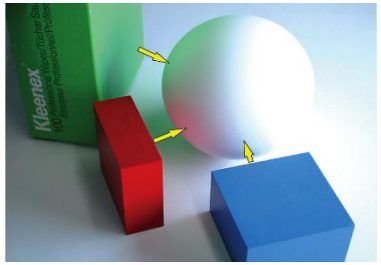
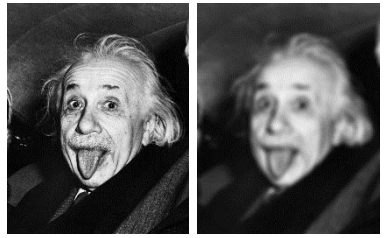
Interest Points

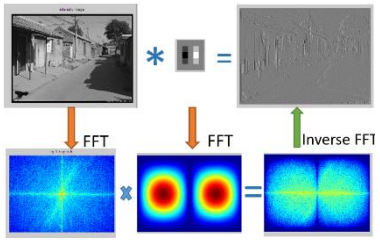
Dr. Sander Ali Khowaja,
Assistant Professor, Department of Telecommunication Engineering
Faculty of Engineering and Technology, University of Sindh, Pakistan
Senior Member, IEEE – Member, ACM
https://sander-ali.github.io

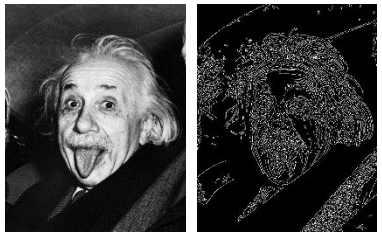
What we have learned so far



- Light and color
  - What an image records
- Filtering in spatial domain
  - Filtering = weighted sum of neighboring pixels
  - Smoothing, sharpening, measuring texture
- Filtering in frequency domain
  - Filtering = change frequency of the input image
  - Denoising, sampling, image compression
- Image pyramid and template matching
  - Filtering = a way to find a template
  - Image pyramids for coarse-to-fine search and multi-scale detection
- Edge detection
  - Canny edge = smooth -> derivative -> thin -> threshold -> link
  - Finding straight lines, binary image analysis

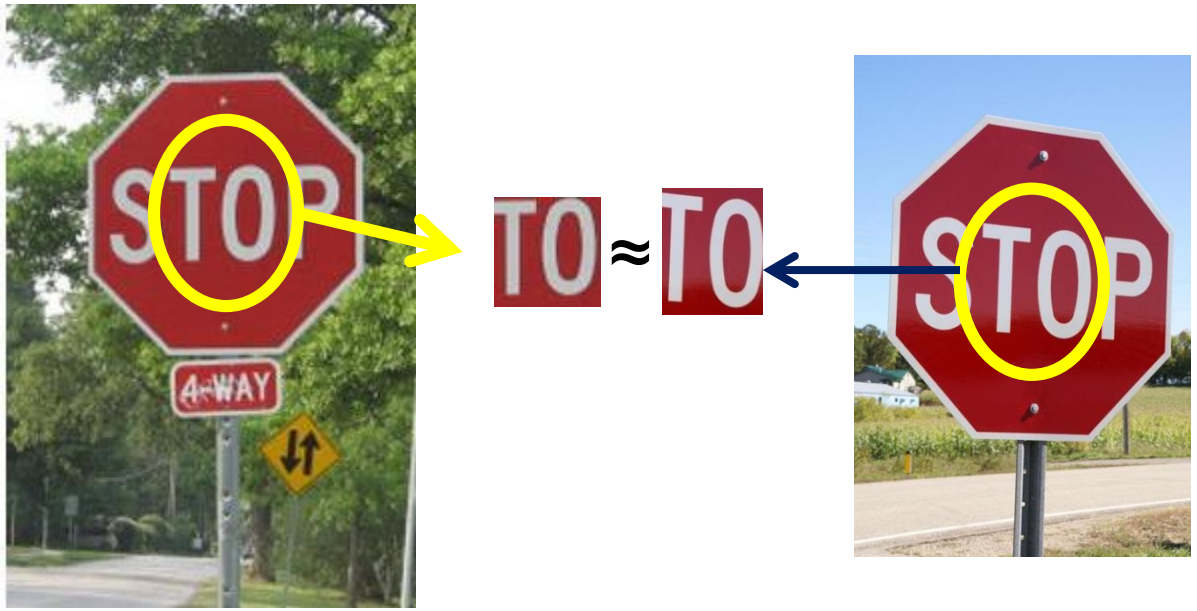






This Module: Correspondence and Alignment

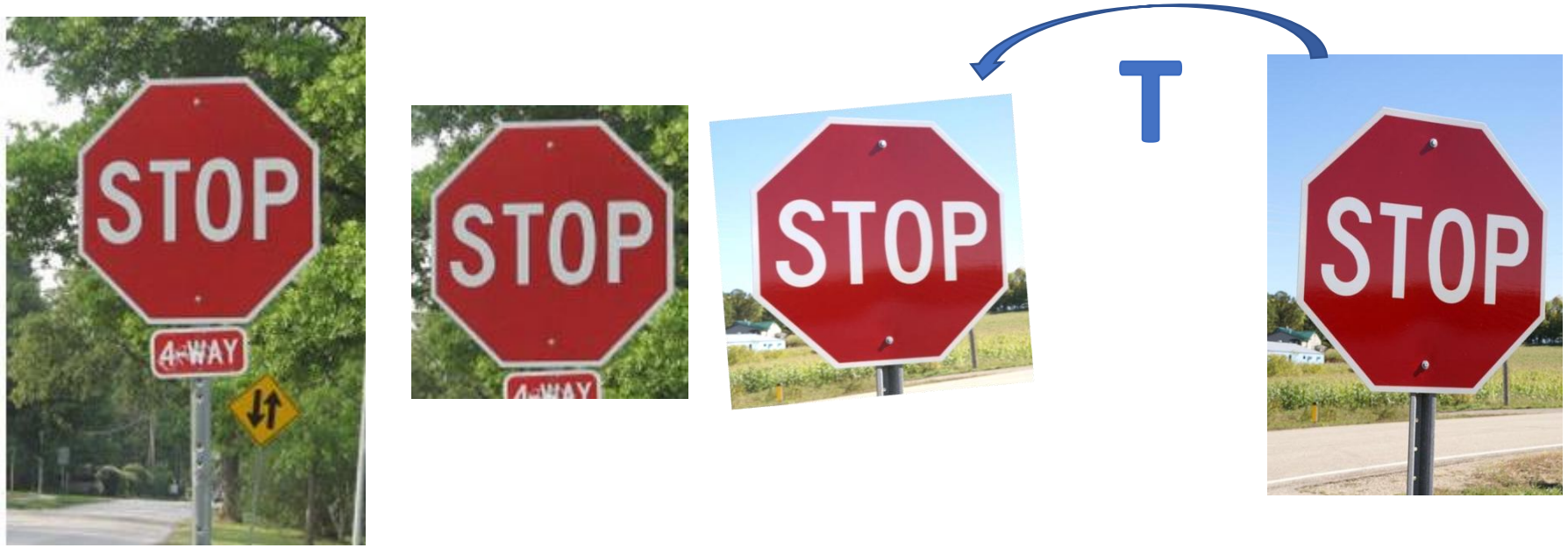
- **Correspondence**:
  matching points, patches, edges, or regions across images



Slide credit: Derek Hoiem

This module: Correspondence and Alignment

- **Alignment/registration**:
  solving the transformation that makes two things match better



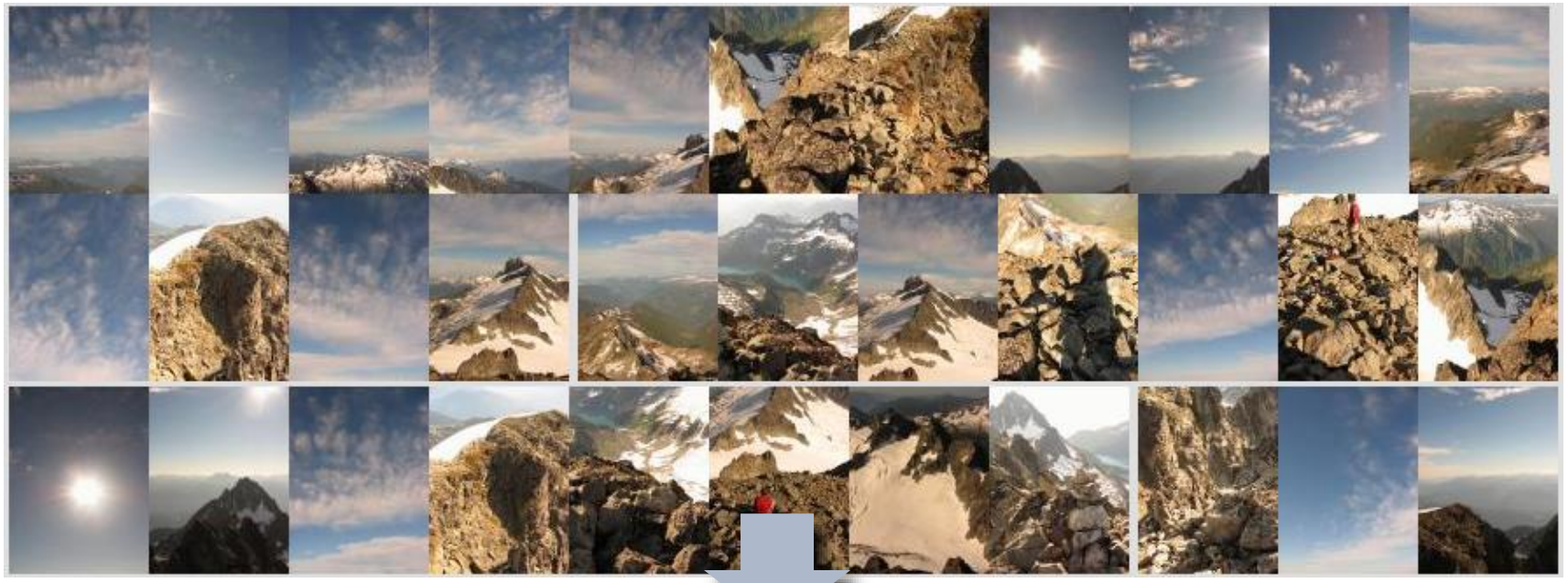
Slide credit: Derek Hoiem

The Three Biggest Problems in Computer Vision

1. **Registration**
2. **Registration**
3. **Registration**

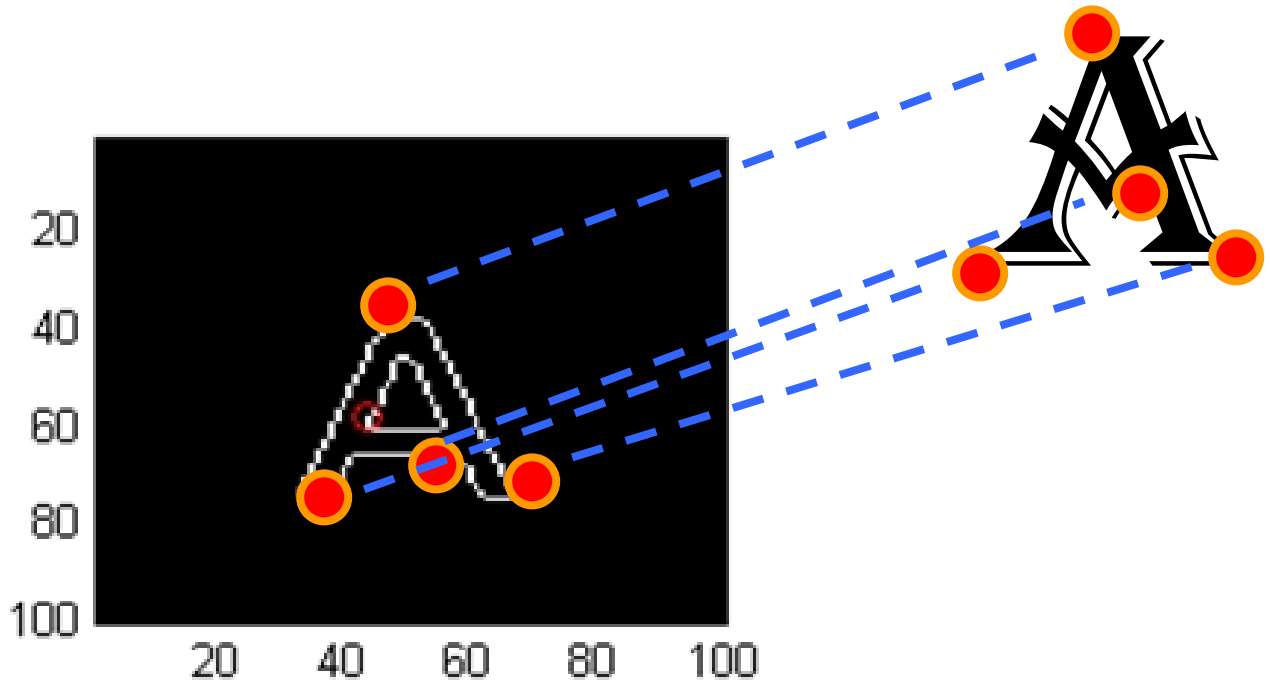
Takeo Kanade (CMU)

