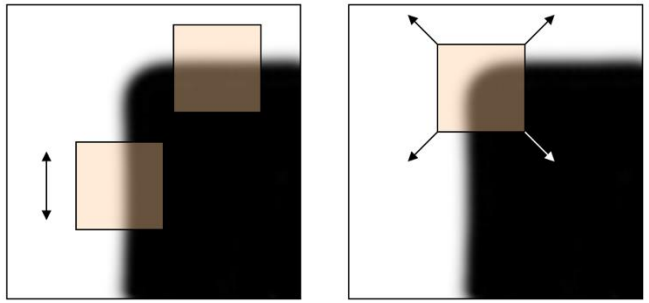
Note inverse relationship:

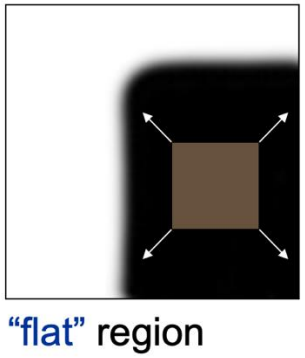
larger eigenvalue = smaller ellipse radii in visualization

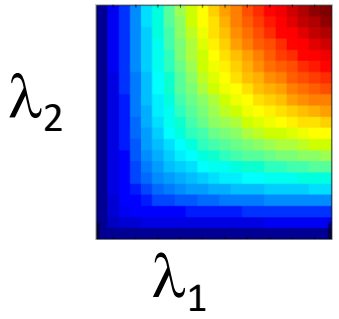


Harris Corner Detection :
Interpreting the Second Moment Matrix

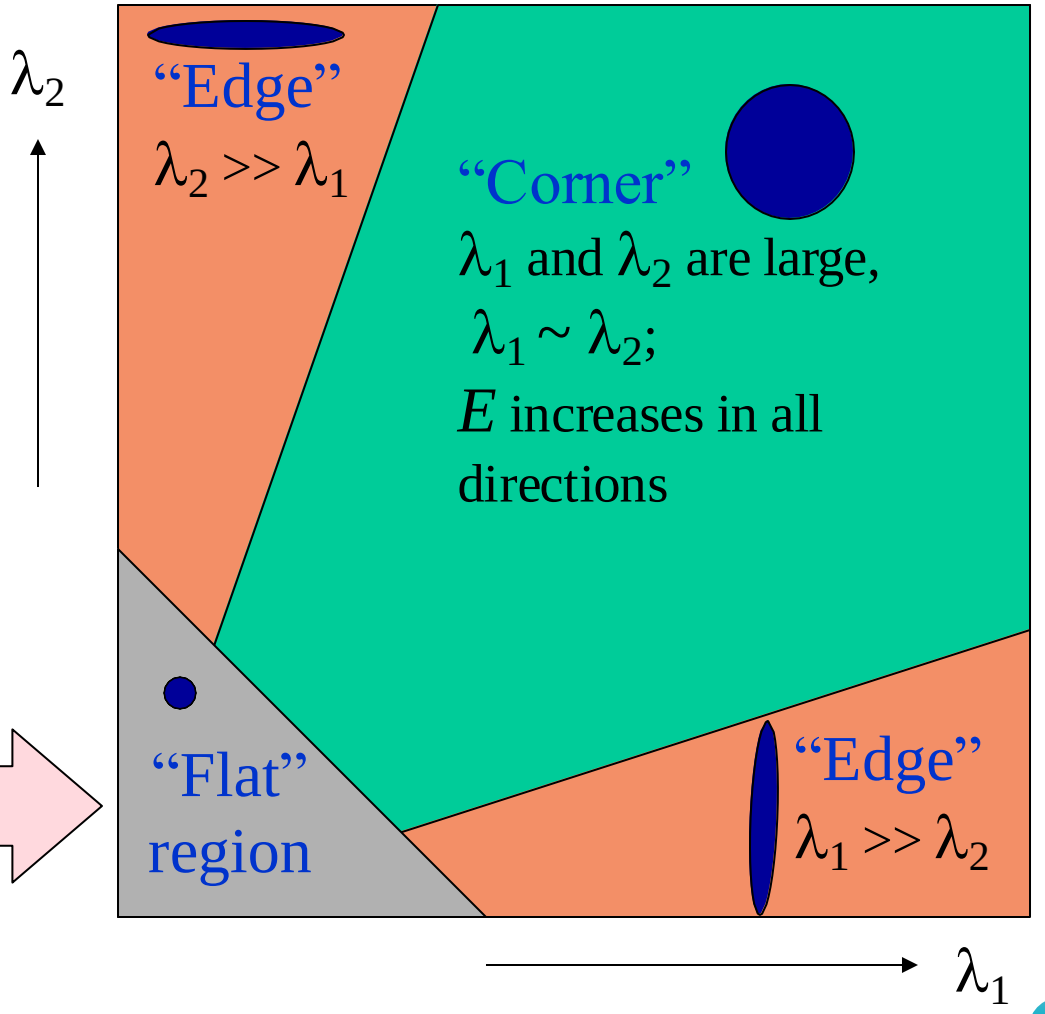
Classification of image points using eigenvalues of M:







λ_1 and λ_2 are small; E is almost constant in all directions



Harris Corner Detection :
Interpreting the Corner response function

$$R = \det(M) - \alpha\,\mathrm{trace}(M)^2 = \lambda_1\lambda_2 - \alpha(\lambda_1 + \lambda_2)^2$$

Cornerness score:

$$\boldsymbol{R = \lambda_1\lambda_2 - \alpha(\lambda_1 + \lambda_2)^2}$$

α: some constant (~0.04 to 0.06)

Remember your linear algebra:

Determinant:

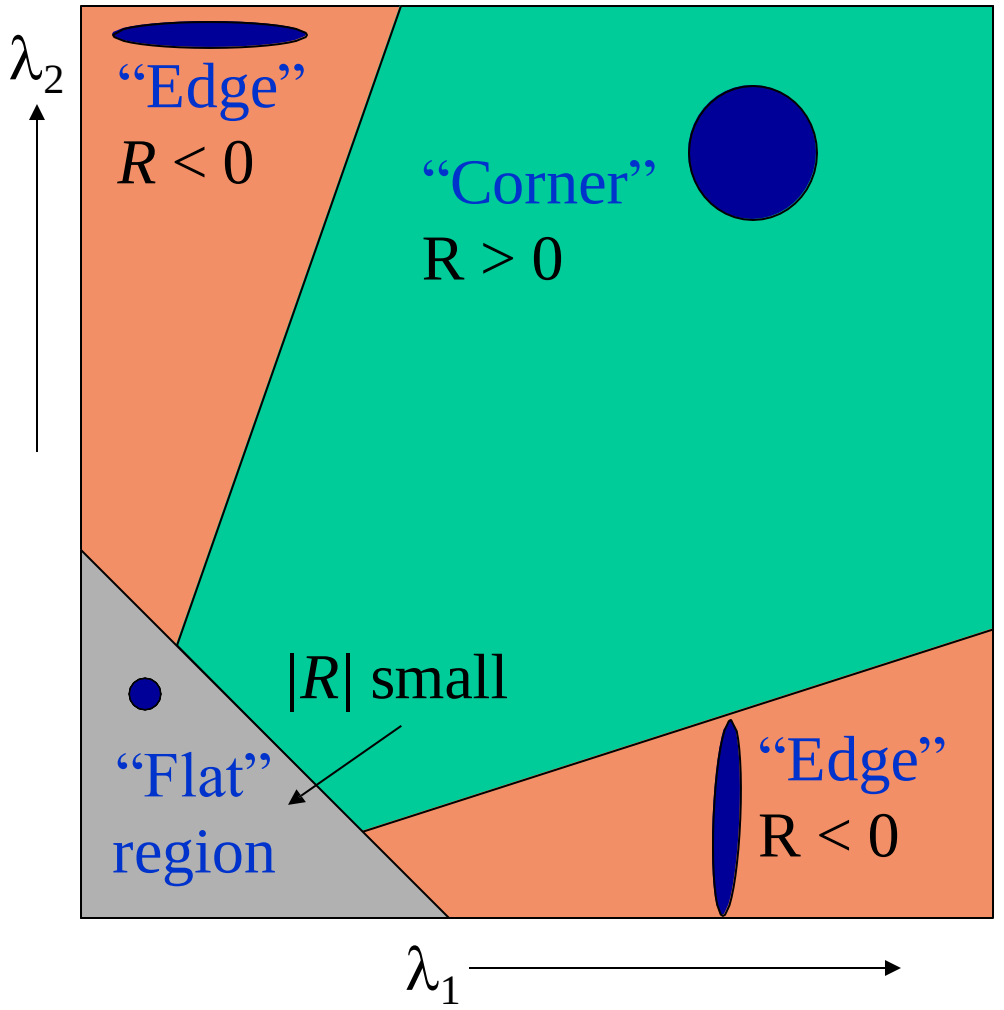
- Determinant of a diagonal matrix is the *product* of all eigenvalues.
- A is diagonal. $\det(A) = \prod_{i=1}^{n} \lambda_i = \lambda_1 \lambda_2 \cdots \lambda_n$.

Trace:

- Trace of a square matrix is the *sum* of its diagonal entries; and is the sum of its eigenvalues. $\mathrm{tr}(A) = \sum_i \lambda_i$.

$$\det \mathbf{M} = \lambda_1\lambda_2 \longrightarrow \det \mathbf{M} = m_{11}m_{22} - m_{21}m_{22}$$

$$\mathrm{tr} = \lambda_1 + \lambda_2 \longrightarrow \mathrm{tr}\,\mathbf{M} = m_{11} + m_{22}$$



Harris Corner Detector

- Harris algorithm utilizes gradient magnitudes to find and detect corners in images.

Algorithm (main steps)

1) Compute M matrix for each pixel to recover cornerness score C.
   Note: We can find M purely from the per-pixel image derivatives!

2) Threshold to find pixels which give large corner response
   (C > threshold).

3) Find the local maxima pixels, i.e., non-maxima suppression.

Next → the details to compute M and R score.

C.Harris and M.Stephens. "A Combined Corner and Edge Detector." *Proceedings of the 4th Alvey Vision Conference*: pages 147—151, 1988.

32

Harris Corner Detector Summary

Algorithm

1. Compute image discrete first derivatives.
2. Compute products of derivates to use in M
3. Create M via window function: Gaussian filter g() with σ

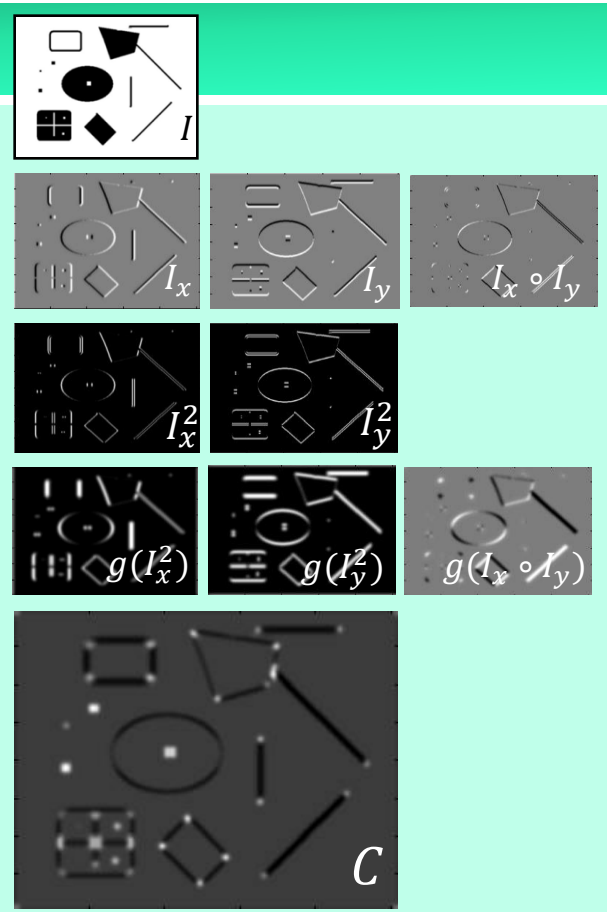
$$g(I_x^2), g(I_y^2), g(I_x \circ I_y)$$

4. Compute cornerness

$$R = \det(M) - \alpha \operatorname{trace}(M)^2 = g(I_x^2) \circ g(I_y^2) - g(I_x \circ I_y)^2 - \alpha\left[g(I_x^2) + g(I_y^2)\right]^2$$

5. Threshold on R to pick high cornerness
6. Non-maximal suppression to pick peaks.

Reminder: $a \circ b$ is element-wise multiplication



Output: A set of **corner points** that can be used for object recognition, tracking, and image matching

Harris (1988).