Automatic Panoramas



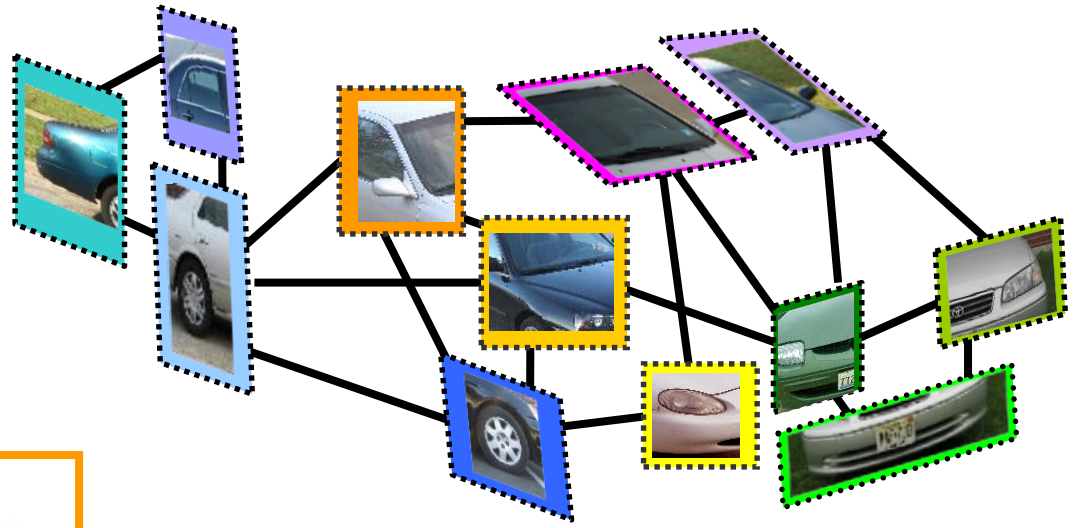
Credit: Matt Brown

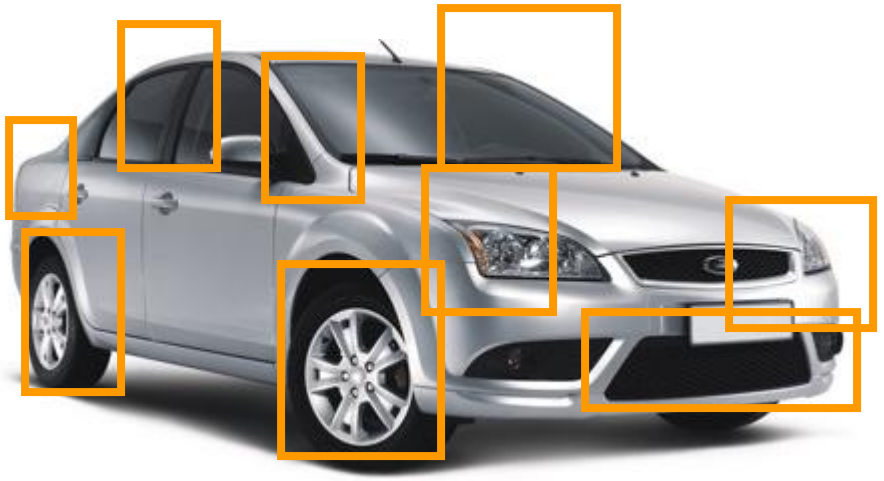
Fitting a 2D shape template



Slide from Silvio Savarese

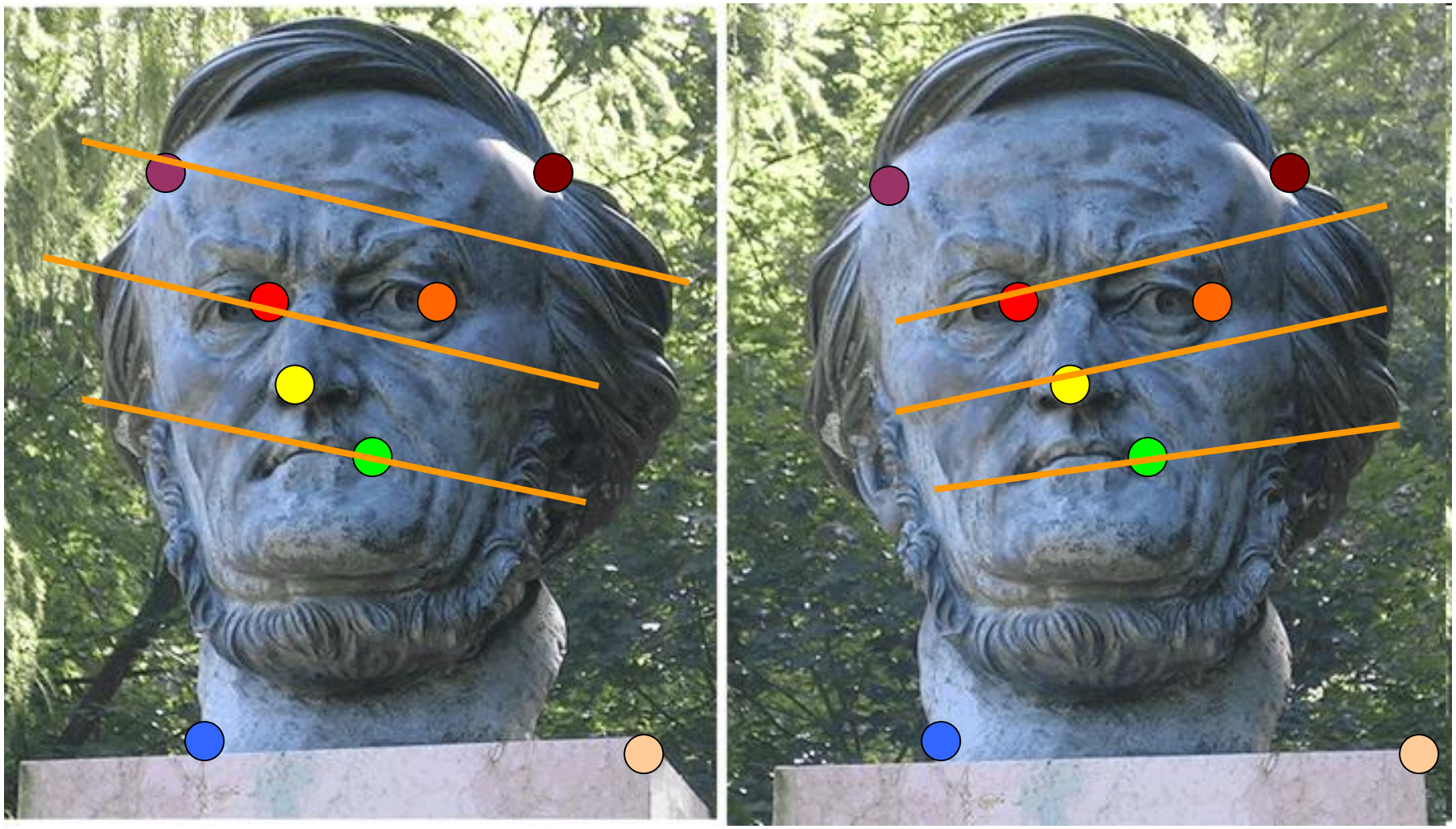
Fitting a 3D object model





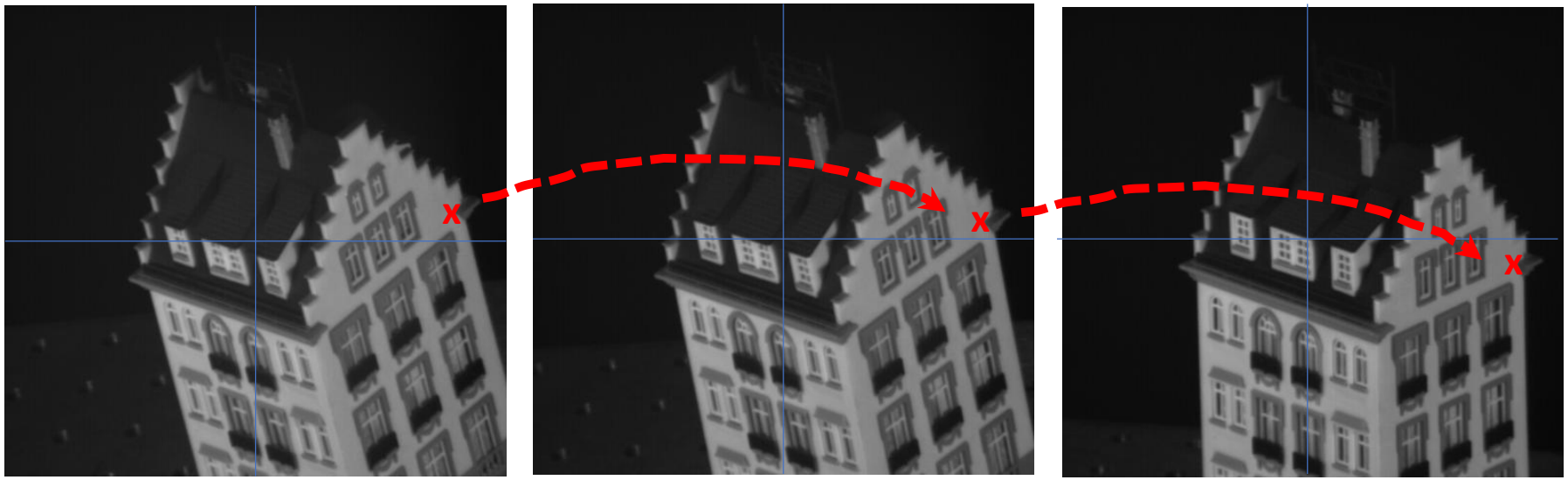
Slide from Silvio Savarese

Estimating Fundamental Matrix that Correspond two Views



Slide from Silvio Savarese

Tracking Points



frame 0 frame 22 frame 49

This Lecture

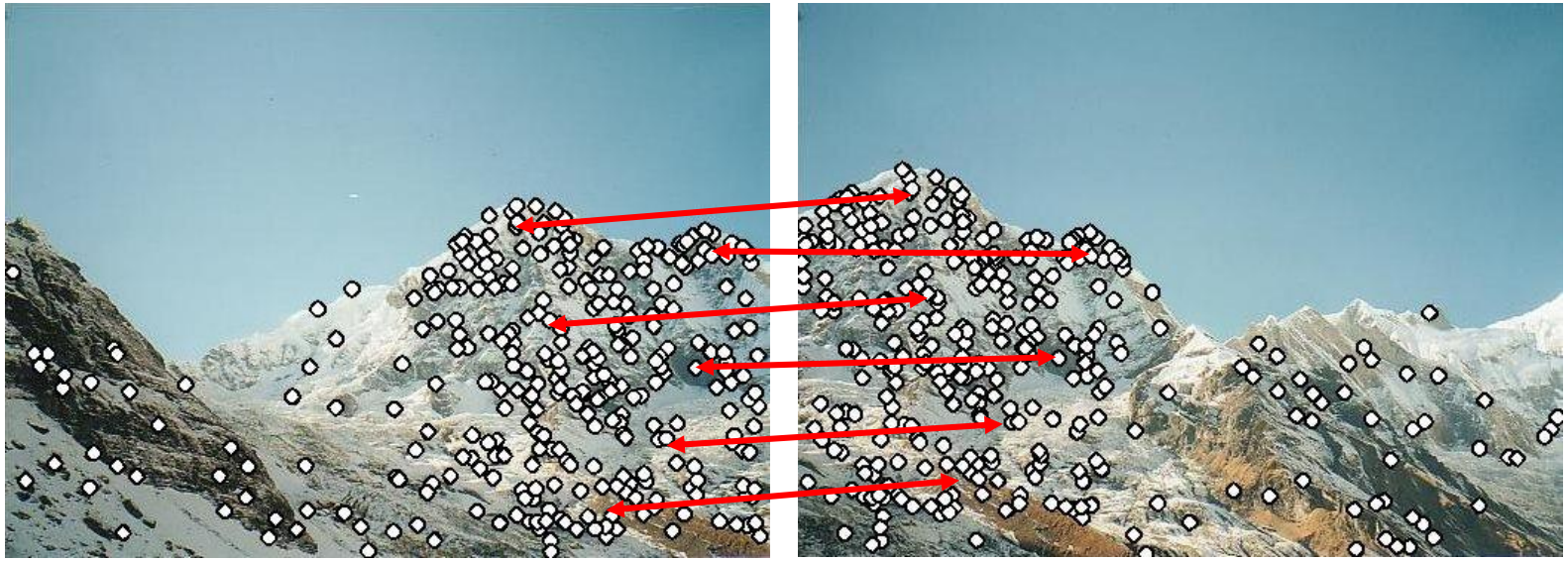
- What is interest point?
- Corner detection
- Handling scale and orientation
- Feature matching

Why Extract Features

- Motivation: panorama stitching
  - We have two images – how do we combine them?

Why Extract Features

- Motivation: panorama stitching
  - We have two images – how do we combine them?



Step 1: extract features
Step 2: match features

Why Extract Features

- Motivation: panorama stitching
  - We have two images – how do we combine them?