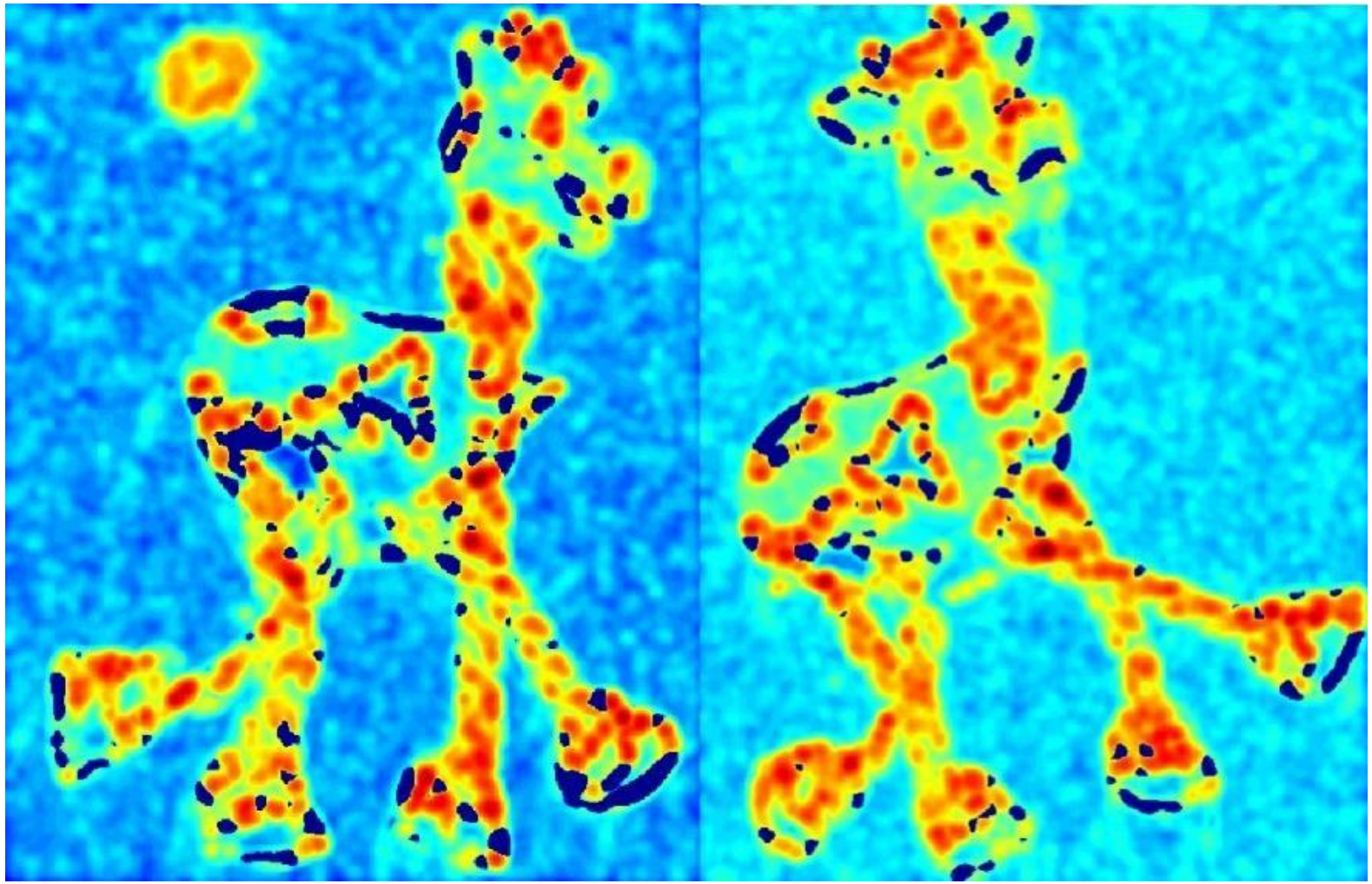
33

Harris Corner Detector: Steps

34

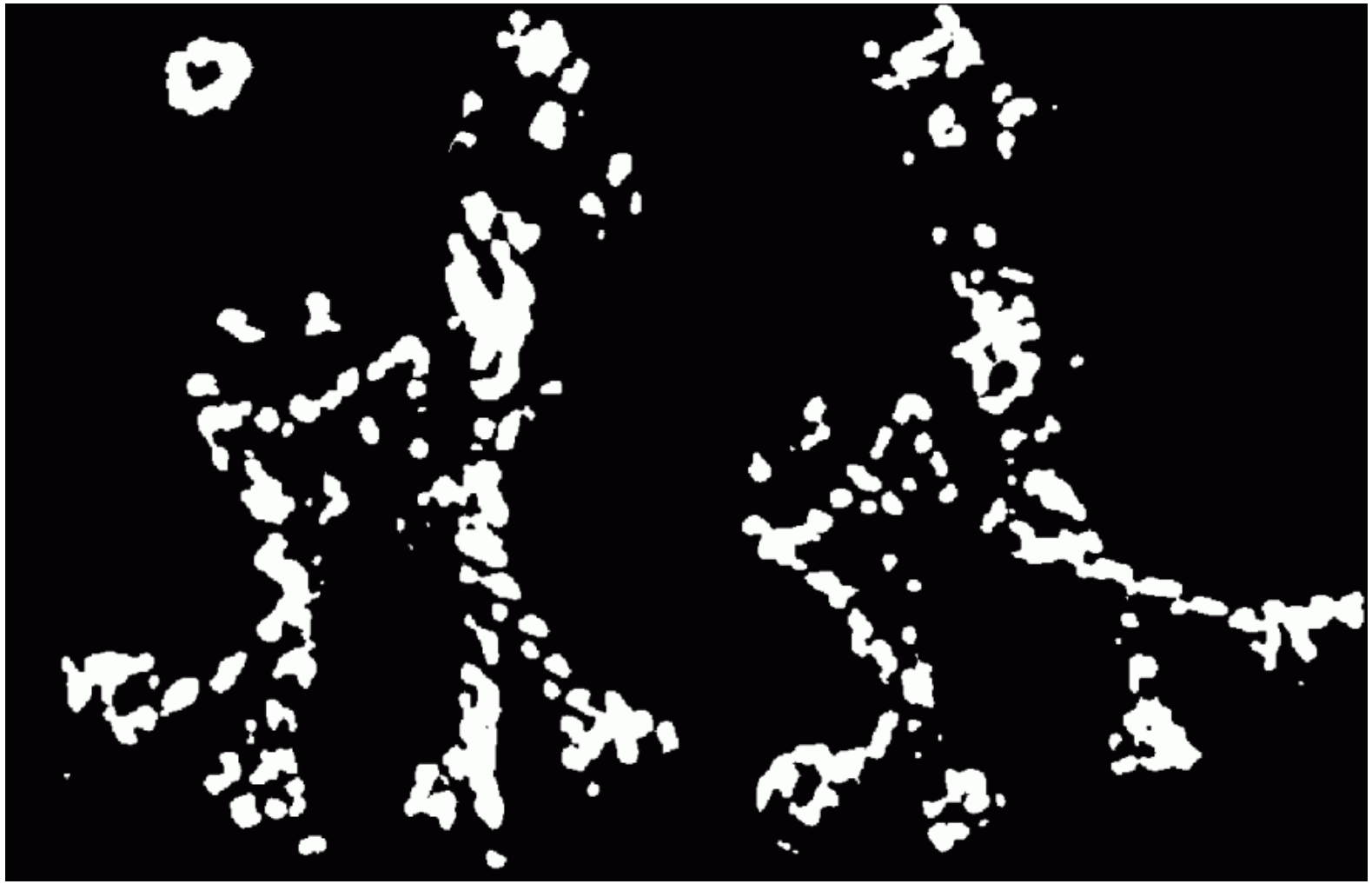
Harris Corner Detector: Steps

Compute corner response R



Harris Corner Detector: Steps

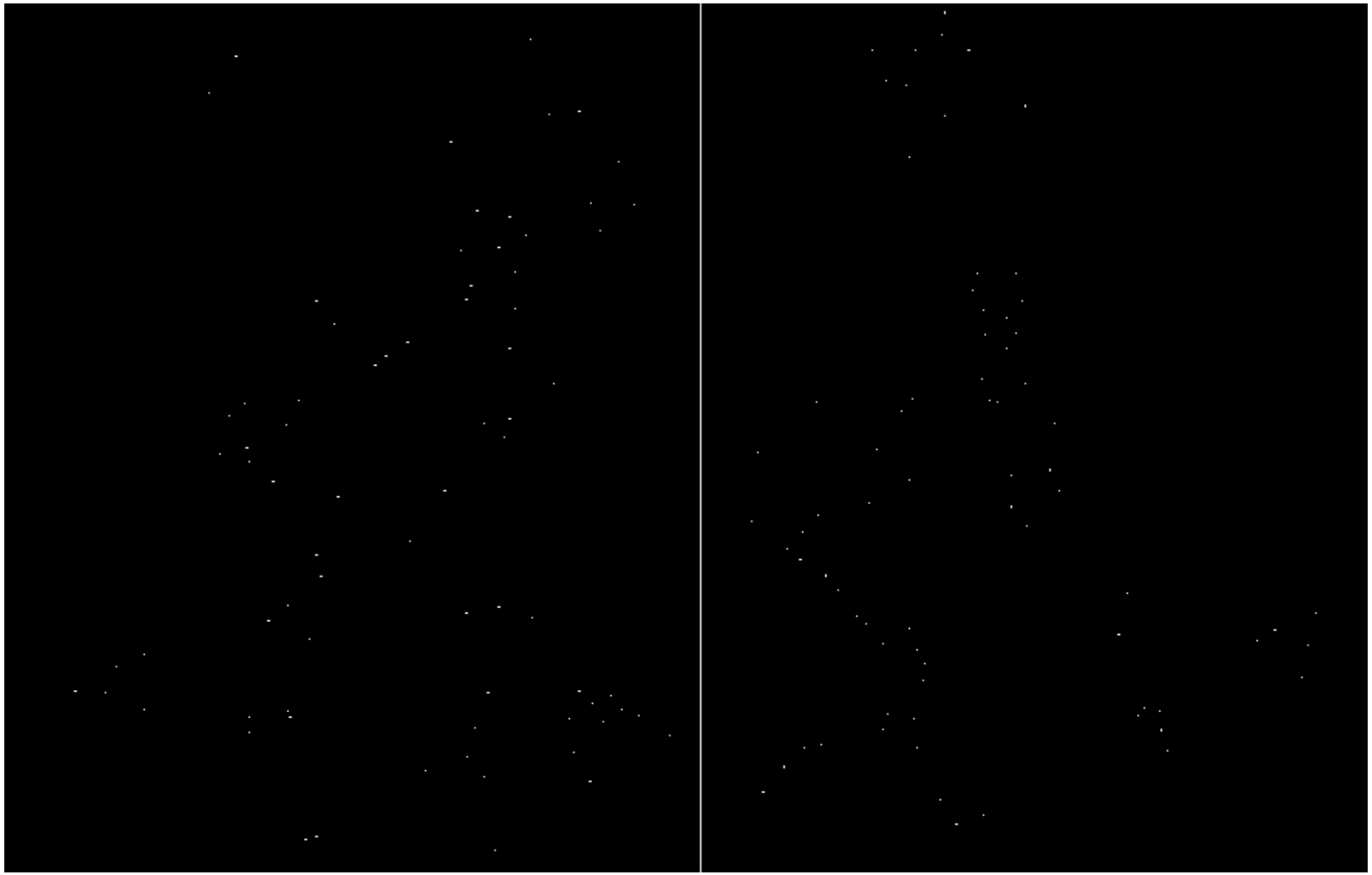
Find points with large corner response: R>threshold



36

Harris Corner Detector: Steps

Take only the points of local maxima of R



37

Harris Corner Detector: Steps

38

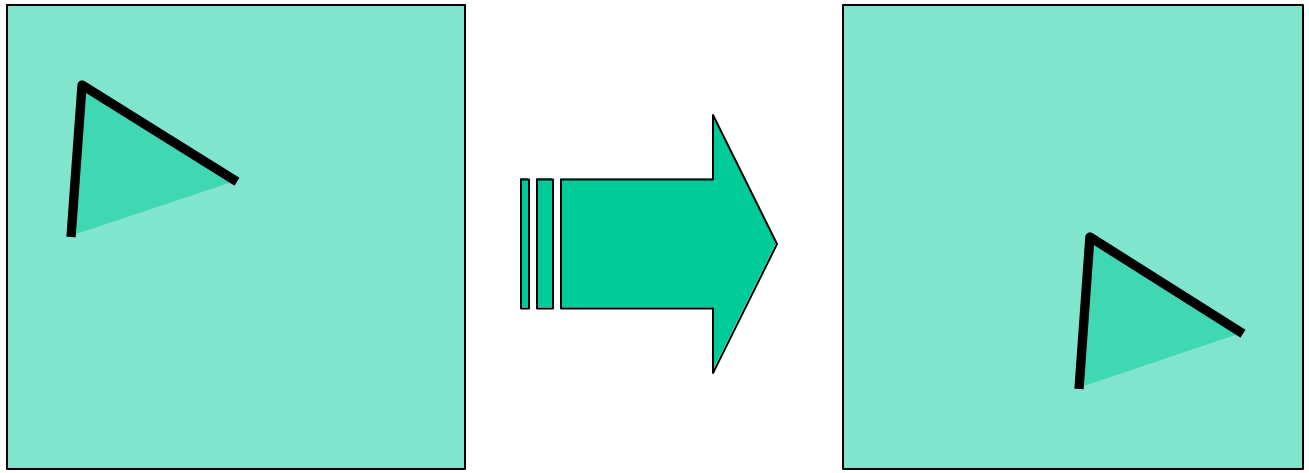
Invariance and Covariance

Are locations *invariant* to photometric transformations and *covariant* to geometric transformations?

- **Invariance: image is transformed and corner locations do not change**
- **Covariance / Equivariance: if we have two transformed versions of the same image, features should be detected in corresponding locations**

39

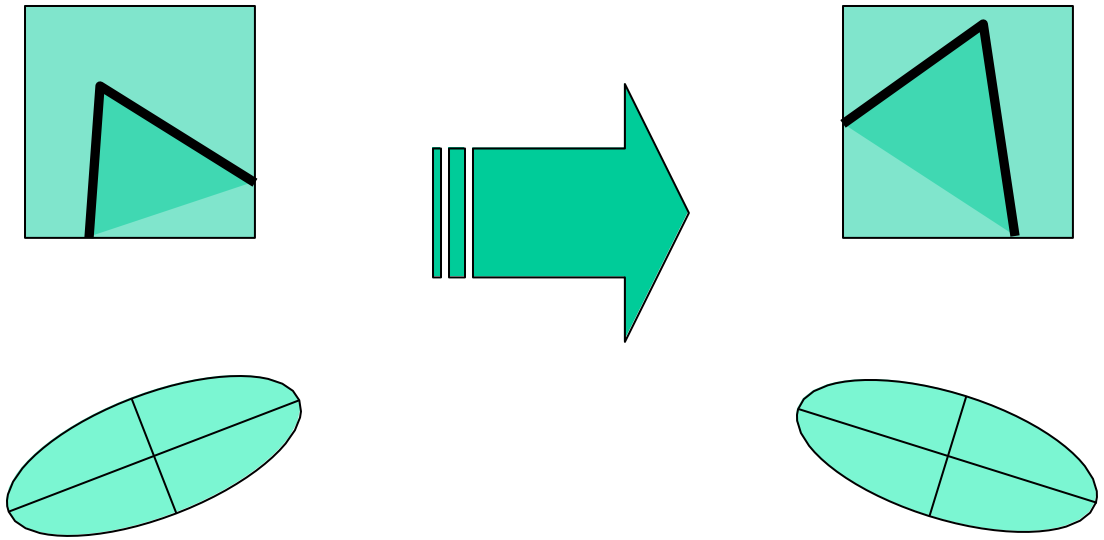
Image Translation



- Derivatives and window function are shift-invariant

Corner location is covariant w.r.t. translation

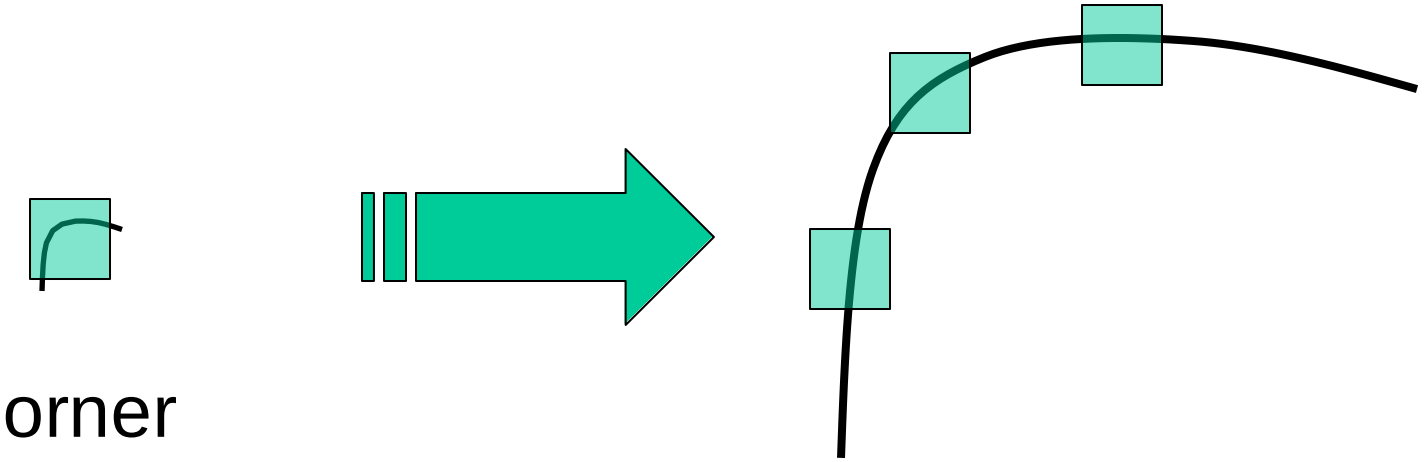
Image Rotation



Second moment ellipse rotates but its shape (i.e. eigenvalues) remains the same

Corner location is covariant w.r.t. rotation

Image Scaling



Corner

All points will be classified as edges

Corner location is not covariant to scaling!

Exercise for Corner Detection

Compute the Second Moment Matrix M

Question 1:

Given the gradient values in x, and y directions for a 3*3 window of an image as shown in below, compute the autocorrelation matrix used in Harris's corner detector.

$$I_x = \begin{bmatrix} 0 & 0 & 0 \\ -0.2 & 0.15 & -0.2 \\ 0.2 & 0.02 & -0.01 \end{bmatrix}, \quad I_y = \begin{bmatrix} 0 & 0 & 0 \\ 0.02 & 0.2 & 0.02 \\ 0.15 & 0.05 & 0.15 \end{bmatrix}$$

Answer:

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

$$I_x{}^2 = (-0.2)^2 + 0.15^2 + (-0.2)^2 + 0.2^2 + 0.02^2 + (-0.01)^2 = 0.143$$

$$I_y{}^2 = 0.02^2 + 0.2^2 + 0.02^2 + 0.15^2 + 0.05^2 + 0.15^2 = 0.0883$$

$$I_x I_y = (-0.2 * 0.02) + (0.15 * 0.2) + (-0.2 * 0.02) + (0.2 * 0.15) + (0.02 * 0.05) + (-0.01 * 0.15)$$
$$= 0.0515$$

43

Exercise for Corner Detection

Compute R Score

Question 2:

A) For each of the following diagonalized autocorrelation matrices, compute Harris' cornerness scores and determine whether there is a corner, a vertical edge, a horizontal edge, or none of them existing. Consider the threshold for corner response to be 10.

$$M_1=\begin{bmatrix}9 & 0\\ 0 & 8\end{bmatrix}, M_2=\begin{bmatrix}10 & 0\\ 0 & 0.5\end{bmatrix}, M_3=\begin{bmatrix}0.02 & 0\\ 0 & 0.05\end{bmatrix}, M_3=\begin{bmatrix}0.1 & 0\\ 0 & 20\end{bmatrix}$$

Answer:

$$R=\lambda_1\lambda_2-k(\lambda_1+\lambda_2)^2 \quad , k=0.04$$

$$R_1=9*8-0.04(9+8)^2=0.59$$ *Horizontal edge*

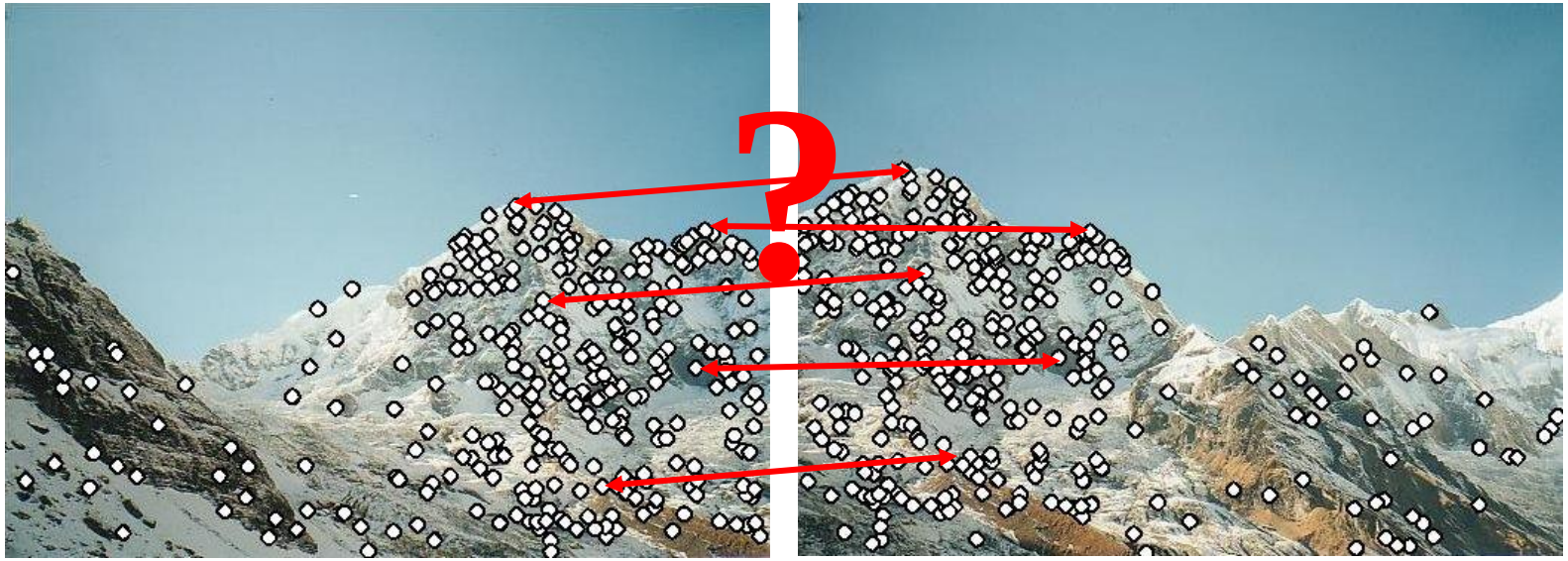
$$R_2=10*0.5-0.04(10+0.5)^2=37.16$$ *Corner*

$$R_3=0.02*0.05-0.04(0.02+0.05)^2=0.000804$$ *None of them exists*

$$R_4=0.1*20-0.04(0.1+20)^2=-14.16$$ *Vertical edge*

Feature Descriptors

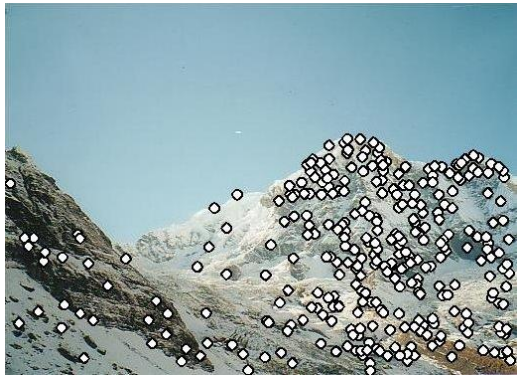
We know how to detect good points
Next question: **How to match them?**



Lots of possibilities (this is a popular research area)

- Simple option: match square windows around the point
- State of the art approach: SIFT
  - David Lowe, UBC http://www.cs.ubc.ca/~lowe/keypoints/

Local Feature: Main Components

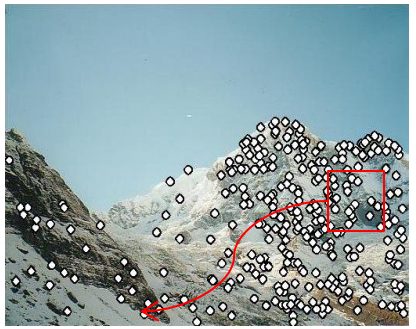


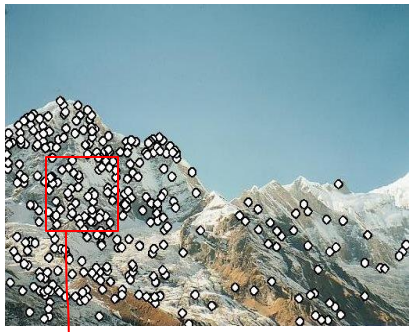
1) Detection:

Identify the interest points (Find a set of distinctive features).

2) Description:

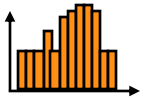
Extract vector feature descriptor surrounding each interest point.

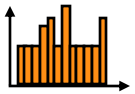




3) Matching:

Determine correspondence between descriptors in two views by Computing distance between feature vectors

x_1  $\mathbf{x}_1 = [x_1^{(1)}, \ldots, x_d^{(1)}]$

x_2  $\mathbf{x}_2 = [x_1^{(2)}, \ldots, x_d^{(2)}]$

Recap: Interesting Points (Keypoints)

Interest point:

- ✓ Has rich image content (brightness variation, color variation, etc.) within the local window.
- ✓ Has a well-defined representation (signature) for matching with other points.
- ✓ Has a well-defined position in the image.

Feature detectors should be invariant or at least robust to:

- Affine changes
- Translation
- Rotation
- Scale change

Suppose we are comparing two images I_1 and I_2

- I_2 may be a transformed version of I_1
- What kinds of **transformations** are we likely to encounter in practice?

We'd like to find the same features regardless of the transformation

- This is called transformational ***invariance***
- They can usually also handle
  - Limited 3D rotations (SIFT works up to about 60 degrees)
  - Limited affine transformations (some are fully affine invariant)
  - Limited illumination/contrast changes

How to achieve invariance?

Need both of the following:

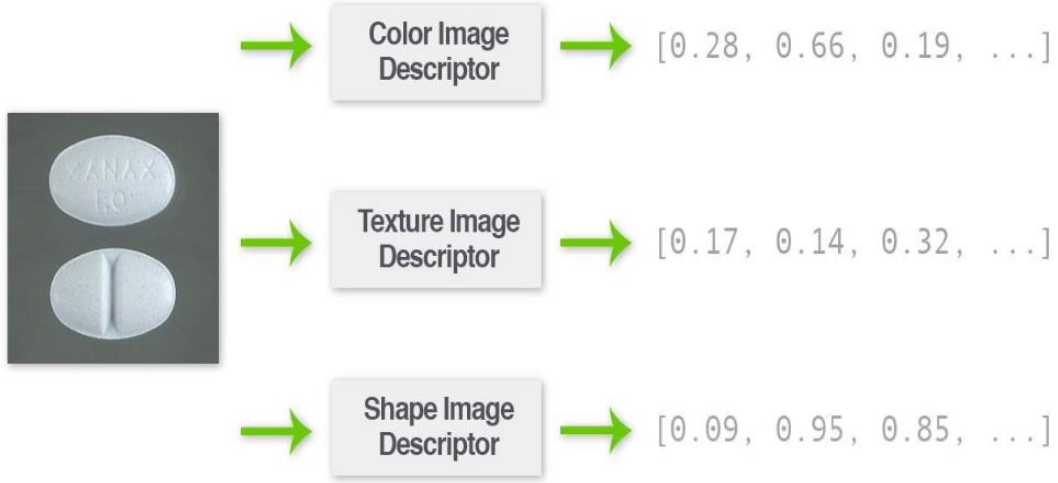
1. **Make sure your detector is invariant**
   - **Harris** is invariant to translation and rotation
   - Scale is trickier
     - common approach is to detect features at many scales using a Gaussian pyramid (e.g., MOPS)
     - More sophisticated methods find "the best scale" to represent each feature (e.g., SIFT)
2. **Design an invariant feature *descriptor***
   - A descriptor captures the information in a region around the detected feature point.

Feature Descriptor

It is a simplified representation of the image that contains only the most important information about the image.