Step 1: extract features
Step 2: match features
Step 3: align images

Image Matching

by Diva Sian

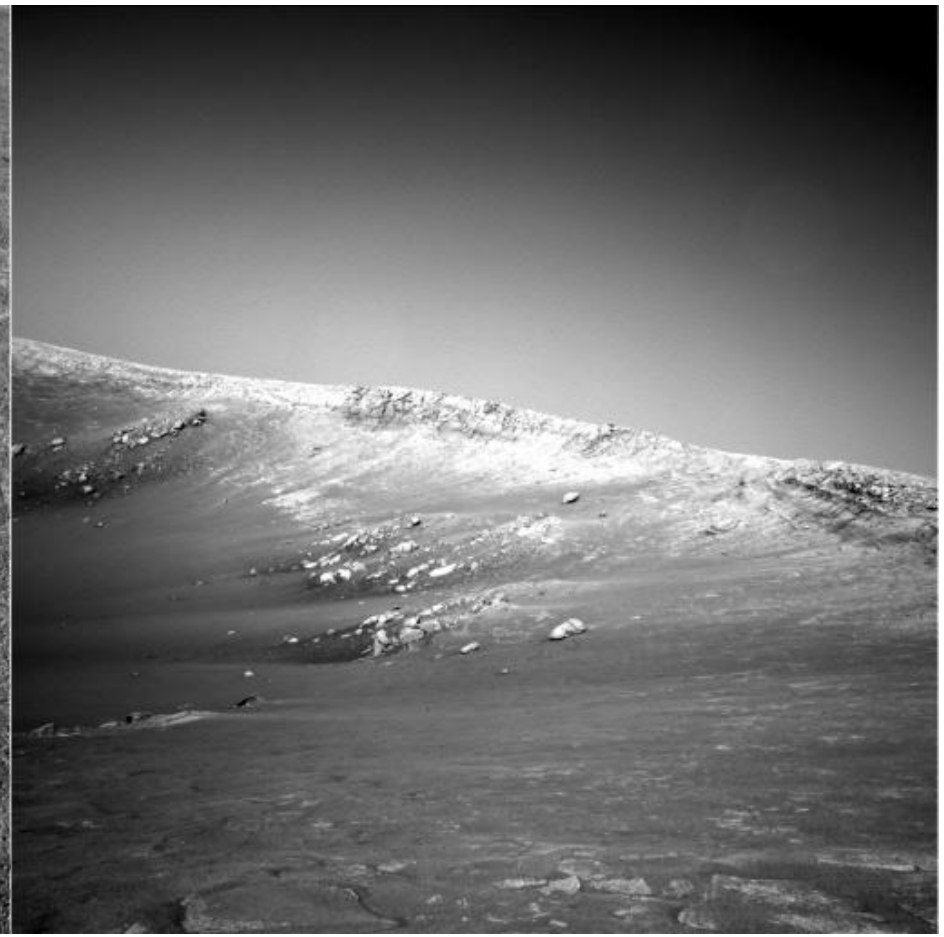
by swashford

Image Matching: Harder Case

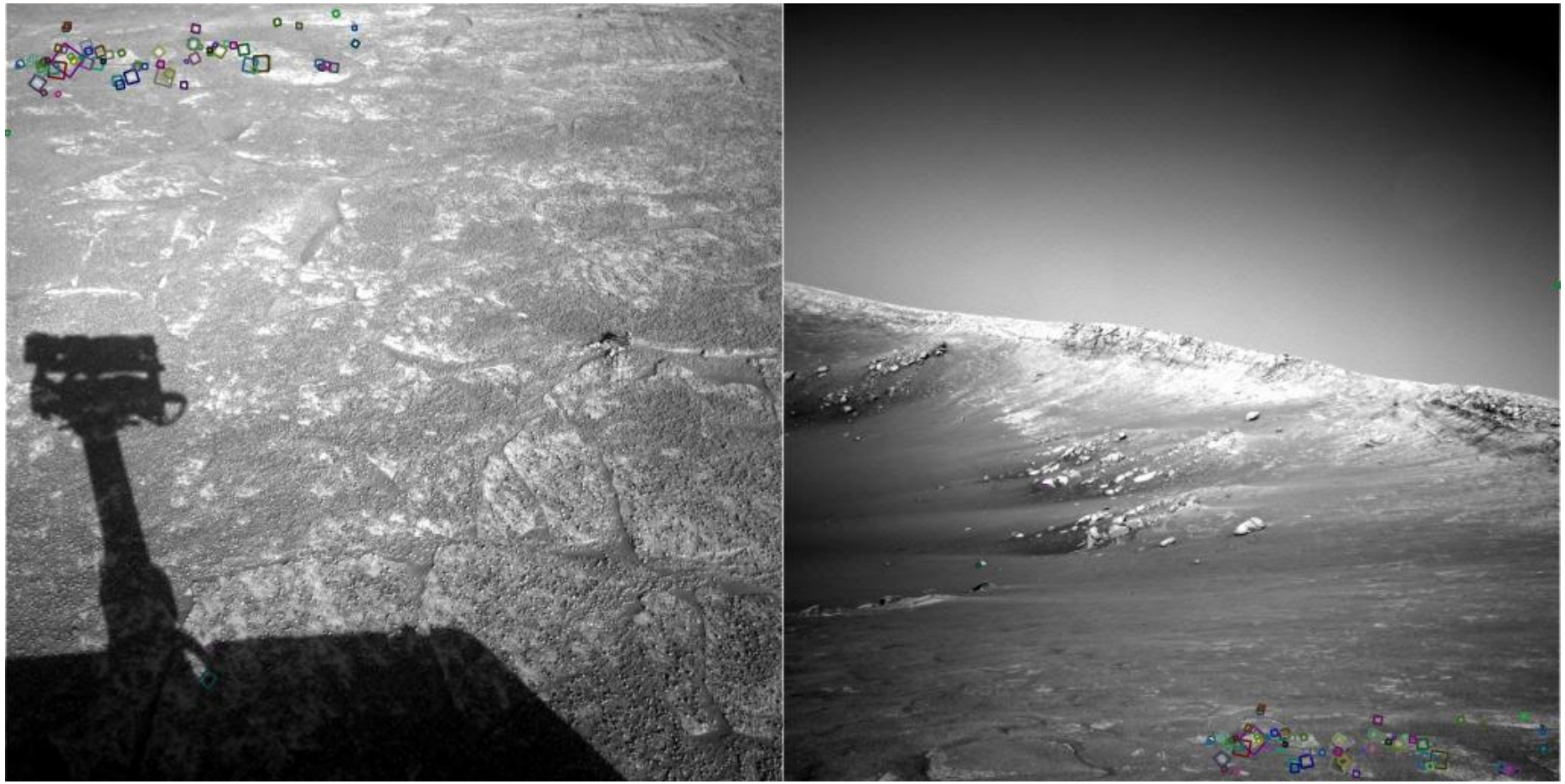
by Diva Sian

by scgbt

What do you think about this



Answer: Look for Tiny Squares



NASA Mars Rover images
with SIFT feature matches

Applications

- Keypoints are used for:
  - Image alignment
  - 3D reconstruction
  - Motion tracking
  - Robot navigation
  - Indexing and database retrieval
  - Object recognition

Advantages of Local Features

Locality

- features are local, so robust to occlusion and clutter

Quantity

- hundreds or thousands in a single image

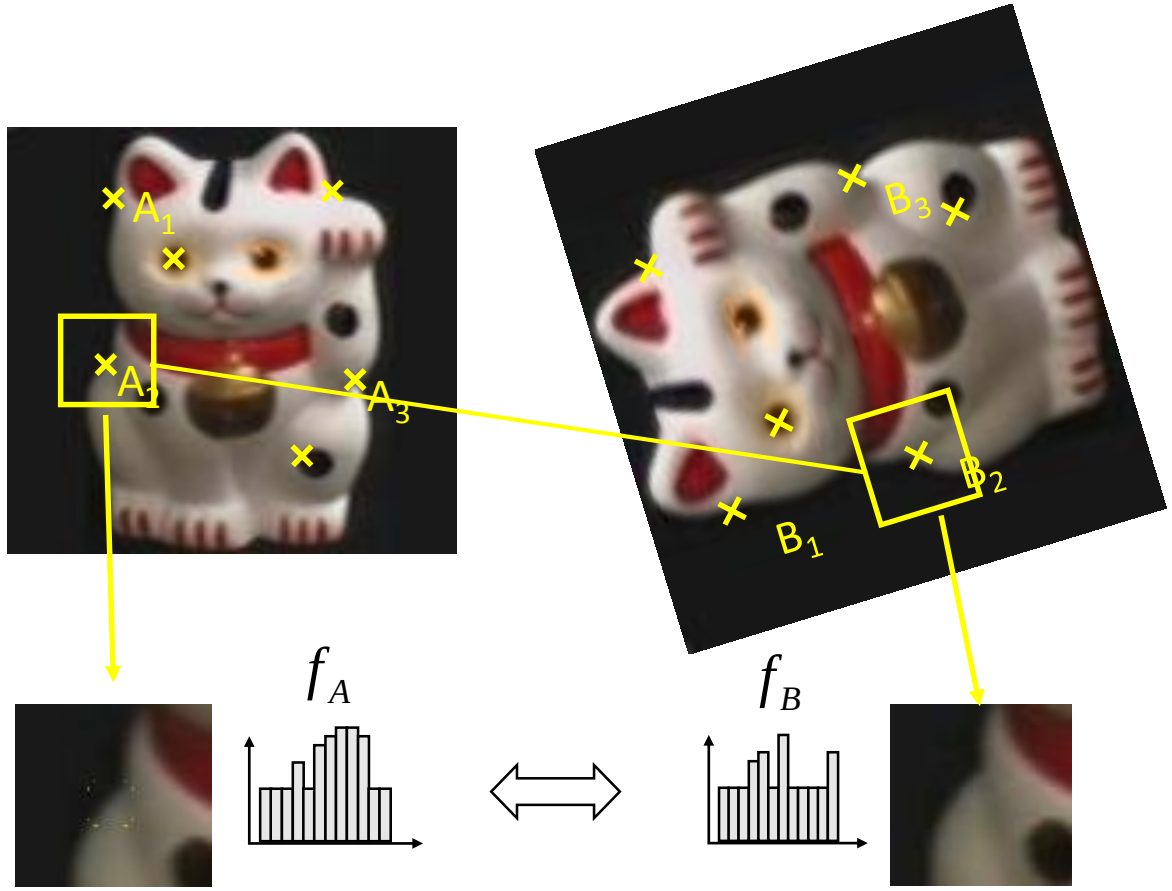
Distinctiveness:

- can differentiate a large database of objects

Efficiency

- real-time performance achievable

Overview of Keypoint Matching



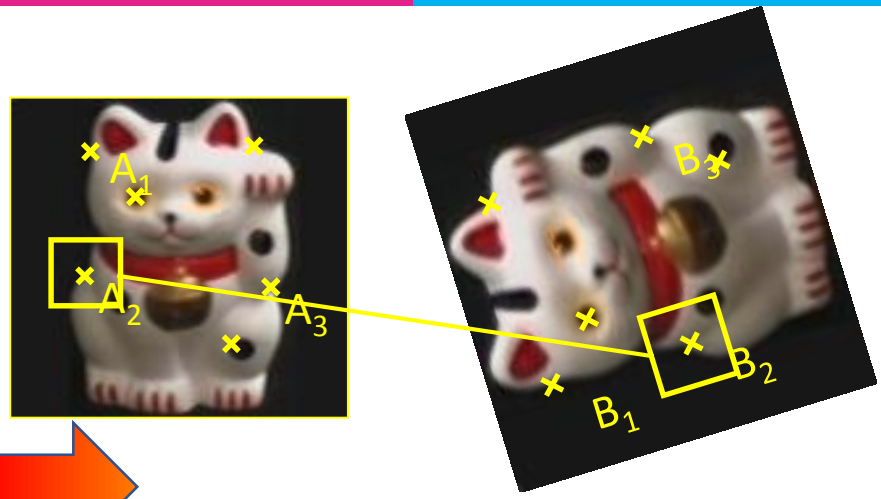
1. Find a set of distinctive key-points
2. Define a region around each keypoint
3. Extract and normalize the region content
4. Compute a local descriptor from the normalized region
5. Match local descriptors