- ***Image descriptor*** *is an algorithm and methodology that governs how an input image is* ***globally*** *quantified and returns a feature vector abstractly representing the image contents.*
- ***feature descriptors*** *is an algorithm and methodology that governs how an input* ***region of an image*** *is* ***locally*** *quantified. A feature descriptor accepts a* single *input image and returns* multiple *feature vectors.*
- ***feature vectors*** are the **output of descriptors** and used to quantify the image.

This process is called ***feature extraction.***



Feature Descriptor

There are several feature descriptors out there.

- **Templates**
  - Intensity, gradients, etc.
- **Histograms**
  - **SIFT: Scale Invariant Feature Transform**
  - **HOG: Histogram of Oriented Gradients**
  - SURF: Speeded-Up Robust Feature

We could measure the image intensity values around the key points to get unique signatures, but we'll focus on **histograms** in this class.

Feature Descriptor: SIFT

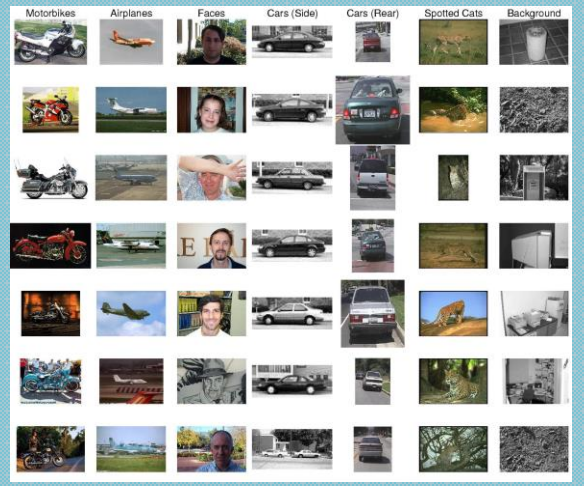
SIFT builds on corner detection by finding **and describing keypoints** for **matching and recognition**.

SIFT: Scale Invariant Feature Transform

Extremely popular (67k citations in 2022)

SIFT Applications:

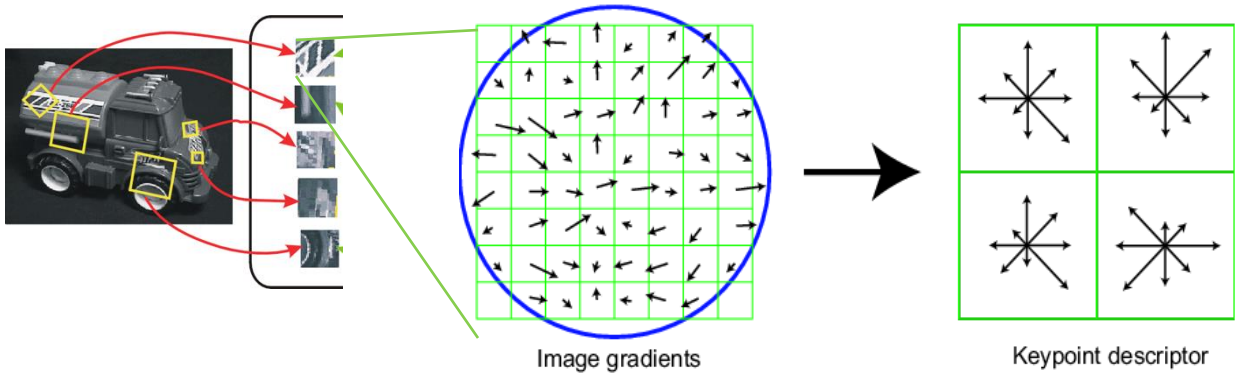
- Object recognition.
- Object categorization.
- Location recognition.
- Robot localization.
- Image retrieval.



SIFT Algorithm

- **Scale Invariant Feature Transform (SIFT)** is a computer vision technique used for feature extractor and object recognition pipeline which is built around using keypoints as features to detect objects regardless of scale.
- SIFT works by identifying keypoints based on their local intensity extrema and computing descriptors that capture the local image information around those keypoints.

1. **Detection (**SIFT Detector**)**
   - Detect points that can be repeatably selected under location/scale change.
2. **Description (**SIFT Descriptor**)**
   - Assign orientation to detected feature points.
   - Construct a descriptor for image patch around each feature point.
3. **Matching**



SIFT Algorithm

Algorithm

1. **Find Harris corners scale-space** extrema as feature point locations

2. **Corner post-processing**
   a. Subpixel position interpolation
   b. Discard low-contrast points
   c. Eliminate points along edges

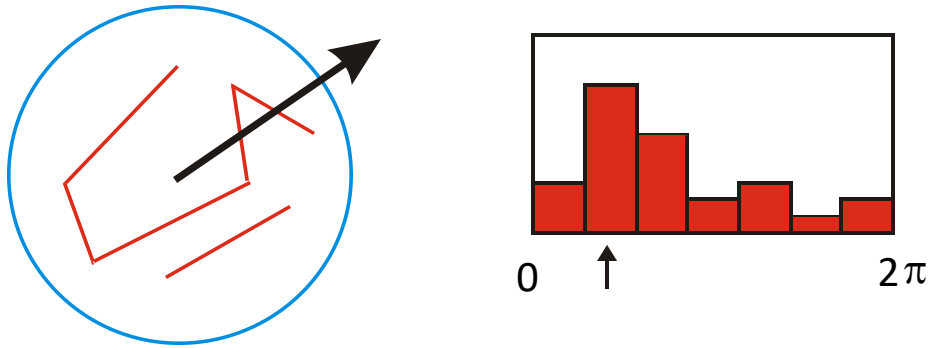
(SIFT Descriptor)

1. Orientation estimation per feature point

2. **Descriptor extraction**
   a. Motivation: We want some sensitivity to spatial layout, but not too much, so blocks of histograms give us that.

3. Orientation Estimation

- Compute gradient orientation histogram.
- Select dominant orientation θ.
- To build this histogram,
  - we bin the possible range of gradient orientations/directions (0 to 2pi radians) into a discrete number of bins (8 here).
  - and, for each bin, we count the number of gradients that have those corresponding orientations.
  - We also compute an average gradient direction (black vector) of all the gradients in the local window; this is the 'dominant orientation' of this neighborhood.

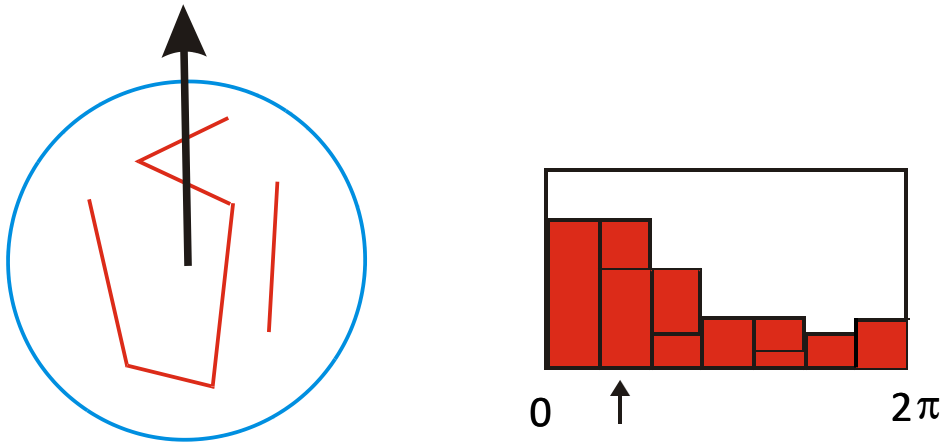


[Lowe, SIFT, 1999] T. Tuytelaars, B. Leibe

3. Orientation Estimation

- Compute gradient orientation histogram.
- Select dominant orientation θ.
- Normalize: rotate to fixed orientation.

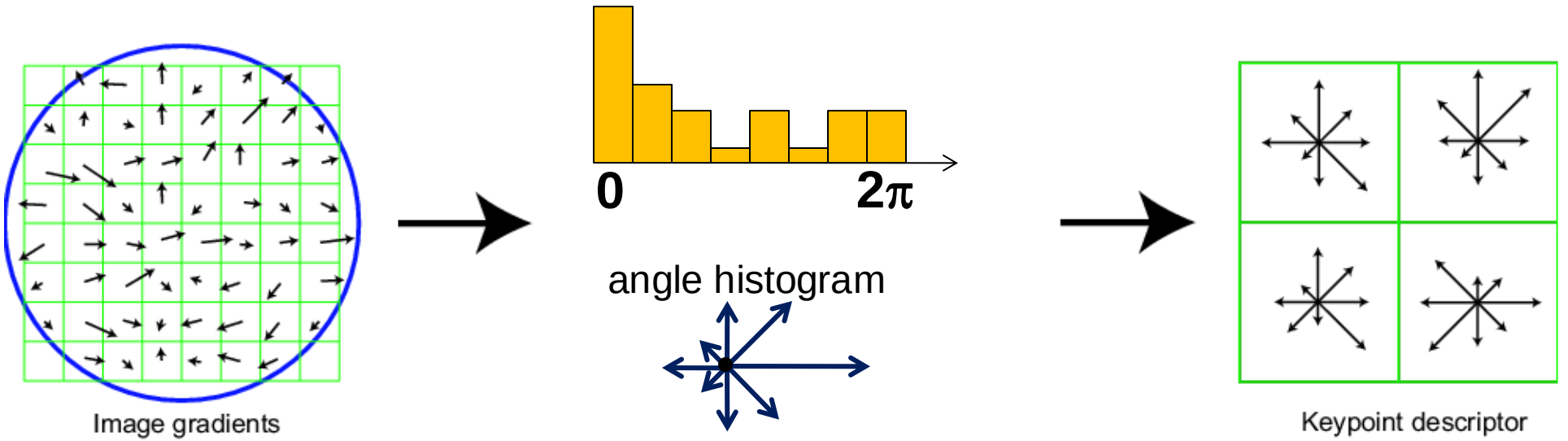
(rotate each region's gradients so that all dominant gradient orientations point in the same direction (e.g. up). This will affect the orientation histogram by shifting the plot. We do this because we want the key point features to be invariant to rotation)



[Lowe, SIFT, 1999] T. Tuytelaars, B. Leibe

4. SIFT descriptor: basic idea

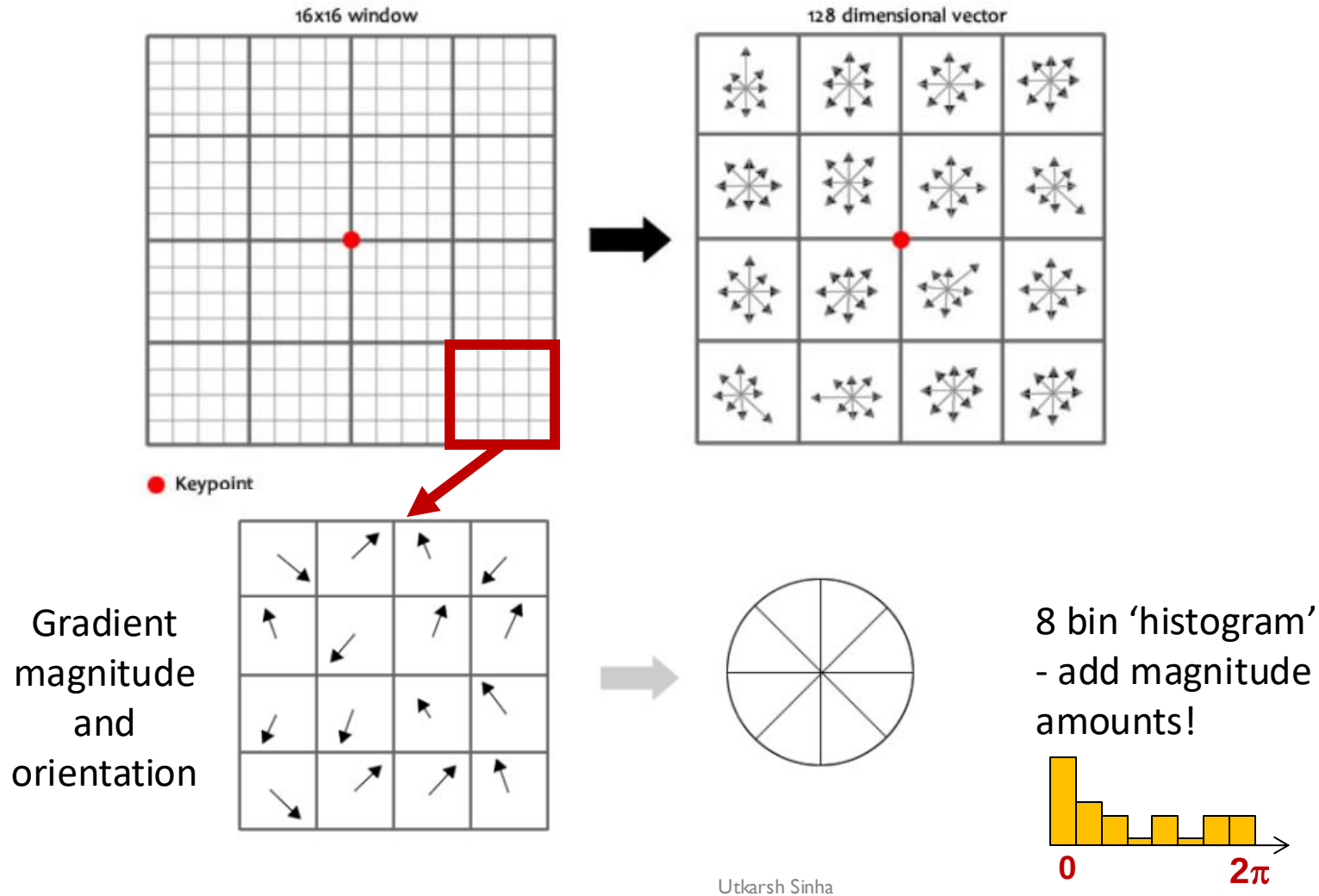
1. Take 16x16 square window around detected feature.
2. Compute edge gradient: orientation (angle of the gradient - 90°), and magnitude for each pixel.
3. Then subdivide the 16x16 window into a 4x4 grid of cells. *(2x2 case shown below)*
4. Throw out weak edges (threshold gradient magnitude)
5. Create histogram of surviving edge orientation for each block.
6. Concatenate 4 x 4 (16 cells) * 8-bin histograms (orientation) into 128-dimension descriptor



Adapted from slide by David Lowe

SIFT Descriptor Extraction

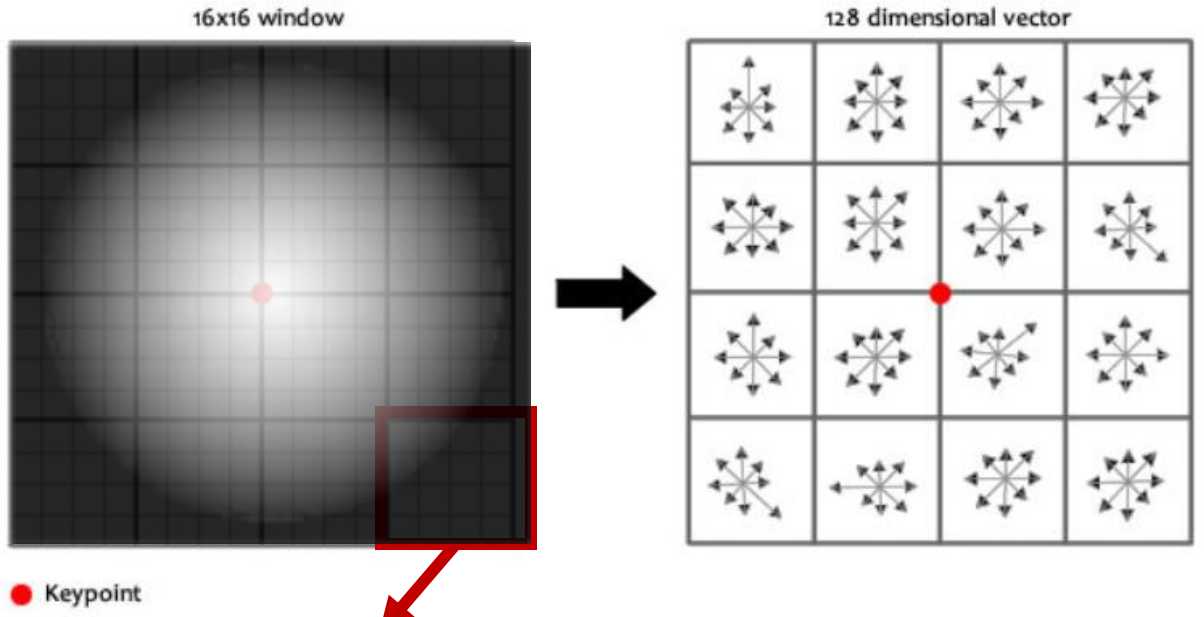
Here we show the process for extracting the **SIFT descriptor** for a single keypoint (red) in the image, which is given with **scale** and **orientation**.



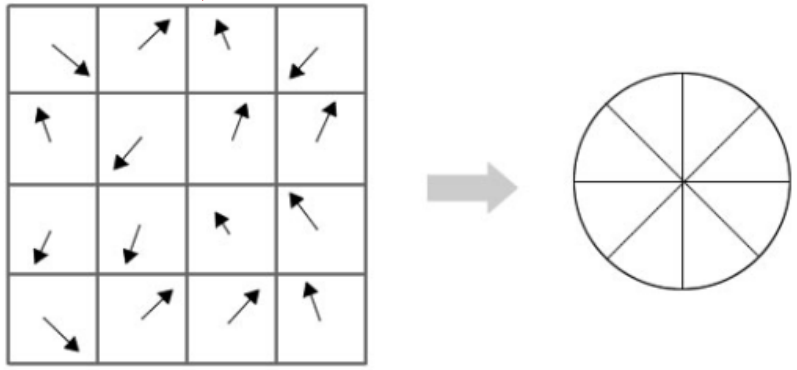
SIFT Descriptor Extraction

Weight 16x16 grid by Gaussian to add location robustness and reduce effect of outer regions

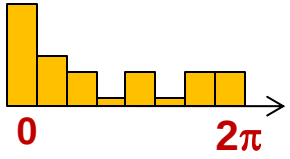
σ = half window width



Gradient magnitude and orientation



8 bin 'histogram' - add magnitude amounts!



- Finally, Concatenate 4 x 4 8-bin histograms into 128 dimension FV

Properties of SIFT

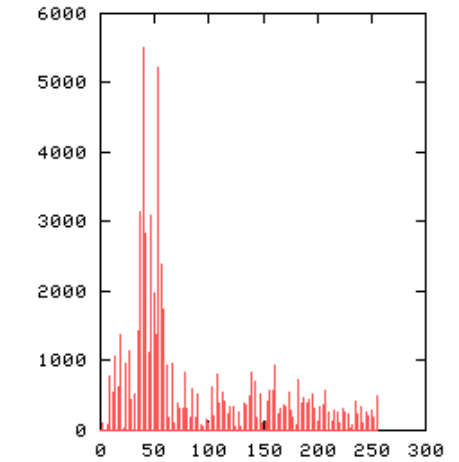
Extraordinarily robust matching technique

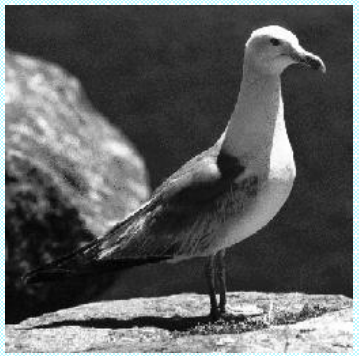
- Can handle changes in viewpoint
  - Up to about 60 degree out of plane rotation
- Can handle significant changes in illumination
  - Sometimes even day vs. night (below)
- Fast and efficient—can run in real time
- Lots of code available

Feature Descriptor: HOG

Global histogram to represent distribution of features
What about a local histogram per detected point?

Histograms





Human detection

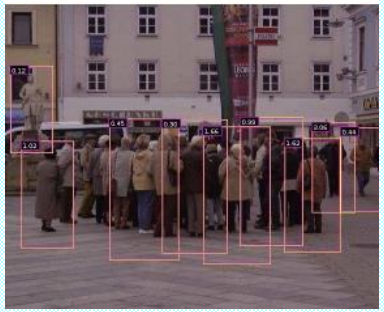
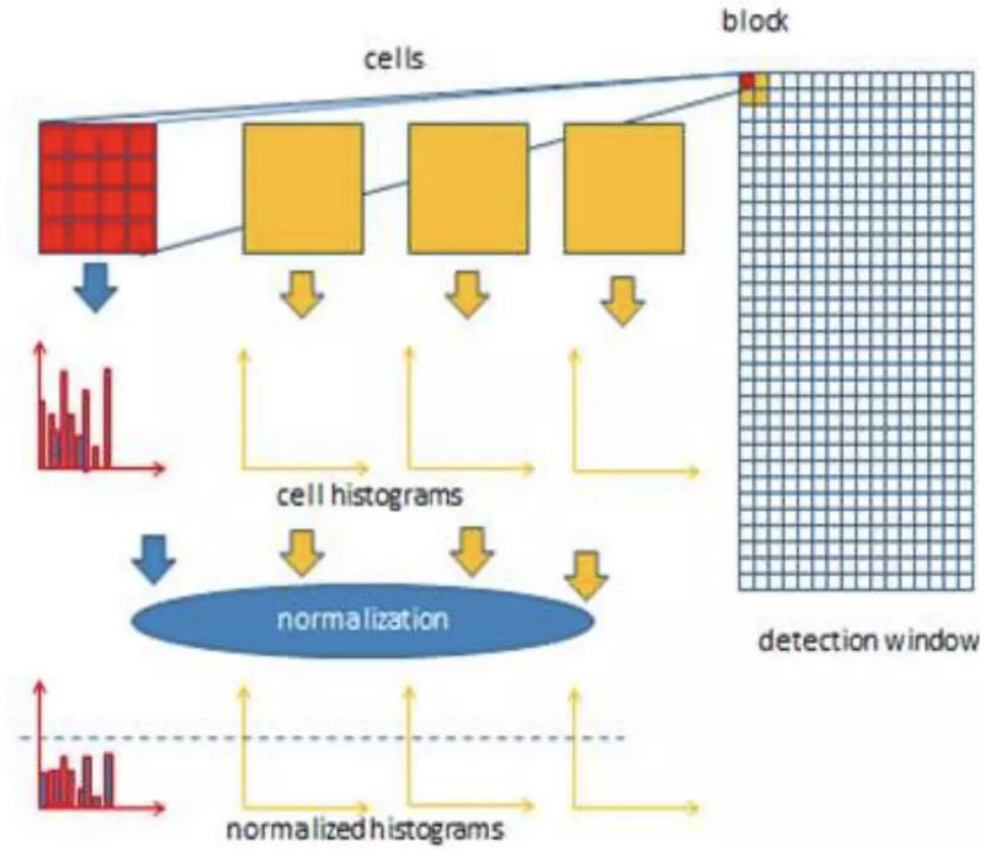


Image Credit: Pixar Animation Studios

Histogram of oriented gradients (HOG)

- The **HOG** descriptor focuses on the **structure** or the **shape** of an object.
- HOG is able to provide the edge direction by extracting the gradient magnitude and orientation of the edges. These orientations are calculated in 'localized' portions.

- ✓ **Step 1:** Pre-processing
- ✓ **Step 2:** Calculate the gradient vector of every pixel, as well as its magnitude and direction
- ✓ **Step 3:**
  a. Divide the image into blocks and cells.
  b. Calculate histogram of gradients in 8x8 cells
- ✓ **Step 4:** Normalize gradients in 16×16 cell
- ✓ **Step 5:** Form HOG Feature vector of the complete image

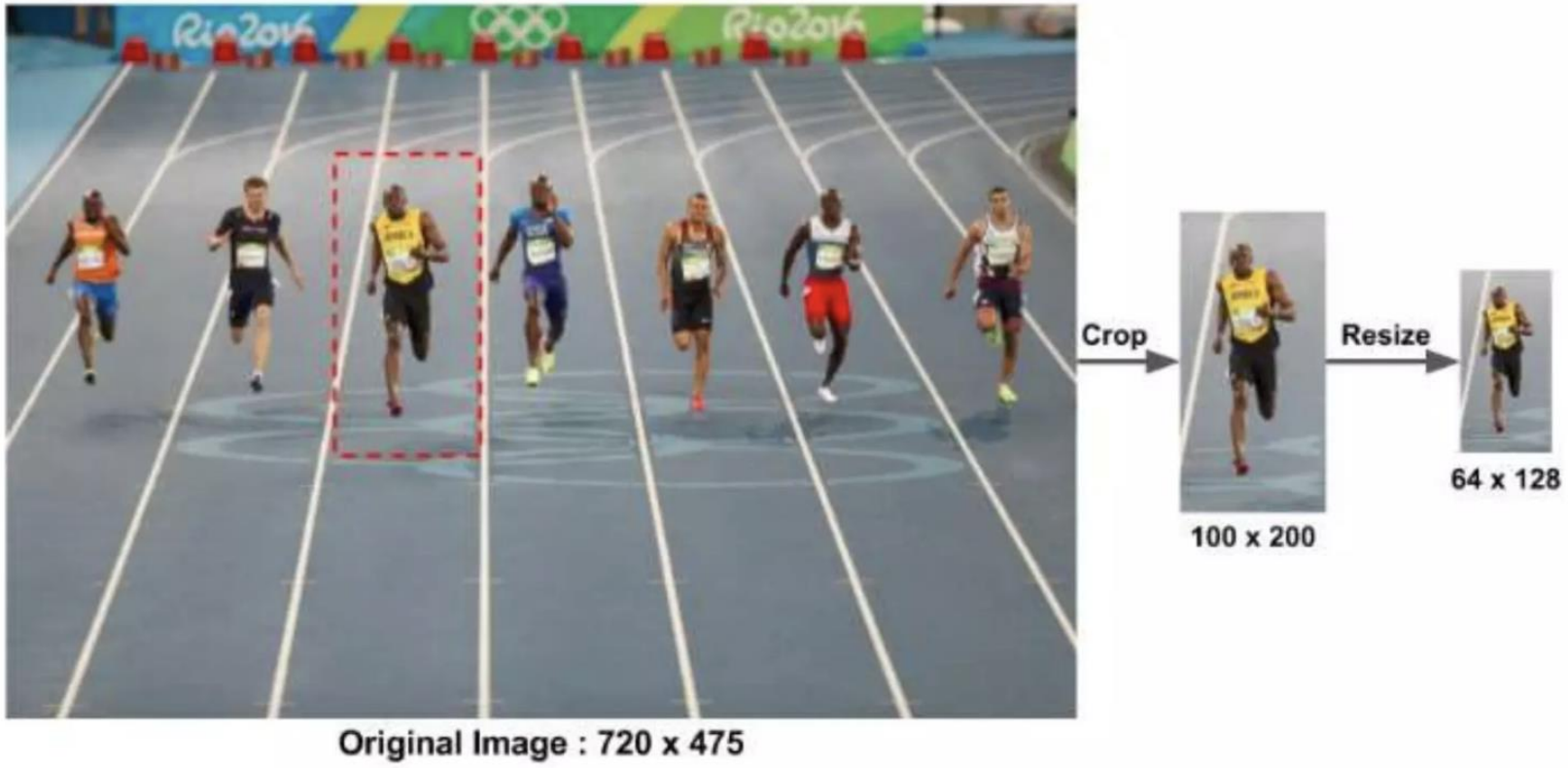


https://www.slideshare.net/AnirudhKanneganti/hog-122762396

HOG Step1: Preprocessing

- Patches being analyzed should have a fixed aspect ratio.