K. Grauman, B. Leibe

Goals for Keypoints

Detect points that are *repeatable* and *distinctive*

Key Trade-offs



Detection

More Repeatable

Robust detection
Precise localization

More Points

Robust to occlusion
Works with less texture

Description

More Distinctive

Minimize wrong matches

More Flexible

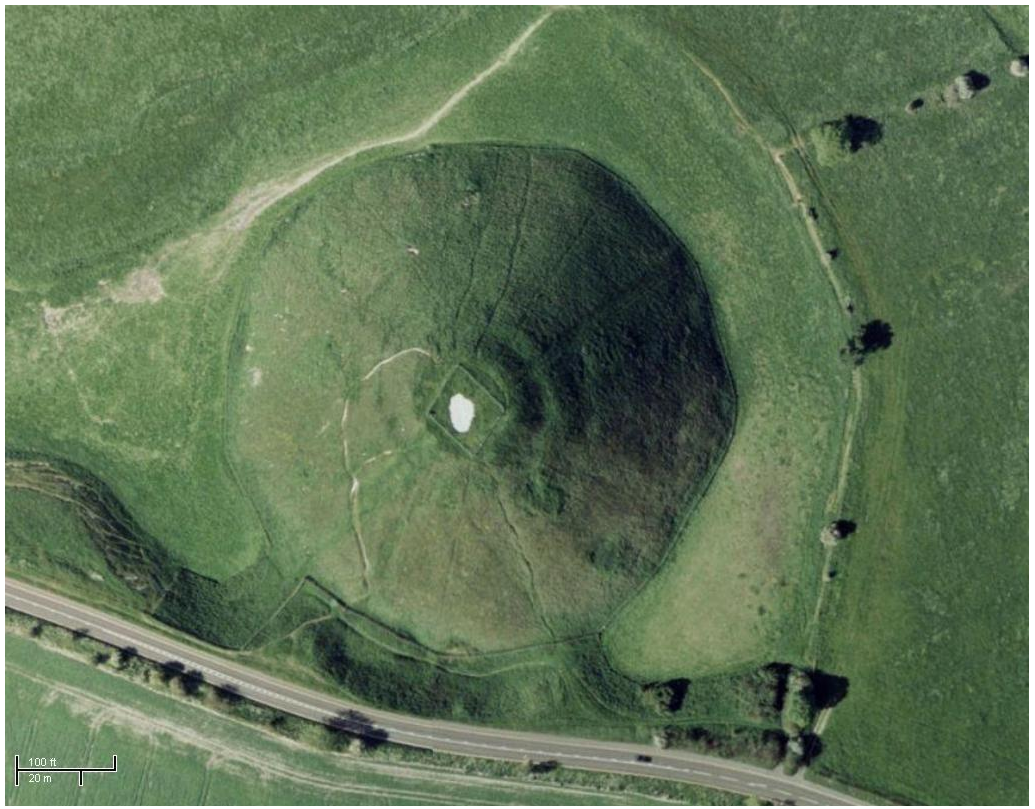
Robust to expected variations
Maximize correct matches

Choosing Interest Points

Where would you tell your friend to meet you?

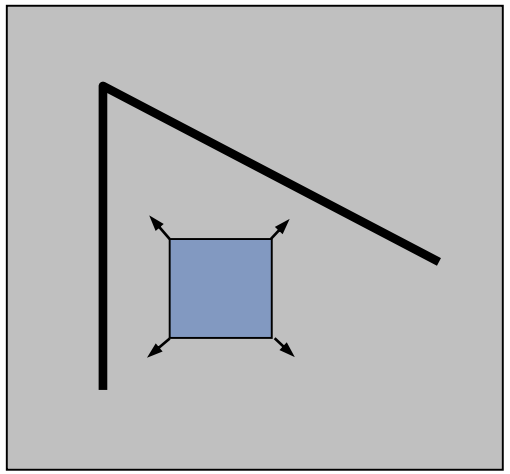
Choosing Interest Points

Where would you tell your friend to meet you?

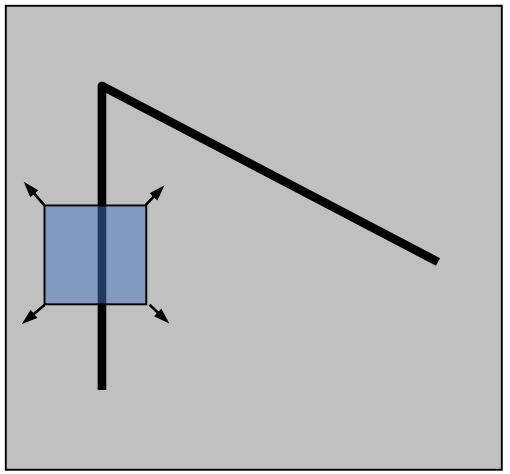


Corner Detection: Basic Idea

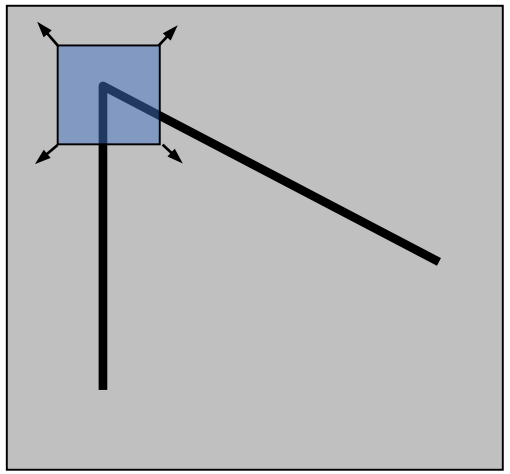
- How does the window change when you shift it?
- Shifting the window in any direction causes a big change



"flat" region:
no change in all directions



"edge":
no change along the edge direction

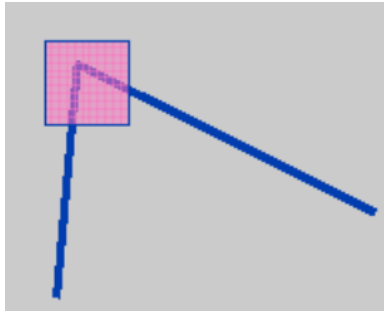


"corner":
significant change in all directions

Credit: S. Seitz, D. Frolova, D. Simakov

Corners

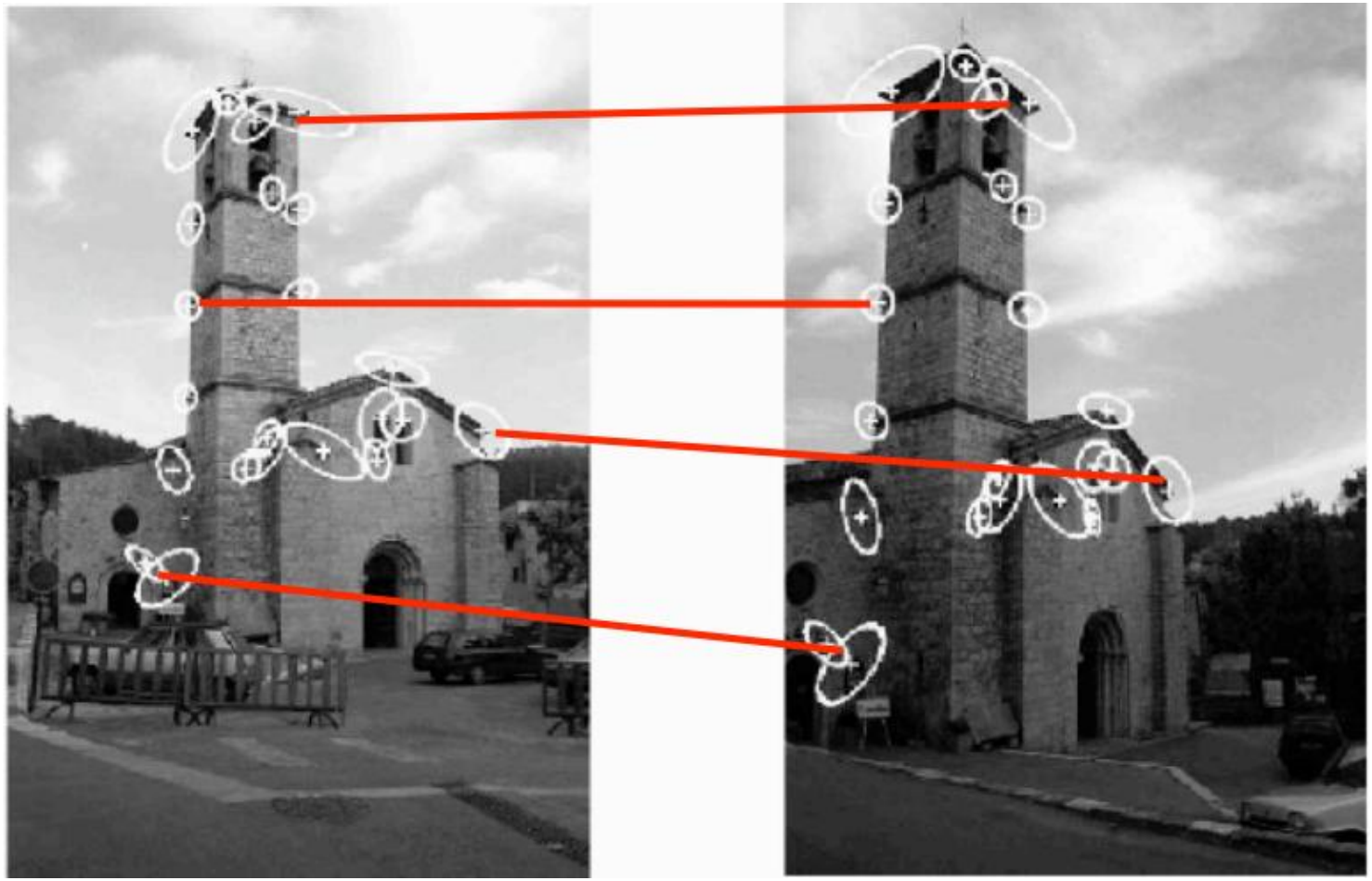
- Point whose local neighborhood stands in two dominant and different edge directions.
- Junction of two edges.



- Corners are mostly termed as interest points as they are invariant to translation, rotation, and illumination.
- Corners are used as a basic building block for various applications such as Motion tracking, image stitching, 2D mosaics, stereo vision, image representation, and so forth.

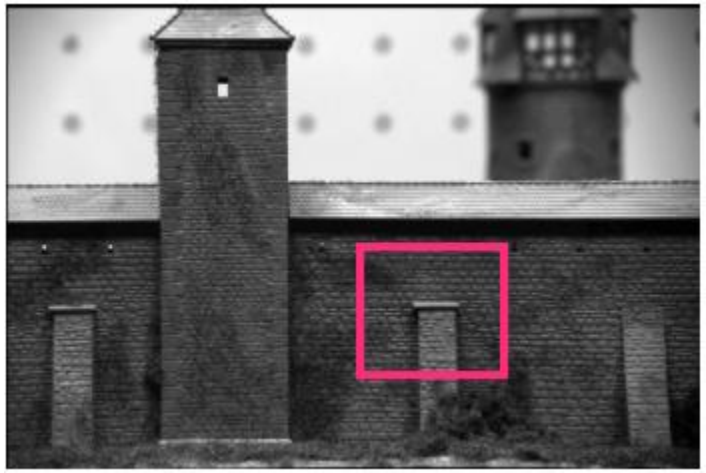
Motivation – Matching Problem

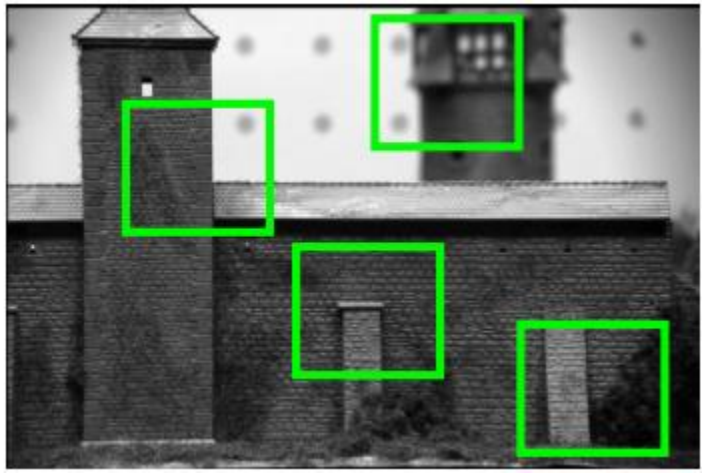
- Vision tasks such as stereo and motion estimation require finding corresponding features across two or more views.



Motivation – Patch Matching

- Elements to be matched are image patches of fixed size

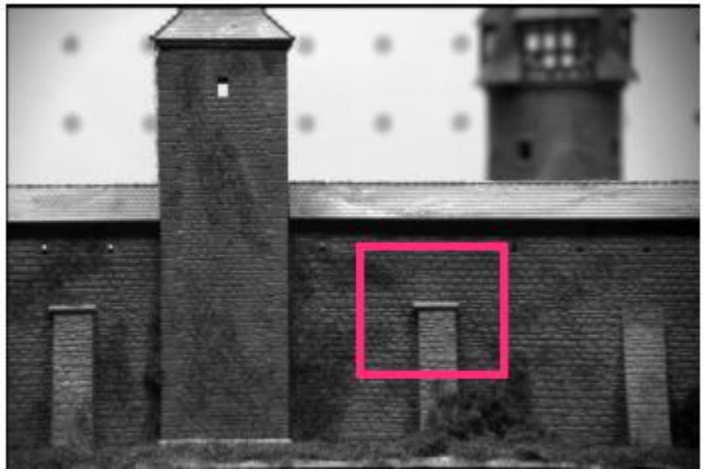


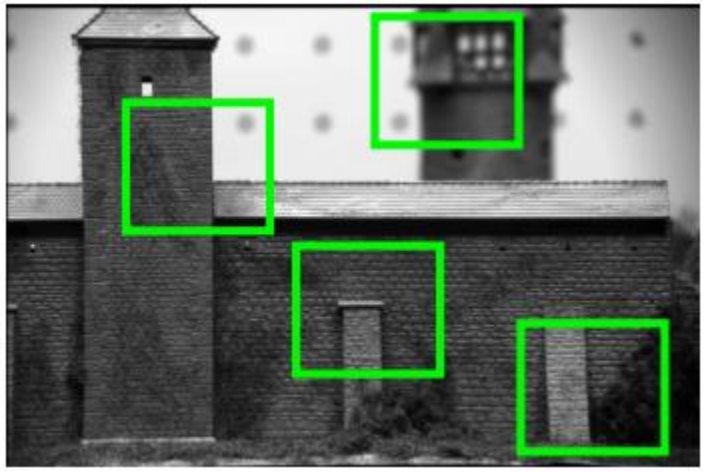


- Task: find the best (most similar) patch in the second image.