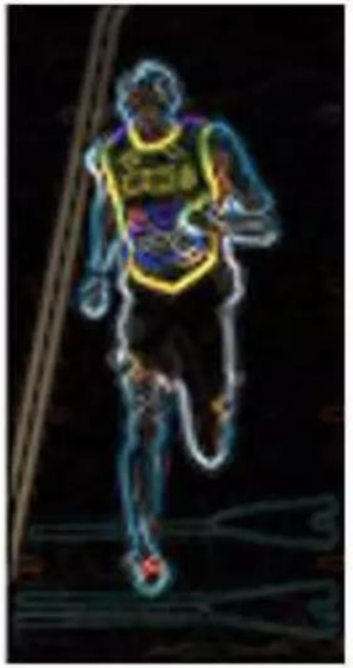
For example, they can be 100×200, 128×256, or 1000×2000 but not 101×205.



https://www.slideshare.net/AnirudhKanneganti/hog-122762396

HOG Step2: Calculate Gradient Image

- Get information on the movement of energy from left to right and up to down
- Apply 1D filter kernel



-1	0	1

-1
0
1

$$g = \sqrt{g_x^2 + g_y^2}$$

$$\theta = \arctan \frac{g_y}{g_x}$$

Left : Absolute value of x-gradient. Center : Absolute value of y-gradient.
Right : Magnitude of gradient.

https://www.slideshare.net/AnirudhKanneganti/hog-122762396

HOG Step3: Calculate histogram in 8x8 cells

- A **histogram** is a plot shows the frequency distribution of a set of continuous data.
  - We are going to take the angle or orientation on the x-axis and the magnitude on the y-axis.
- Now that we have our gradient magnitude and orientation representations, before calculating the histogram, we need first to **divide** our image up into **cells** and **blocks.**

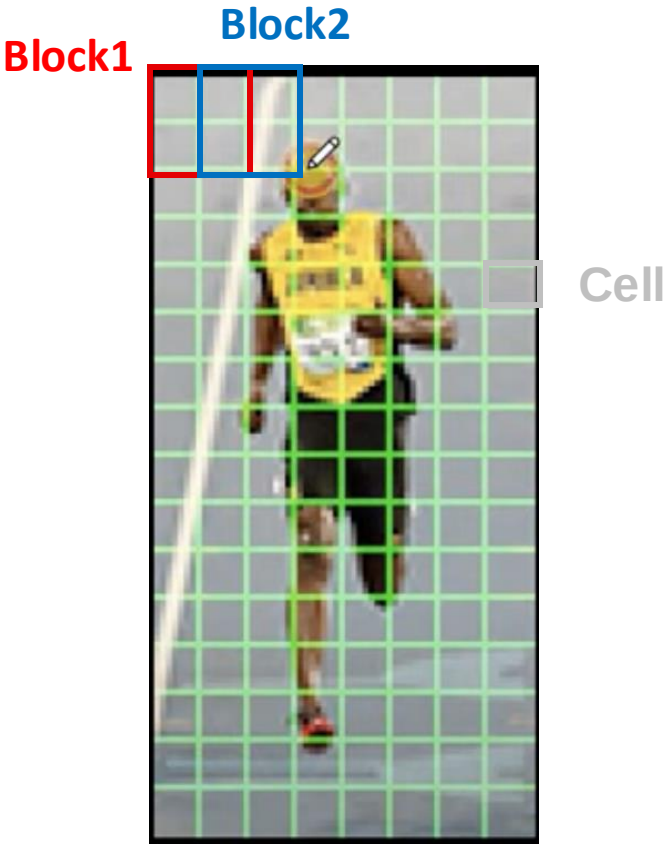
- A "**cell**" is a rectangular region defined by the number of pixels that belong in each cell.
- A "**block**" is group of cells.

✓ Each cell has 8x8 pixels.

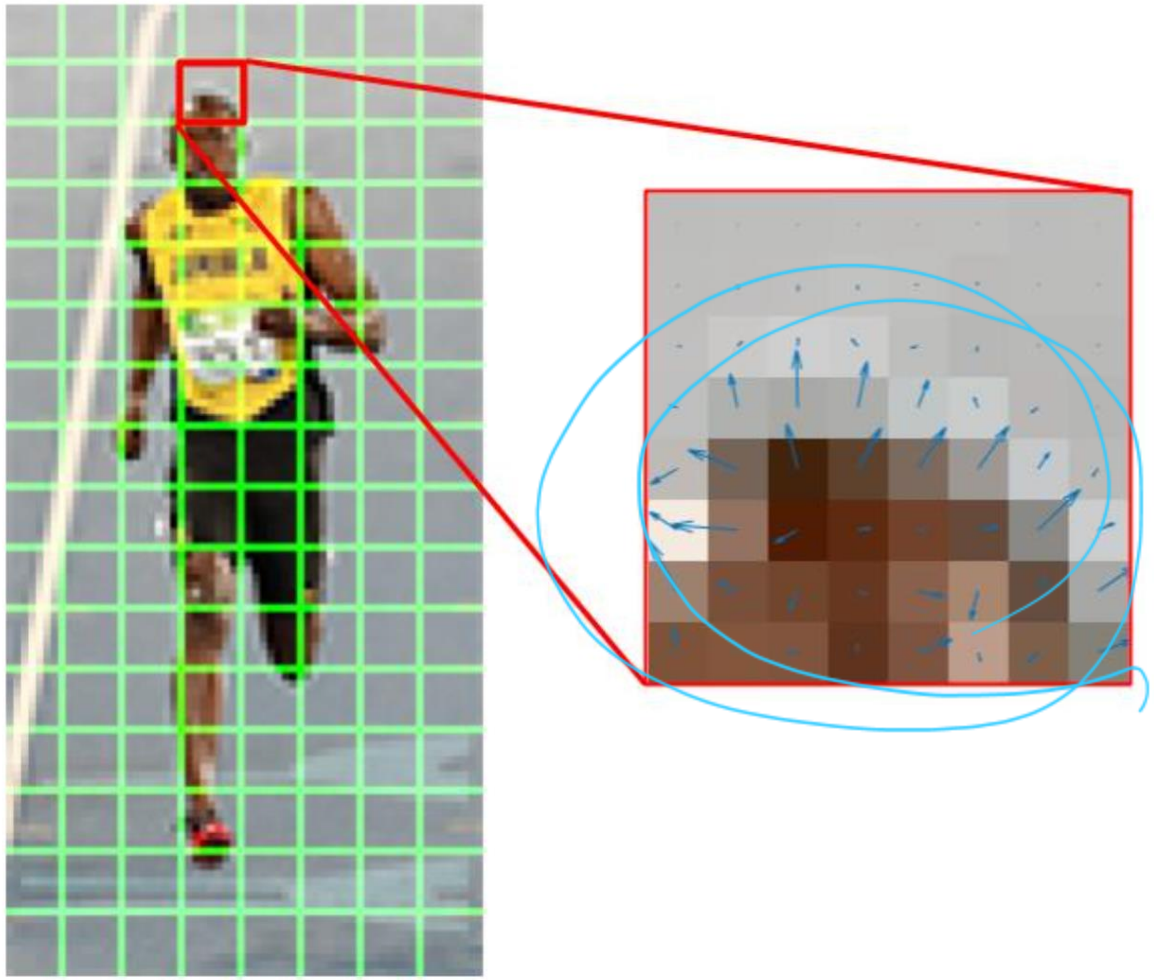
✓ Total cells are 8x16 (image size/cell size = 64x128 / 8x8).

✓ Combine each 2x2 cells into one block with 50% overlap.

✓ Total blocks are 7x15 = 105 blocks.



HOG Step3: Calculate histogram in 8x8 cells



2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

Gradient Magnitude

80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

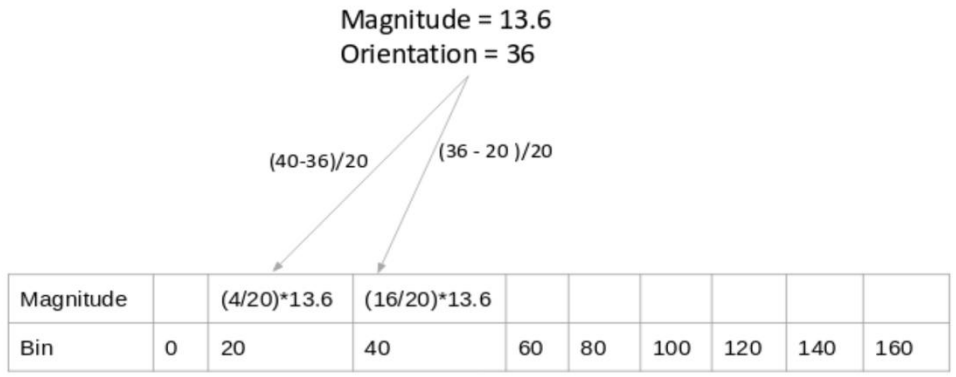
Gradient Direction

Center : The RGB patch and gradients represented using arrows.
Right : The gradients in the same patch represented as numbers

HOG Step3: Calculate histogram in 8x8 cells

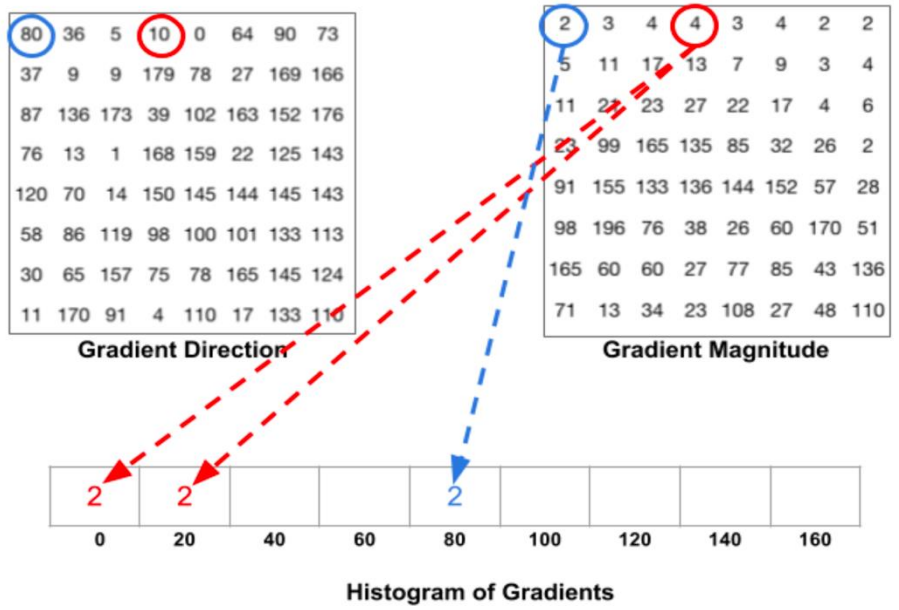
- This step is to create a histogram of gradients in these 8×8 cells. The histogram contains 9 bins corresponding to angles 0, 20, 40 … 160.
- The angles are between 0 and 180 degrees which are called **“unsigned” gradients.**

NOTE: Empirically it has been shown that unsigned gradients work better than signed gradients for pedestrian detection.

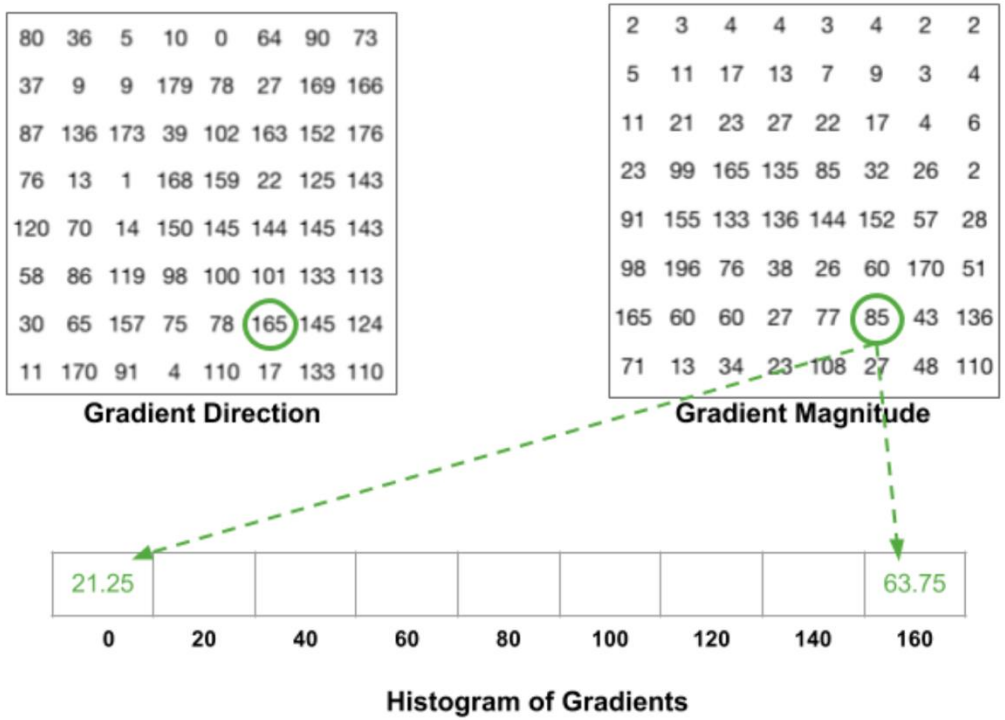


Magnitude		(4/20)*13.6	(16/20)*13.6						
Bin	0	20	40	60	80	100	120	140	160

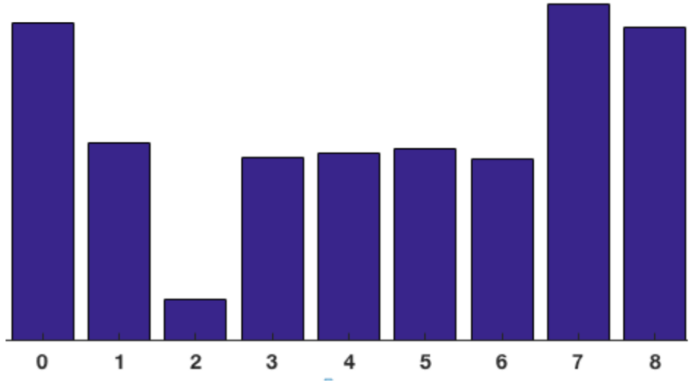
Calculating weighted votes in each bin



HOG Step3: Calculate histogram in 8x8 cells



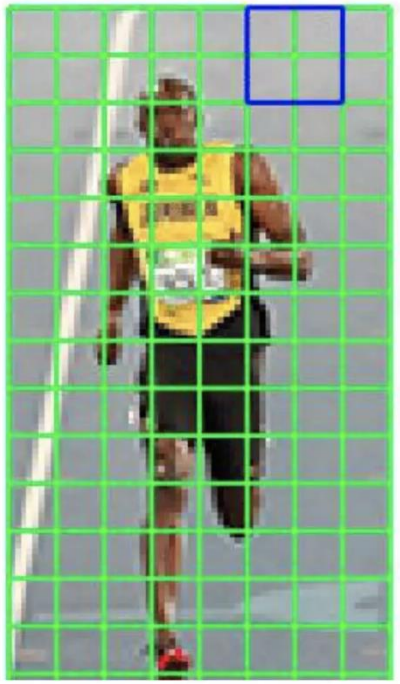
For color images, the gradients of the three channels are evaluated. The magnitude of gradient at a pixel is the maximum of the magnitude of gradients of the three channels, and the angle is the angle corresponding to the maximum gradient.