Not all patches are created equal

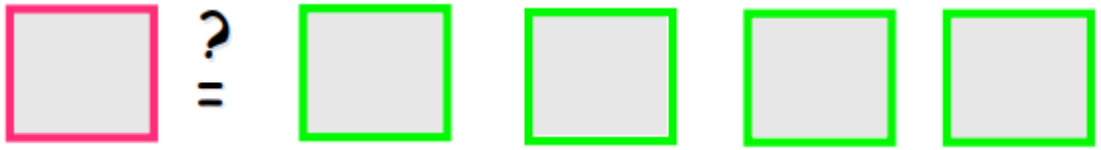
- Intuition: This would be a good patch for matching, since it is very distinctive (there is only one patch in the second frame that looks similar).





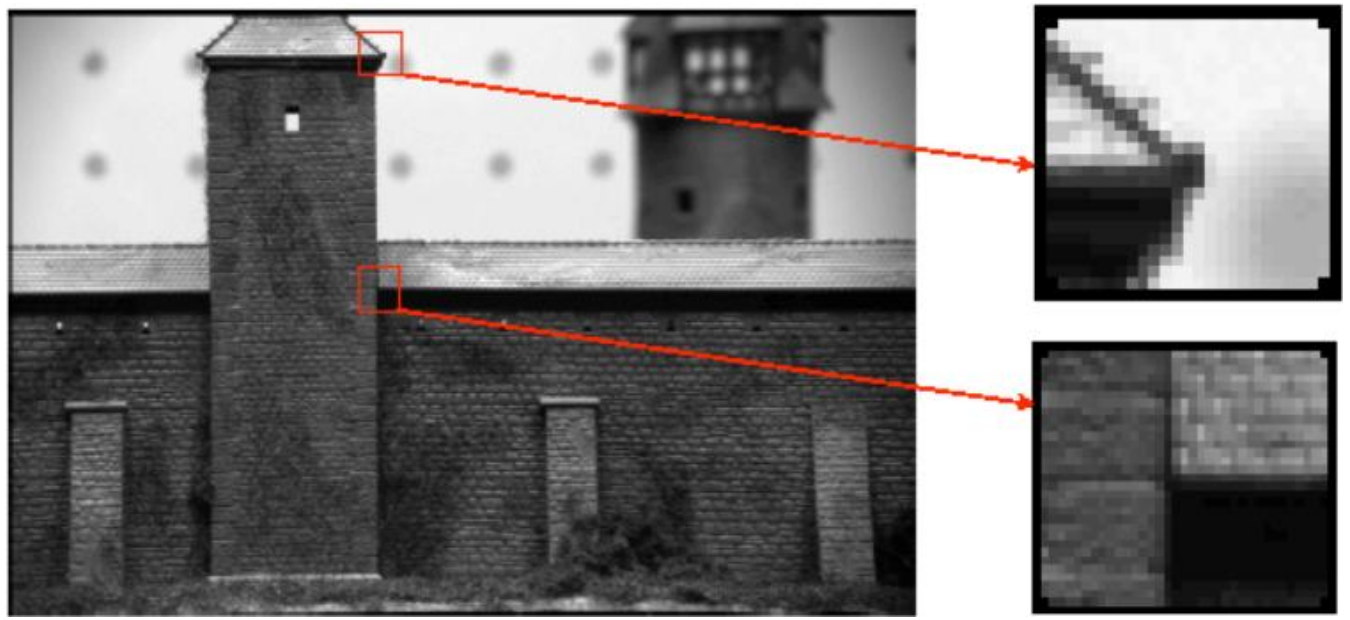
Not all patches are created equal

- Intuition: This would be a Bad patch for matching, since it is not very distinctive (there are many similar patches in the second frame)



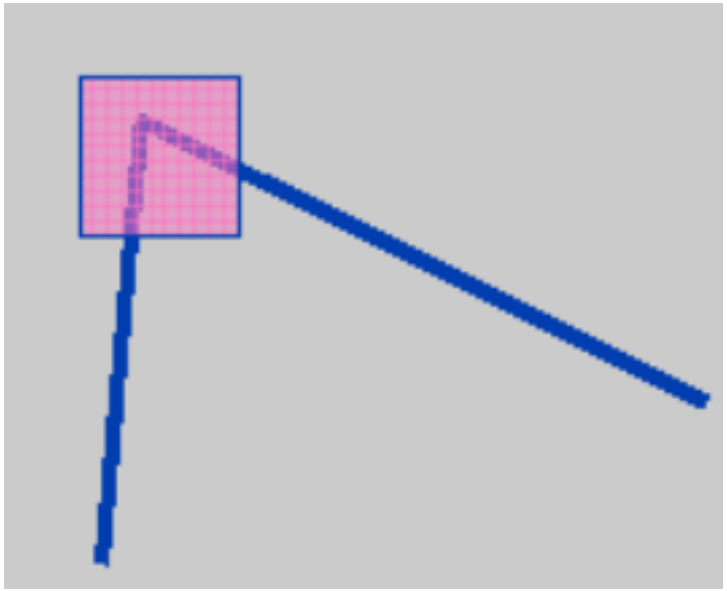
Why Corners?

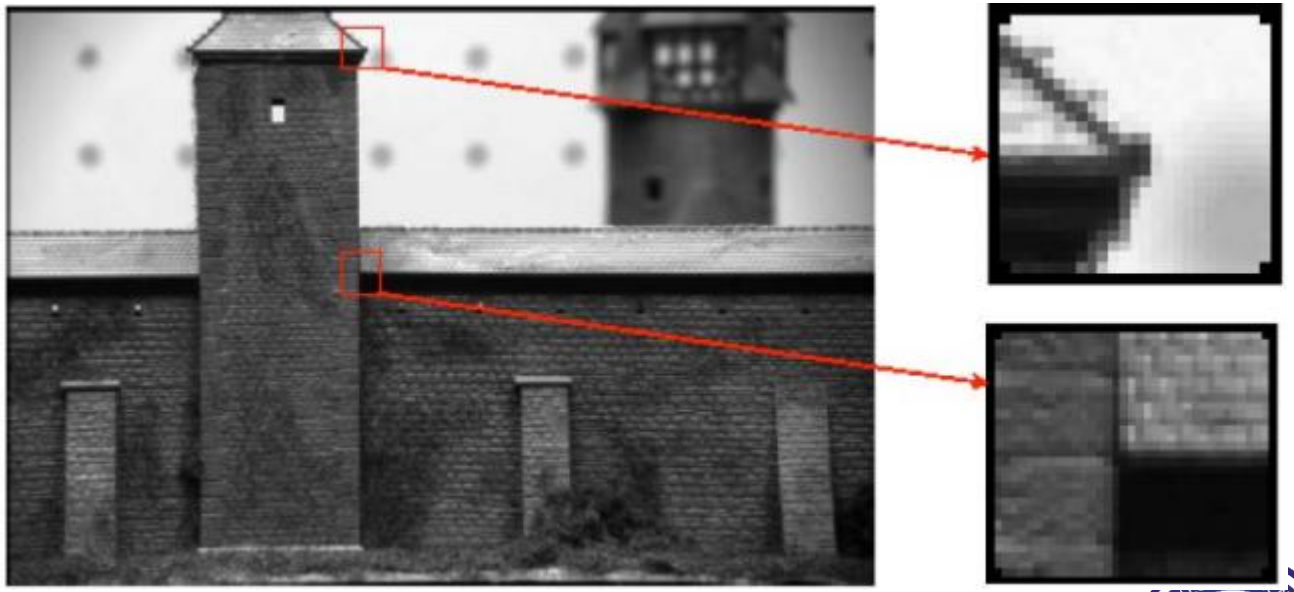
- Intuitively, junctions of contours
- Generally more stable features over changes of viewpoint.
- Intuitively, large variations in the neighborhood of the point in all directions.
- Simply, there are good features to match.



Corner Basic Idea

- We should easily recognize the point by looking at intensity values within a small window.
- Shifting the window in any direction should yield a large change in appearance.





33

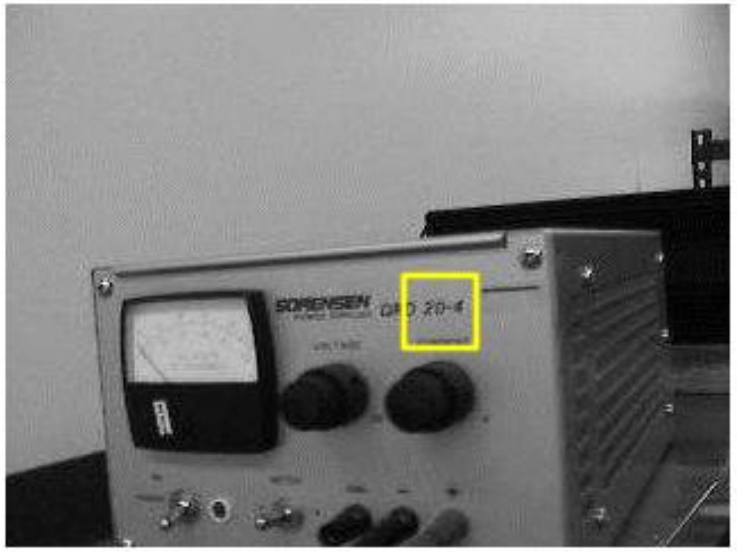
Sum of Squared Differences

- Say we want to track a patch from image I0 to I1
  - We'll assume that intensities don't change, and only translational motion
  - So we expect that

$$I_1(\mathbf{x}_i + \mathbf{u}) = I_0(\mathbf{x}_i)$$

  - In practice we will have noise, so they won't be exactly equal
- But we can search for the displacement u = (x,y) that minimizes the sum of squared differences

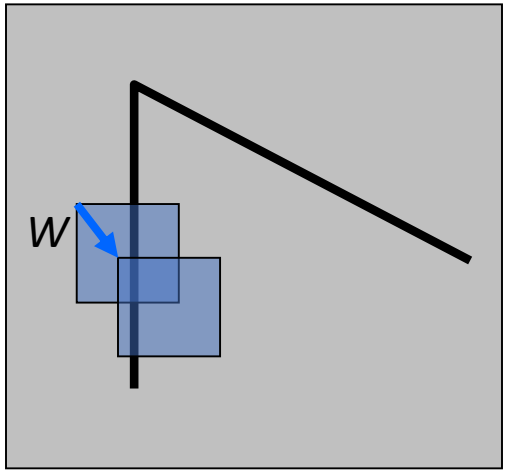
$$E(\mathbf{u}) = \sum_i w(\mathbf{x}_i)\left[I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)\right]^2$$



Harris Corner Detection

Consider shifting the window *W* by (*u,v*)

- how do the pixels in W change?
- compare each pixel before and after by summing up the squared differences (SSD)
- this defines an SSD "error" *E*(*u,v*):



$$E(u,v) = \sum_{(x,y)\in W} (I(x+u,y+v) - I(x,y))^2$$

Harris Corner Detection Math: Small Motion Assumption

Taylor Series expansion of *I*:

$$I(x+u, y+v) = I(x,y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \text{higher order terms}$$

If the motion (u,v) is small, then first order approximation is good

$$I(x+u, y+v) \approx I(x,y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v$$

$$\approx I(x,y) + \begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

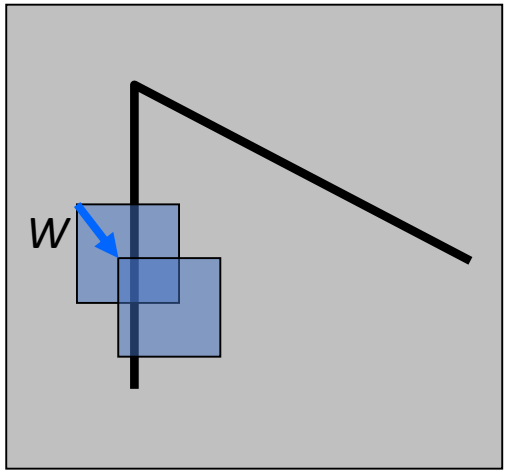
$$\text{shorthand: } I_x = \frac{\partial I}{\partial x}$$

Plugging this into the formula on the previous slide…

Harris Corner Detection: The Math

Using the small motion assumption,
replace I with a linear approximation

(Shorthand: $I_x = \frac{\partial I}{\partial x}$)



$$E(u,v) = \sum_{(x,y)\in W} (I(x+u,y+v) - I(x,y))^2$$

$$\approx \sum_{(x,y)\in W} (I(x,y) + I_x(x,y)u + I_y(x,y)v - I(x,y))^2$$

$$\approx \sum_{(x,y)\in W} (I_x(x,y)u + I_y(x,y)v)^2$$

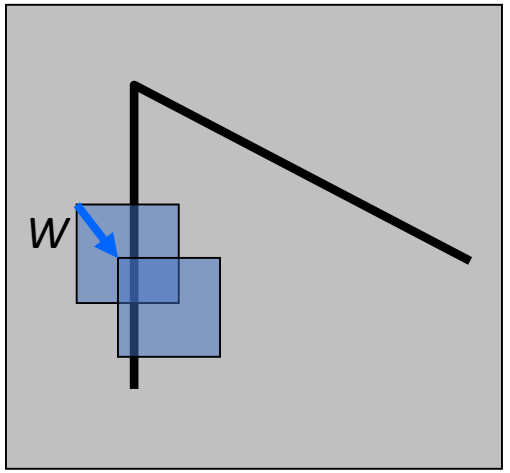
Corner Detection: The Math

$$E(u,v) \approx \sum_{(x,y)\in W} \left(I_x(x,y)u + I_y(x,y)v\right)^2$$

$$\approx \sum_{(x,y)\in W} \left(I_x^2u^2 + 2I_xI_yuv + I_y^2v^2\right)$$

$$\approx Au^2 + 2Buv + Cv^2$$

$$A = \sum_{(x,y)\in W} I_x^2 \qquad B = \sum_{(x,y)\in W} I_xI_y \qquad C = \sum_{(x,y)\in W} I_y^2$$



- Thus, *E*(*u*,*v*) is locally approximated as a *quadratic form*

Second Order Moment

The surface $E(u,v)$ is locally approximated by a quadratic form.

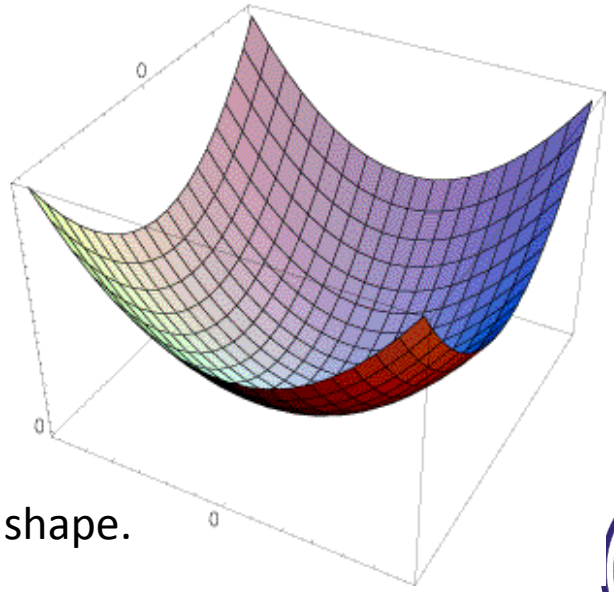
$$E(u,v) \approx Au^2 + 2Buv + Cv^2$$

$$\approx \begin{bmatrix} u & v \end{bmatrix} \underbrace{\begin{bmatrix} A & B \\ B & C \end{bmatrix}}_{H} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$A = \sum_{(x,y)\in W} I_x^2$$

$$B = \sum_{(x,y)\in W} I_x I_y$$

$$C = \sum_{(x,y)\in W} I_y^2$$



Let's try to understand its shape.

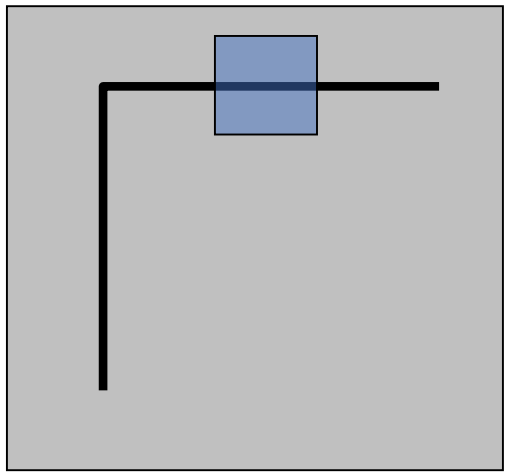
Corner Detection: The Math

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \underbrace{\begin{bmatrix} A & B \\ B & C \end{bmatrix}}_{H} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$A = \sum_{(x,y)\in W} I_x^2$$

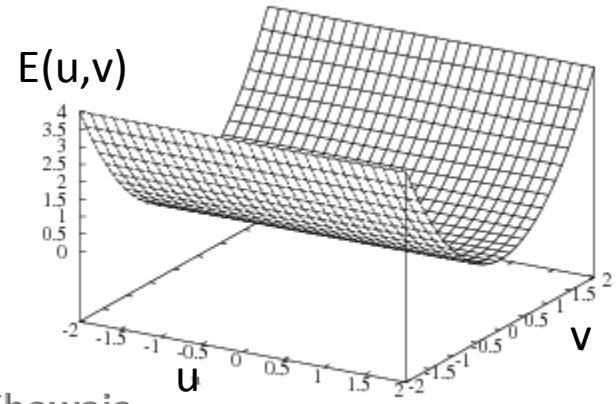
$$B = \sum_{(x,y)\in W} I_x I_y$$

$$C = \sum_{(x,y)\in W} I_y^2$$



Horizontal edge: $I_x = 0$

$$H = \begin{bmatrix} 0 & 0 \\ 0 & C \end{bmatrix}$$



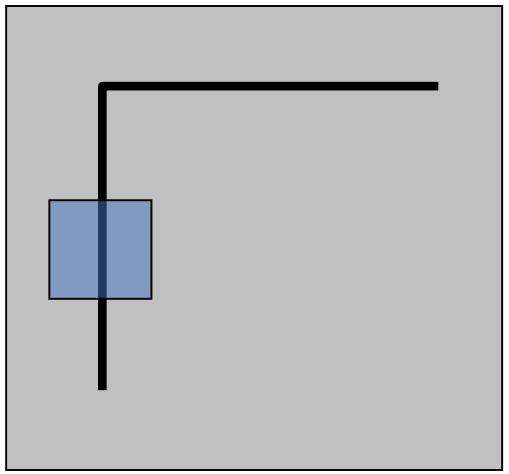
Corner Detection: The Math

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \underbrace{\begin{bmatrix} A & B \\ B & C \end{bmatrix}}_{H} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$A = \sum_{(x,y)\in W} I_x^2$$

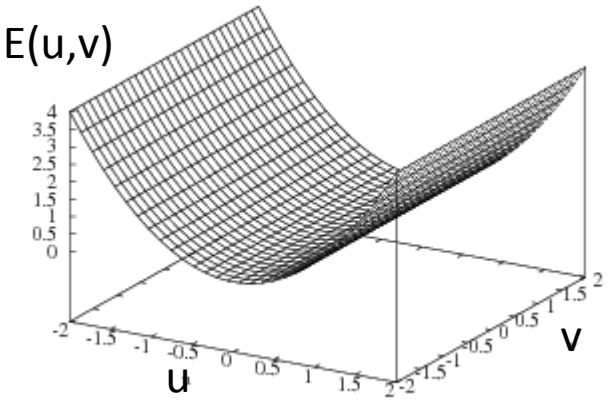
$$B = \sum_{(x,y)\in W} I_x I_y$$

$$C = \sum_{(x,y)\in W} I_y^2$$



Vertical edge: $I_y = 0$

$$H = \begin{bmatrix} A & 0 \\ 0 & 0 \end{bmatrix}$$



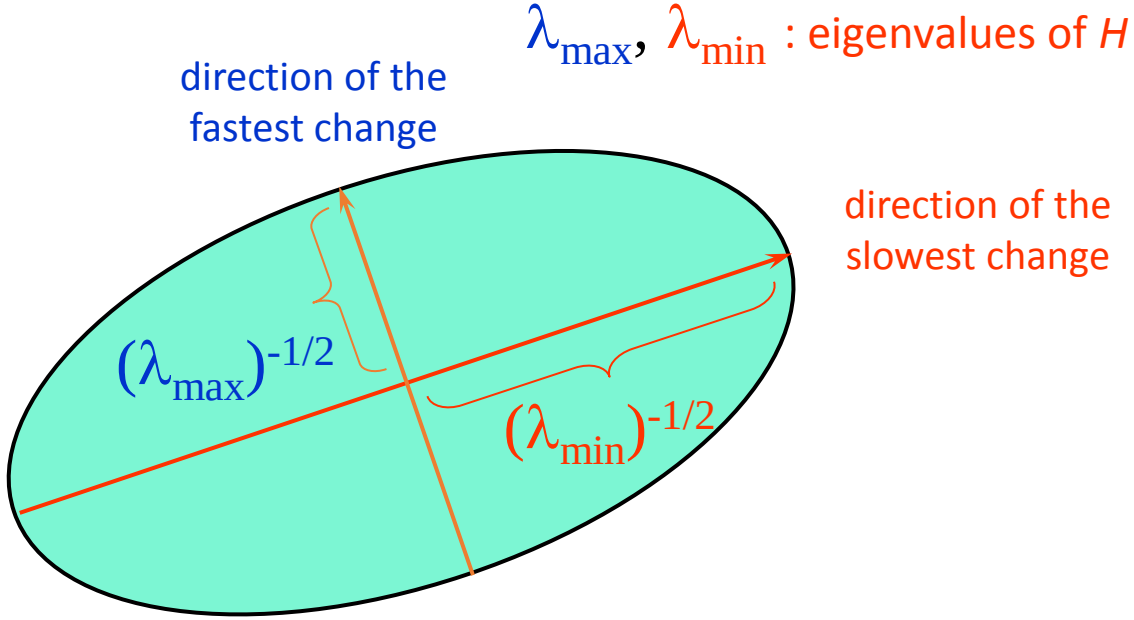
General Case

The shape of *H* tells us something about the *distribution of gradients* around a pixel

We can visualize *H* as an ellipse with axis lengths determined by the *eigenvalues* of *H* and orientation determined by the *eigenvectors* of *H*

Ellipse equation:

$$[u \;\; v] \;\; H \begin{bmatrix} u \\ v \end{bmatrix} = \text{const}$$



Quick Eigenvalue/Eigenvector review

The **eigenvectors** of a matrix **A** are the vectors **x** that satisfy:

$$Ax = \lambda x$$

The scalar λ is the **eigenvalue** corresponding to **x**

- The eigenvalues are found by solving:

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

- In our case, ***A*** = ***H*** is a 2x2 matrix, so we have
- The solution:

$$det \begin{bmatrix} h_{11} - \lambda & h_{12} \\ h_{21} & h_{22} - \lambda \end{bmatrix} = 0$$

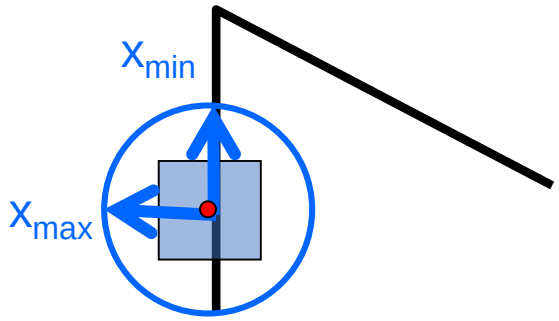
$$\lambda_{\pm} = \frac{1}{2} \left[(h_{11} + h_{22}) \pm \sqrt{4h_{12}h_{21} + (h_{11} - h_{22})^2} \right]$$

Once you know λ, you find **x** by solving

$$\begin{bmatrix} h_{11} - \lambda & h_{12} \\ h_{21} & h_{22} - \lambda \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 0$$

Corner Detection: The Math

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \underbrace{\begin{bmatrix} A & B \\ B & C \end{bmatrix}}_{H} \begin{bmatrix} u \\ v \end{bmatrix}$$



$$Hx_{\max} = \lambda_{\max} x_{\max}$$

$$Hx_{\min} = \lambda_{\min} x_{\min}$$

Eigenvalues and eigenvectors of H

- Define shift directions with the smallest and largest change in error
- x_{max} = direction of largest increase in *E*
- λ_{max} = amount of increase in direction x_{max}
- x_{min} = direction of smallest increase in *E*
- λ_{min} = amount of increase in direction x_{min}

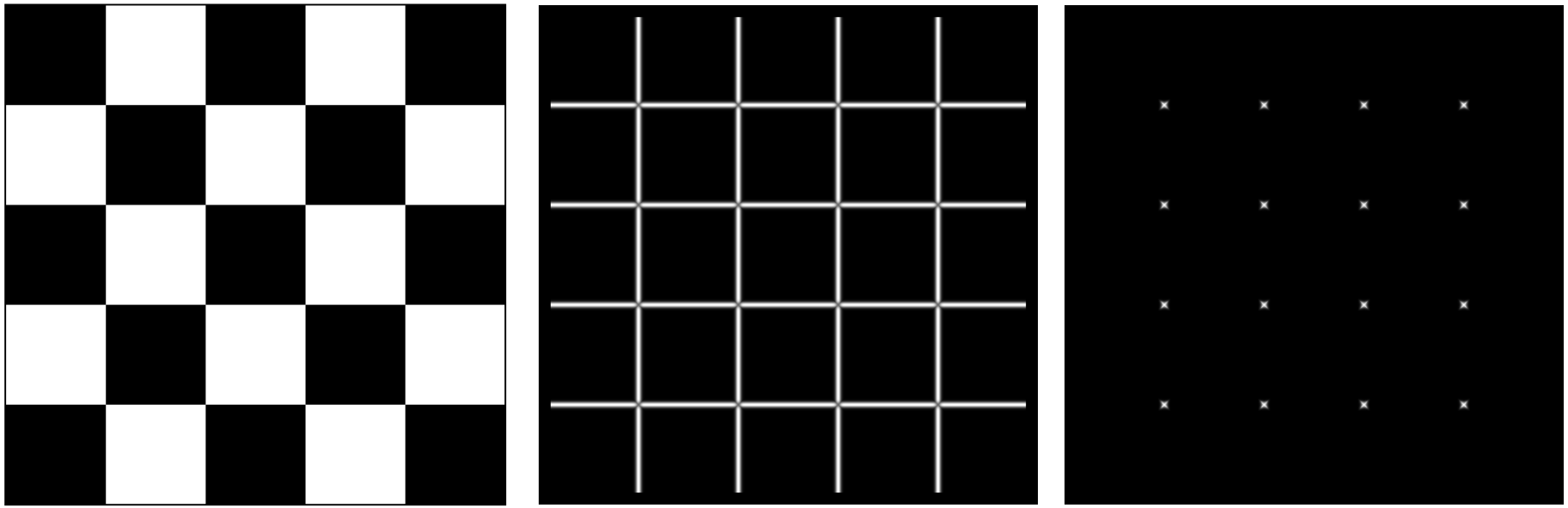
Corner Detection: The Math

How are λ_{max}, x_{max}, λ_{min}, and x_{min} relevant for feature detection?