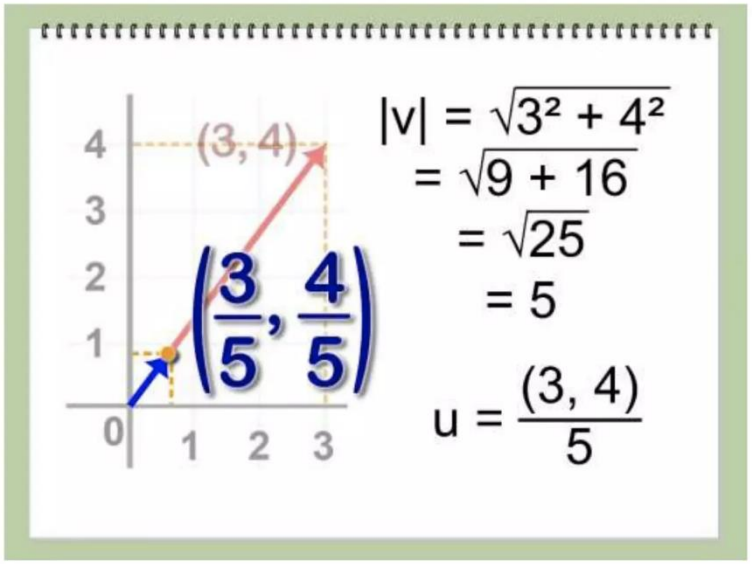
The contributions of all the pixels in the 8×8 cells are added up to create the 9-bin histogram.
For the patch in the previous slide, it looks like this.

HOG Step4: 16×16 Block Normalization

- In step3, we created a histogram based on the gradient of the image, which are sensitive to overall lighting.
  - If you make the image darker by dividing all pixel values by 2, the gradient magnitude will change by half, and therefore the histogram values will change by half.
- **We want our descriptor to be independent of lighting variations** (*i.e., we would like to "normalize" the histogram so they are not affected by lighting variations).*
- We can see that normalizing a vector removes the scale.



• A 16×16 block has 4 histograms which can be concatenated to form a 36 x 1 element vector



HOG Step4: 16×16 Block Normalization

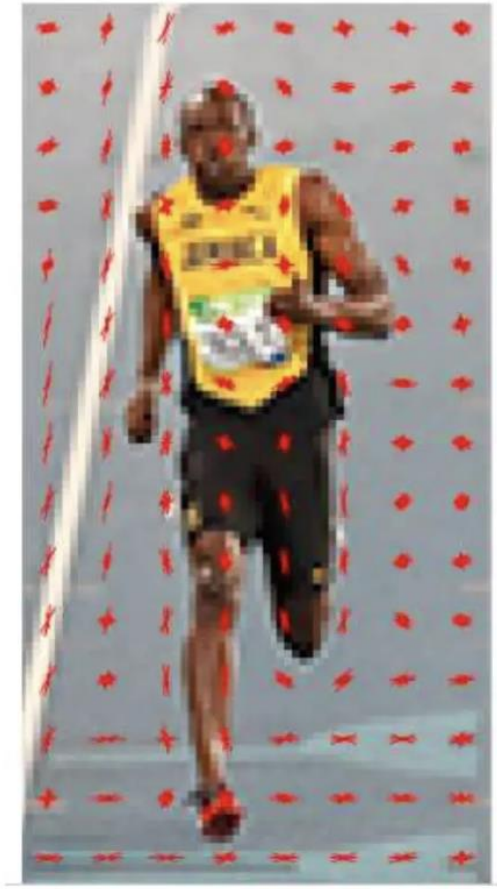
- Let's say we have an RGB color vector **[128, 64, 32].** The length of this vector is.

- This is also called the **L2 norm** of the vector. $\sqrt{128^2 + 64^2 + 32^2} = 146.64$
- Dividing each element of this vector by **146.64** gives us a normalized vector **[0.87, 0.43, 0.22].**
- Now consider another vector in which the elements are twice the value of the first vector 2 x [128, 64, 32] = [256, 128, 64].
- Now normalizing [256, 128, 64] will result in [0.87, 0.43, 0.22], which is the same as the **normalized version of the original RGB vector.**
- We can see that normalizing a vector removes the scale.

- Now that we know how to normalize a vector, We can simply normalize the 9×1 histogram the same way we normalized the 3×1 vector above.
- However, in practice, a better idea is to normalize over a bigger sized block of 16×16.
- A 16×16 block has 4 histograms of (8x8) which can be concatenated to form a 36 x 1 element vector and it can be normalized just the way a 3×1 vector is normalized.

HOG Algorithms

- To calculate the final feature vector (FV) for the entire image patch, the 36×1 vectors are concatenated into one large vector.
  - **What is the size of this vector ?**

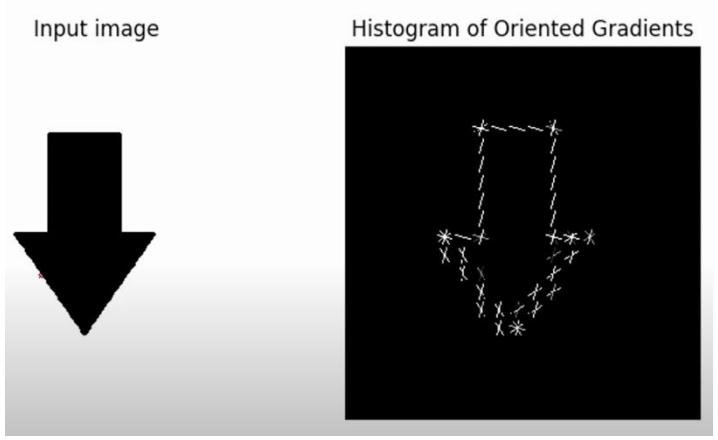


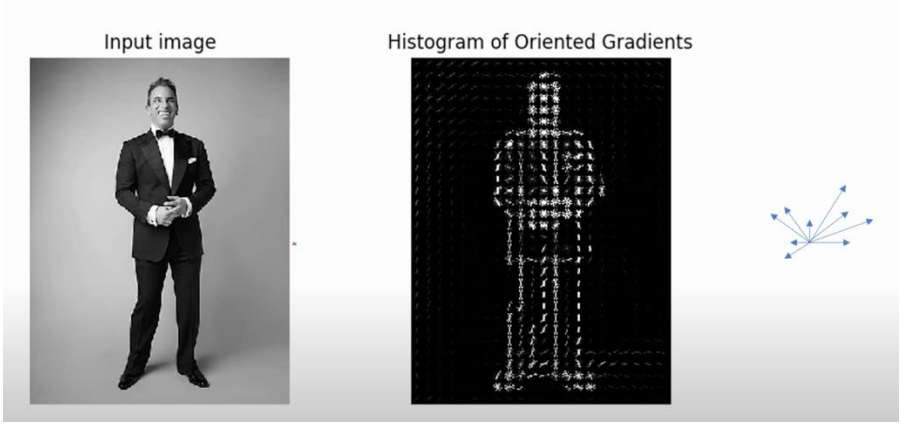
1. How many positions of the 16×16 blocks do we have ? **There are 7 horizontal and 15 vertical positions making a total of 7 x 15 = 105 positions.**

2. Each 16×16 block is represented by a 36×1 vector.

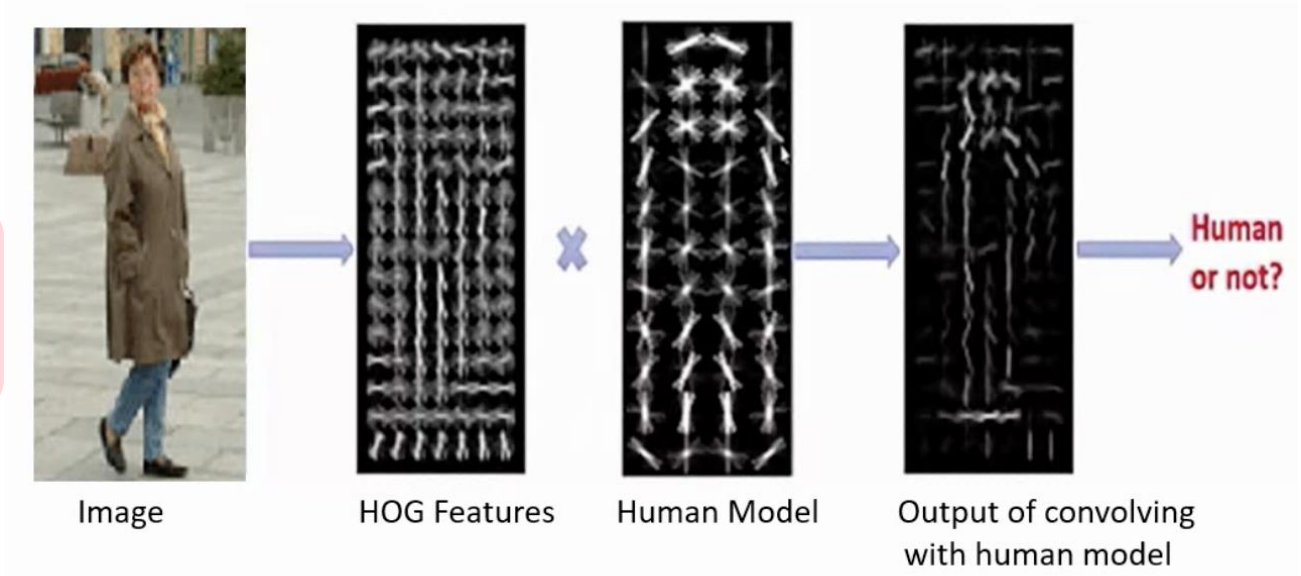
✓ So when we concatenate them all into one large vector we obtain a
36×105 = **3780** dimensional vector.

HOG Examples





Object Detection with HOG + SVM Classifier



Histograms of Oriented Gradients for Human Detection
Navneet Dalal and Bill Triggs

HOG: More illustration example

HOG Intro → Image Preprocessing → Calculate Gradient & Orientation → Build histogram → Normalisation

P32 | C8

P64

C16

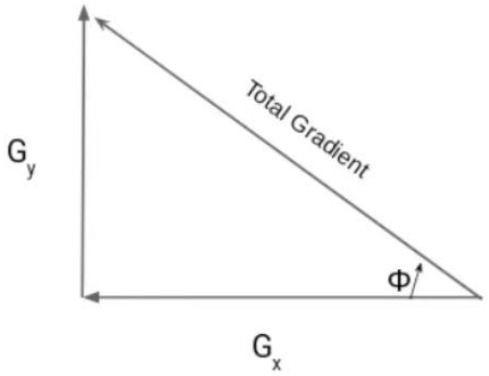
Select a 4 x 4 cell

Pixel values for selected cell

121	10	78	96
48	152	64	125
145	78	85	89
154	214	56	200

For edge pixels, HOG uses padding

From Pythagoras theorem



Gradient in x dir. (Gx)
= 89 - 78
= 11

Gradient in y dir. (Gy)
= 64 - 56
= 8

Total Gradient Magnitude
= $\sqrt{[(Gx)^2+(Gy)^2]}$
= $\sqrt{[(11)^2+(8)^2]}$
= 13.6

Orientation (Φ)
= atan(Gy / Gx)
= 36 deg.

SKILLCATE

HOG: More illustration example

HOG Intro → Image Preprocessing → Calculate Gradient & Orientation → Build histogram → Normalisation

Total Gradient Magnitude = 13.6

Orientation (Φ) = 36 deg.

(40-36)/20

(36 - 20)/20

Magnitude		(4/20)*13.6	(16/20)*13.6						
Bin	0	20	40	60	80	100	120	140	160
Features	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

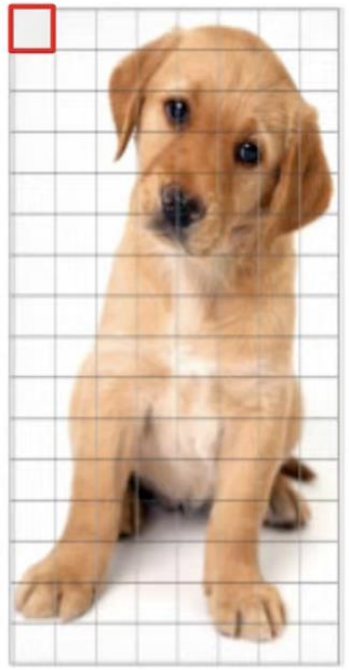
Weight

16 4

20 36 40

P32 | C8

Matrix 1 x 9



P64

C16

SKILLCATE

Summary: SIFT vs. HOG

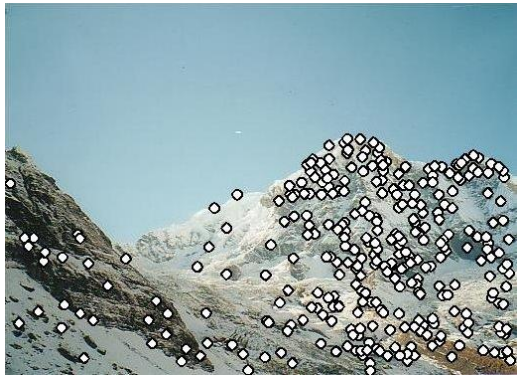
Both **SIFT** and **HOG** are Orientation based descriptors that are very powerful because they are robust to changes in brightness.