- What's our feature scoring function?

Want $E(u,v)$ to be large for small shifts in all directions

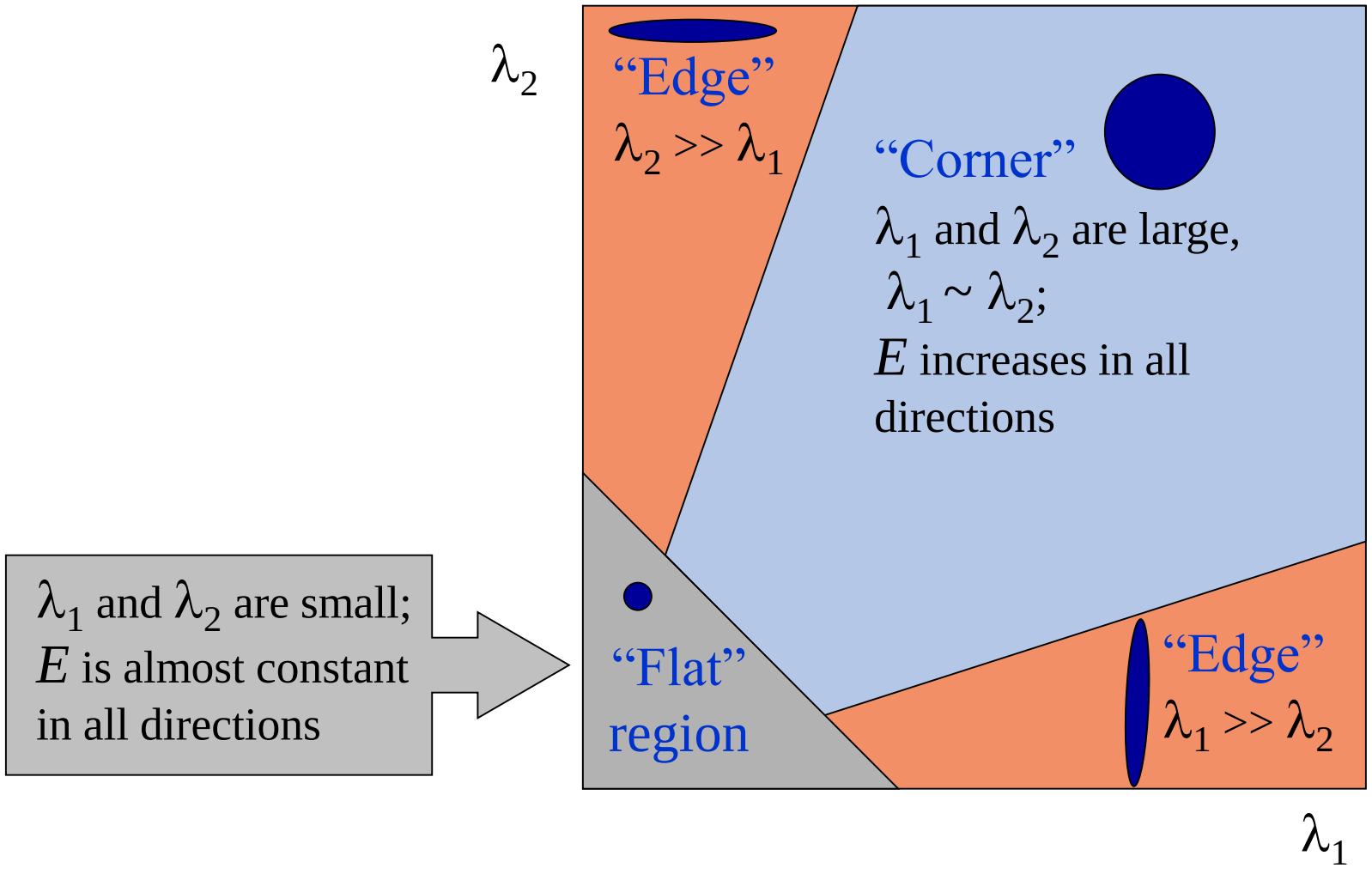
- the minimum of $E(u,v)$ should be large, over all unit vectors $[u\ v]$
- this minimum is given by the smaller eigenvalue (λ_{min}) of H



I λ_{max} λ_{min}

Interpreting Eigenvalues

Classification of image points using eigenvalues of *M*:



In some books, Corner Detection Math

- We can estimate the stability of the SSD score
- First, take the Taylor series expansion of the image
  - Recall Taylor series expansion of a function *f(x)* near x_0

$$f(x) = f(x_0) + \left[\frac{df}{dx}\right]_{x_0} (x - x_0) + \cdots$$

  - For vectors **x**, **u**

$$I_0(\mathbf{x}_i + \Delta\mathbf{u}) \approx I_0(\mathbf{x}_i) + \nabla I_0(\mathbf{x}_i) \cdot \Delta\mathbf{u}$$

- Then

$$E(\Delta\mathbf{u}) = \sum_i w(\mathbf{x}_i)\left[I_0(\mathbf{x}_i + \Delta\mathbf{u}) - I_0(\mathbf{x}_i)\right]^2$$

$$\approx \sum_i w(\mathbf{x}_i)\left[I_0(\mathbf{x}_i) + \nabla I_0(\mathbf{x}_i) \cdot \Delta\mathbf{u} - I_0(\mathbf{x}_i)\right]^2$$

$$= \sum_i w(\mathbf{x}_i)\left[\nabla I_0(\mathbf{x}_i) \cdot \Delta\mathbf{u}\right]^2$$

where

- $\mathbf{x}_i = (x_i, y_i)^T$ is some image point
- $\Delta\mathbf{u} = (\Delta x, \Delta y)^T$ is the displacement from that point
- $\nabla I_0(\mathbf{x}_i)$ is the gradient of the image at $\mathbf{x}_i$

In some books, Corner Detection Math

- Expand the SSD score

$$E(\Delta\mathbf{u}) = \sum_i w(\mathbf{x}_i)\left[\nabla I_0(\mathbf{x}_i)\cdot\Delta\mathbf{u}\right]^2$$

$$= \sum_i w(\mathbf{x}_i)\left[\Delta x\frac{\partial I_0(\mathbf{x}_i)}{\partial x} + \Delta y\frac{\partial I_0(\mathbf{x}_i)}{\partial y}\right]^2$$

$$= \sum_i w(\mathbf{x}_i)\left[\Delta x\left(\frac{\partial I_0(\mathbf{x}_i)}{\partial x}\right)^2\Delta x + 2\Delta x\left(\frac{\partial I_0(\mathbf{x}_i)}{\partial x}\right)\left(\frac{\partial I_0(\mathbf{x}_i)}{\partial y}\right)\Delta y + \Delta y\left(\frac{\partial I_0(\mathbf{x}_i)}{\partial y}\right)^2\Delta y\right]$$

$$= \Delta x\left[\sum_i w(\mathbf{x}_i)\left(\frac{\partial I_0(\mathbf{x}_i)}{\partial x}\right)^2\right]\Delta x + 2\Delta x\left[\sum_i w(\mathbf{x}_i)\left(\frac{\partial I_0(\mathbf{x}_i)}{\partial x}\right)\left(\frac{\partial I_0(\mathbf{x}_i)}{\partial y}\right)\right]\Delta y + \Delta y\left[\sum_i w(\mathbf{x}_i)\left(\frac{\partial I_0(\mathbf{x}_i)}{\partial y}\right)^2\right]\Delta y$$

$$= \Delta x\left(w * I_{xx}\right)\Delta x + 2\Delta x\left(w * I_{xy}\right)\Delta y + \Delta y\left(w * I_{yy}\right)\Delta y$$

$$= \begin{pmatrix}\Delta x & \Delta y\end{pmatrix}\begin{pmatrix}w * I_{xx} & w * I_{xy}\\ w * I_{xy} & w * I_{yy}\end{pmatrix}\begin{pmatrix}\Delta x\\ \Delta y\end{pmatrix}$$

- where $I_x^2 = \left(\frac{\partial I}{\partial x}\right)^2, I_{xy} = \left(\frac{\partial I}{\partial x}\right)\left(\frac{\partial I}{\partial y}\right), \quad I_y^2 = \left(\frac{\partial I}{\partial y}\right)^2$

In some books, Corner Detection Math

- The SSD score is

$$E(\Delta\mathbf{u}) = \begin{pmatrix}\Delta x & \Delta y\end{pmatrix}\begin{pmatrix} w*I_{xx} & w*I_{xy} \\ w*I_{xy} & w*I_{yy}\end{pmatrix}\begin{pmatrix}\Delta x \\ \Delta y\end{pmatrix}$$

$$= \Delta\mathbf{u}^T \mathbf{A} \Delta\mathbf{u}$$

$$I_x^2 = \left(\frac{\partial I}{\partial x}\right)^2, I_{xy} = \left(\frac{\partial I}{\partial x}\right)\left(\frac{\partial I}{\partial y}\right),$$

$$I_y^2 = \left(\frac{\partial I}{\partial y}\right)^2$$

- where

$$\mathbf{A} = w*\begin{pmatrix} I_x^2 & I_{xy} \\ I_{xy} & I_y^2\end{pmatrix}$$

Note – the "" indicates correlation of w with each element of* **A**

- *w* is a small *NxN* averaging filter such as a box filter
- it essentially averages the values over a small local neighborhood

- **A** can tell us how stable the SSD score is, at a given point

Possible Interest Point Measures

- Look at the smallest eigenvalue – if it is sufficiently large, then we have a candidate point (**Shi-Tomasi**)

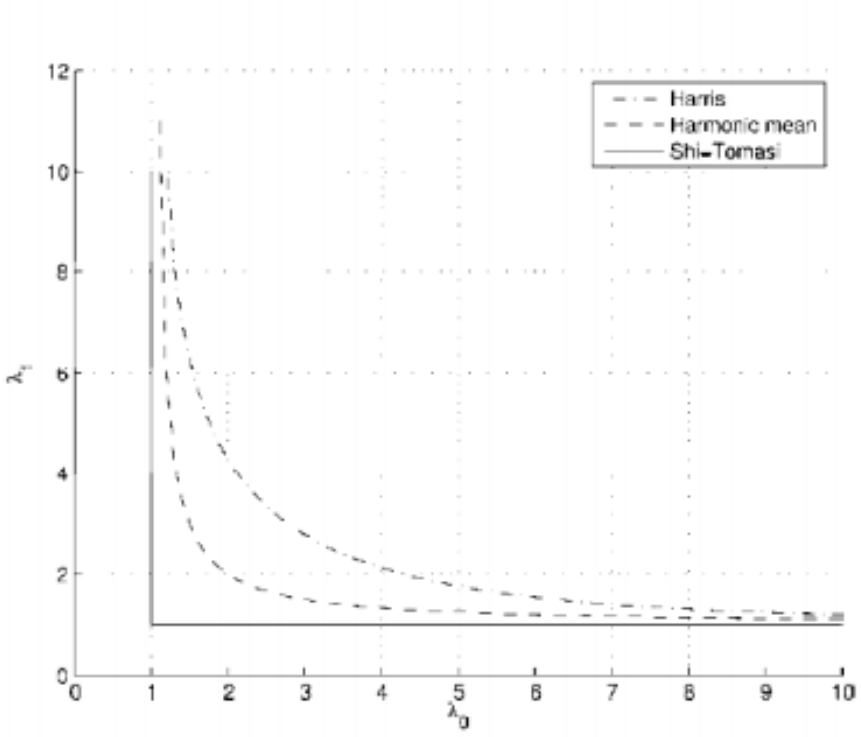
- Alternative measures that don't require square roots:

(Harris)

$$\det(\mathbf{A}) - \alpha\ \mathrm{tr}(\mathbf{A})^2 = \lambda_0\lambda_1 - \alpha(\lambda_0 + \lambda_1)^2$$