SIFT	HOG
SIFT computes the gradient histogram only for patches (usually 16*16 divided into 16 cells) **around specific interest points** obtained by taking the DoG's in the scale space. SIFT is specifically a 128d FV describes a 16×16 window patch.	HOG is computed for an entire image by dividing the image into smaller cells and summing up the gradients over every pixel within each cell in an image.
1.Based on first order gradients 2.It is evaluated around scale invariant feature points obtained using the difference of gaussian (DoG) key point detector. 3.Hand engineered and thus does not learn the representation by itself. 4.SIFT features are usually compared by computing the Euclidean distance between them.	1.Based on first order image gradients pooled in orientation bins. 2.It is evaluated all over the image. 3.Hand engineered, no learning algorithms for HOG features. 4.HOG is used to classify patches using classifiers such as SVM's.

Local Feature Matching

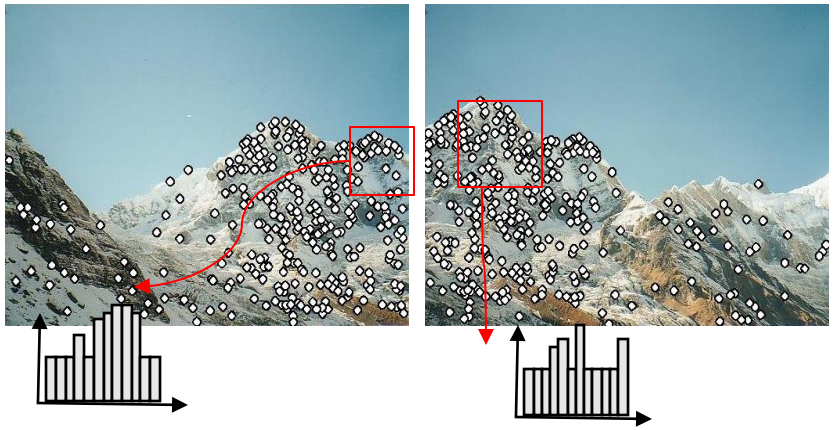
1) Detection: ✓

Identify the interest points (Find a set of distinctive features).



2) Description: ✓

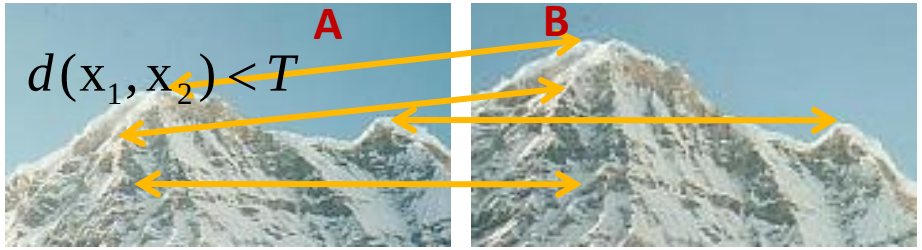
Extract vector feature descriptor surrounding each interest point.



3) Matching:

Determine correspondence between descriptors in two views by Computing **distance** between feature vectors

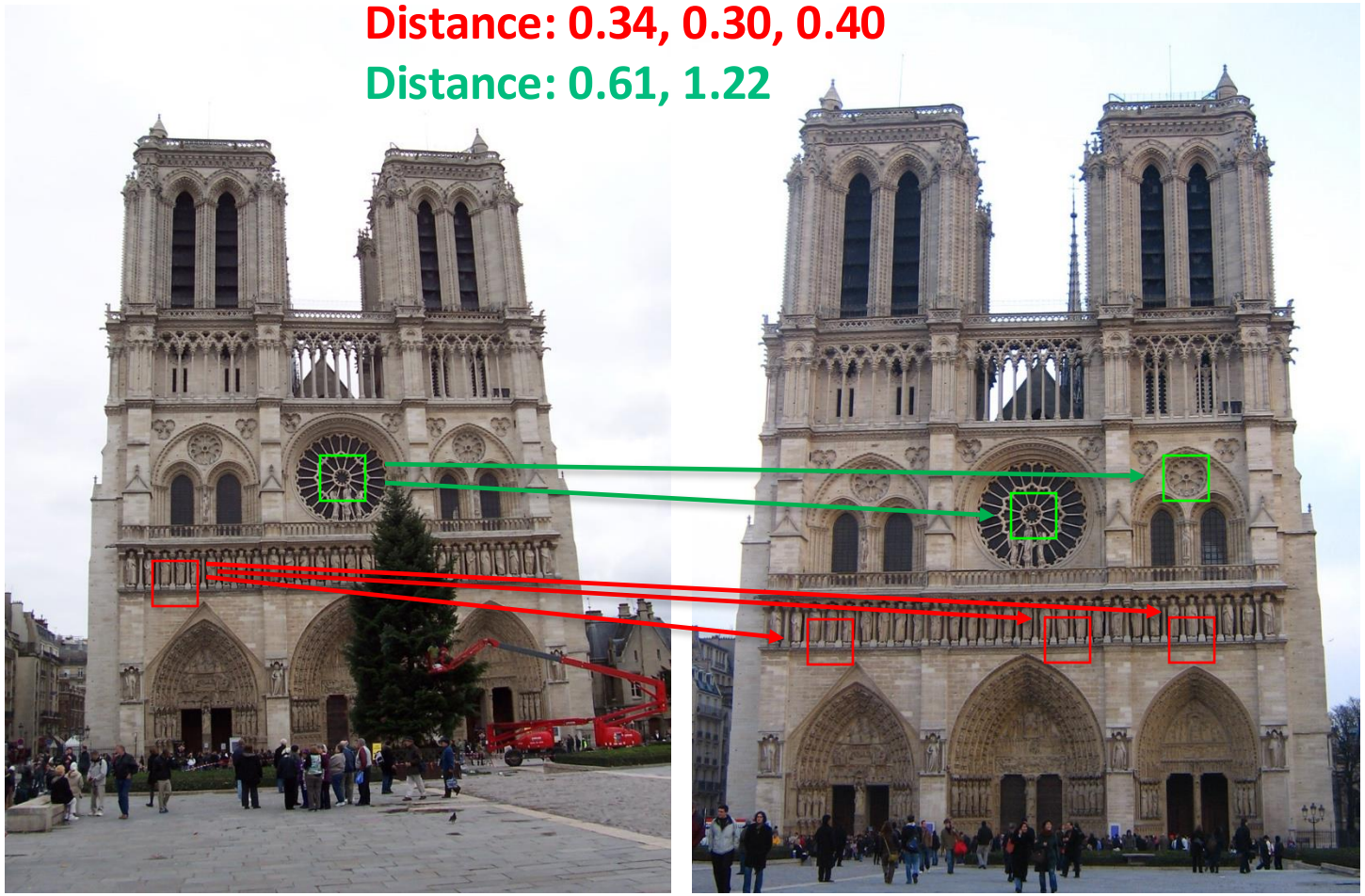
*(For each feature in **A**, find nearest neighbor in **B**.)*



Kristen Grauman

Local Feature Matching

Intuitively, we perform matching by first finding key points and calculating their feature vectors/descriptors in each image independently before computing the similarities between the feature vectors. We have a match between key points when their feature vectors are similar enough (more on this later).



Given a feature in I_1, how to find the best match in I_2?

1. Define distance function that compares two descriptors
2. Test all the features in I_2, find the one with min distance

Euclidean Distance vs. Cosine Similarity

These are two different ways of measuring the similarity of two vectors (in this case, two SIFT feature descriptors). Cosine similarity, unlike Euclidian distance, only measures the angle between two vectors and is independent of the vectors' magnitudes.

1. Euclidean distance:

$$\mathrm{d}(\mathbf{p}, \mathbf{q}) = \mathrm{d}(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

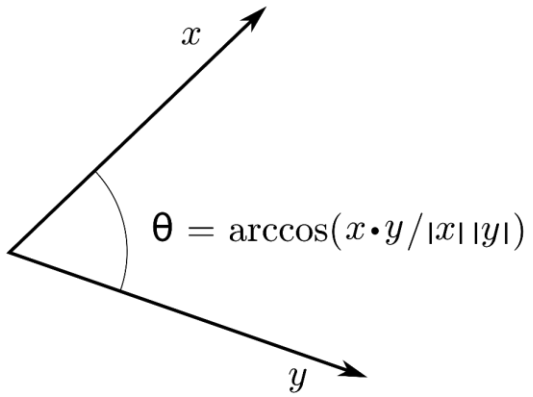
$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

$$\|\mathbf{q} - \mathbf{p}\| = \sqrt{(\mathbf{q} - \mathbf{p}) \cdot (\mathbf{q} - \mathbf{p})}.$$

2. Cosine similarity:

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\|_2 \|\mathbf{b}\|_2 \cos\theta$$

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|_2 \|\mathbf{B}\|_2}$$



Wikipedia

Feature Matching Criteria

- **Criteria 1:**
  - Compute distance in feature space, e.g., Euclidean distance between 128-dim SIFT descriptors.
  - Match point to lowest distance (nearest neighbor).

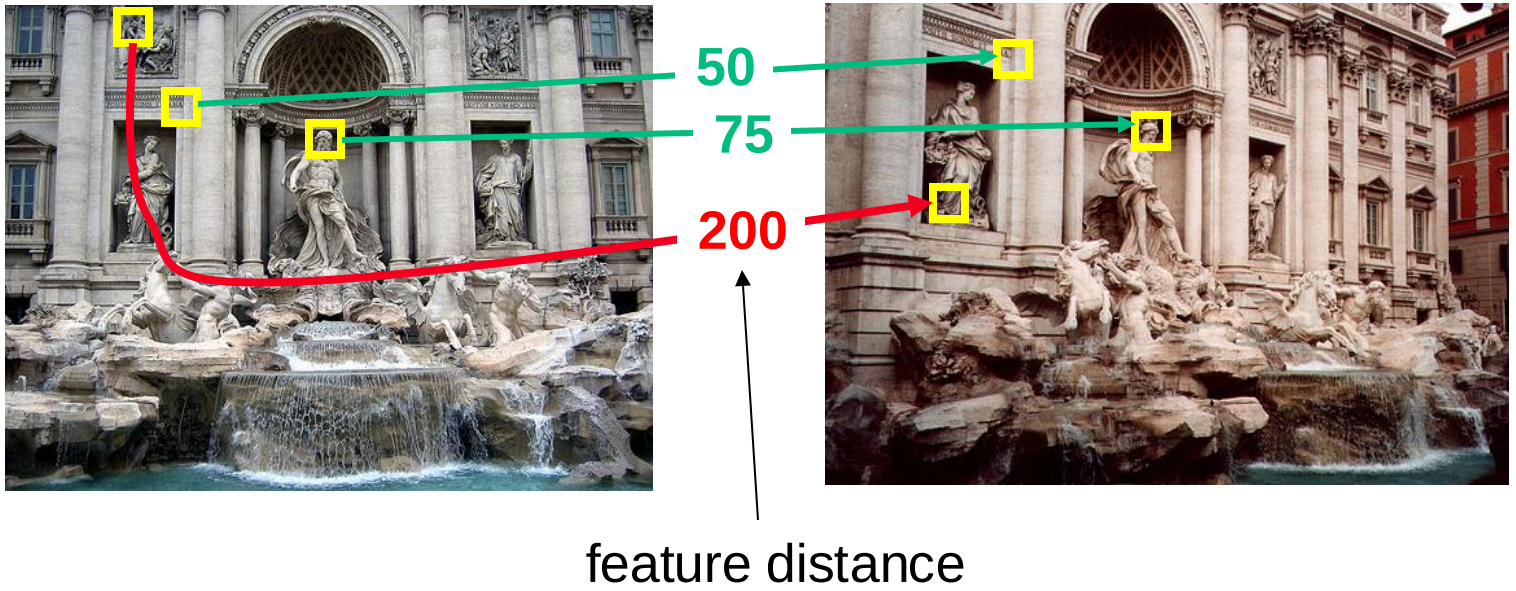
- **Problems:**
  - Does everything have a match?

- **Criteria 2:**
  - Compute distance in feature space, e.g., Euclidean distance between 128-dim SIFT descriptors.
  - Match point to lowest distance (nearest neighbor).
  - Ignore anything higher than threshold (no match!).

- **Problems:**
  - Threshold is hard to pick
  - Non-distinctive features could have lots of close matches, only one of which is correct

Evaluating the results

How can we measure the performance of a feature matcher?



The distance threshold affects performance

- **True positives** = # of detected matches that are correct
  - Suppose we want to maximize these—how to choose threshold?
- **False positives** = # of detected matches that are incorrect
  - Suppose we want to minimize these—how to choose threshold?

References

- Rafael C. Gonzalez and Richard E. Wood, Digital Image Processing, 2nd Edition.
- **Corner Detection, SIFT Descriptor,** Shree K. Nayar, Monograph FPCV-2-1, First Principles of Computer Vision, Columbia University, New York, May. 2022
- *Introduction to computer vision,* **James Hays, Brown University**. Spring. 2023
- **Richard Szeliski,** Computer Vision: Algorithms and Applications
- https://www.slideshare.net/AnirudhKanneganti/hog-122762396

- Feature Extraction using SIFT and HOG (in python): https://www.neuralception.com/featureextractors-2/