(Harmonic mean)

$$\frac{\det(\mathbf{A})}{\mathrm{tr}(\mathbf{A})} = \frac{\lambda_0\lambda_1}{\lambda_0 + \lambda_1}$$



From *Computer Vision: Algorithms and Applications*, Richard Szeliski, 2010, Springer

MATLAB Code

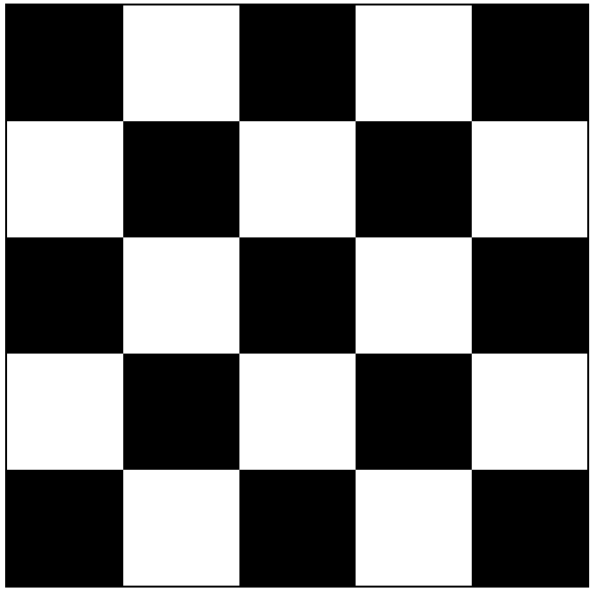
```
% Find corner points
clear all
close all

I = double(imread('test000.jpg'));

% Apply Gaussian blur
sd = 1.0;
I = imfilter(I, fspecial('gaussian', round(6*sd), sd));

imshow(I, []);

% Compute the gradient components
Gx = imfilter(I, [-1  1]);
Gy = imfilter(I, [-1; 1]);

% Compute the outer product g'g at each pixel
Gxx = Gx .* Gx;
Gxy = Gx .* Gy;
Gyy = Gy .* Gy;
```

```
N = 13;

w = ones(N);      % The neighborhood

% Sum the G's over the window size.
A11 = imfilter(Gxx, w);
A12 = imfilter(Gxy, w);
A22 = imfilter(Gyy, w);

% At each pixel (x,y), we have the 2x2 matrix
%   [A11(x,y)   A12(x,y);
%    A21(x,y)   A22(x,y)]
% Of course, A21 = A12.

% Compute the interest score:  det(A)/trace(A)
detA = A11.*A22 - A12.^2;    % Computes det(A) at each pixel
traceA = A11 + A22;          % Computer trace(A) at each pixel
s = detA./traceA;            % The interest score

% If trace(A)=0 anywhere, we get "not a number".
s(isnan(s)) = 0;         % If any NaN's, just set them to zero
```

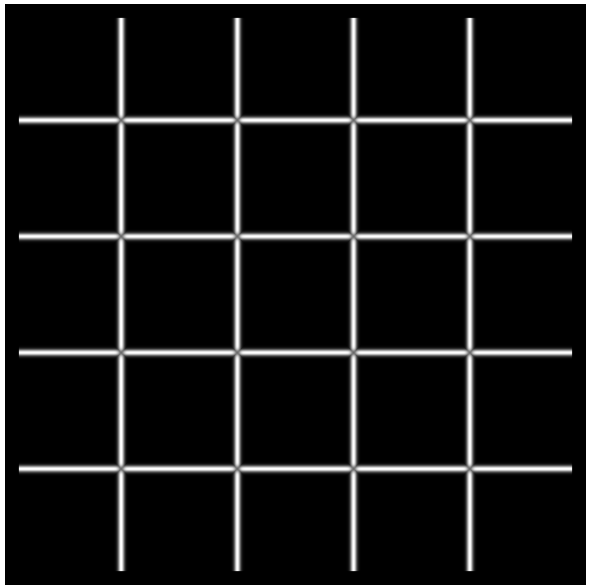
Corner Detection Summary

Here's what you do

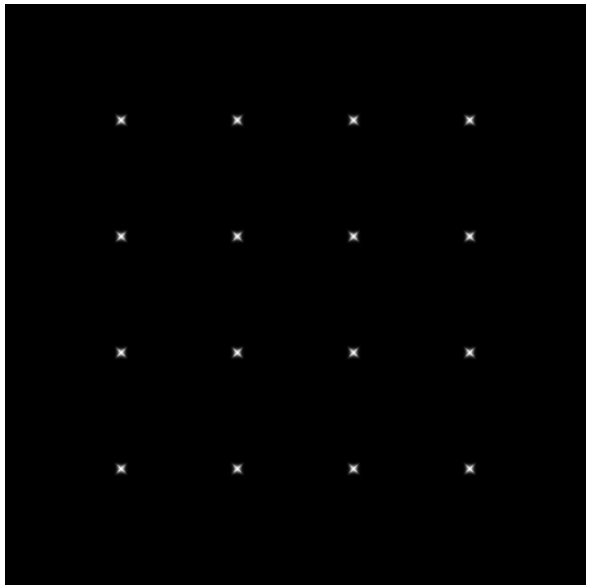
- Compute the gradient at each point in the image
- Create the *H* matrix from the entries in the gradient
- Compute the eigenvalues.
- Find points with large response (λ_{min} > threshold)
- Choose those points where λ_{min} is a local maximum as features



I



λ_{max}

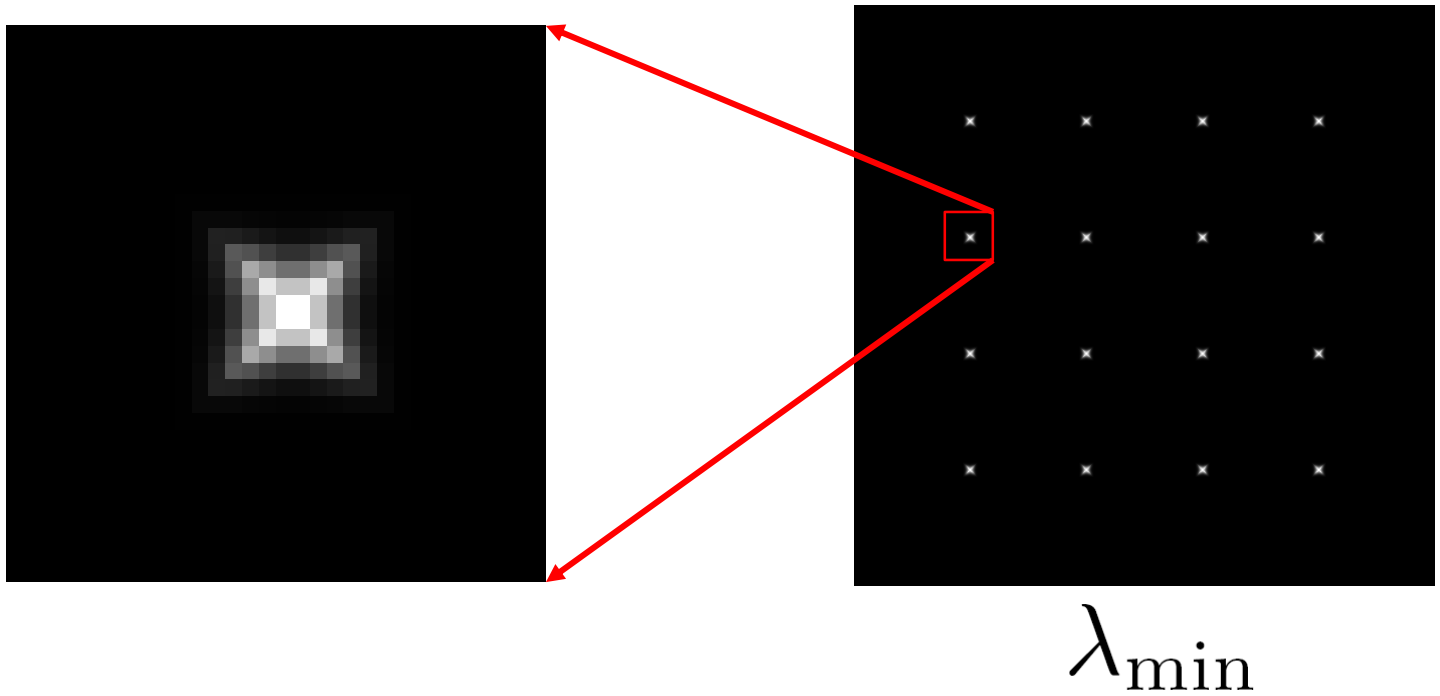


λ_{min}

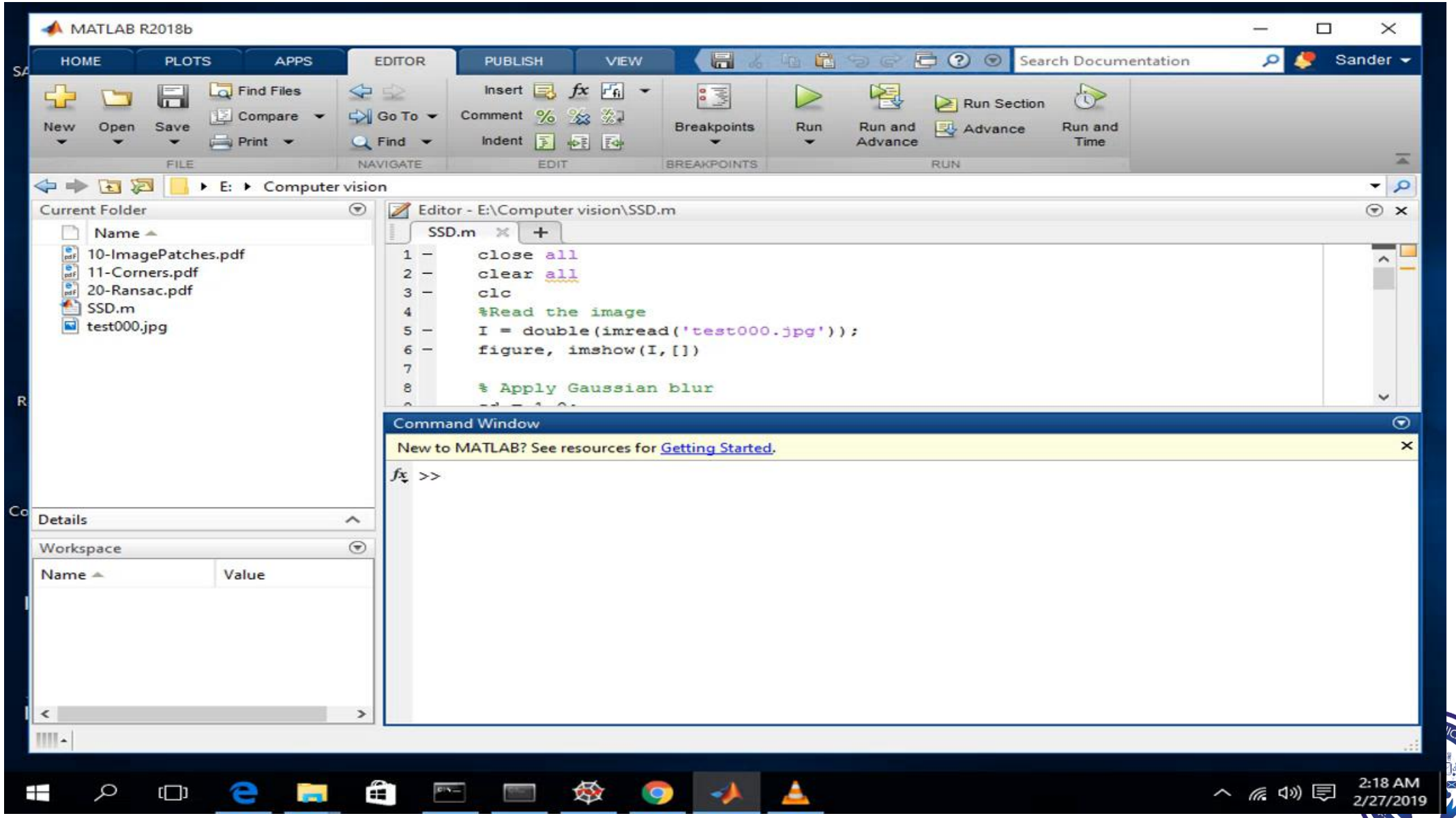
Corner Detection Summary

Here's what you do

- Compute the gradient at each point in the image
- Create the *H* matrix from the entries in the gradient
- Compute the eigenvalues.
- Find points with large response (λ_{min} > threshold)
- Choose those points where λ_{min} is a local maximum as features



Demo


MATLAB R2018b
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Search Documentation
Sander
New
Open
Save
Find Files
Compare
Print
Go To
Find
Insert
Comment
Indent
Breakpoints
Run
Run and Advance
Run Section
Advance
Run and Time
FILE
NAVIGATE
EDIT
BREAKPOINTS
RUN
E: ▸ Computer vision
Current Folder
Name
10-ImagePatches.pdf
11-Corners.pdf
20-Ransac.pdf
SSD.m
test000.jpg
Editor - E:\Computer vision\SSD.m
SSD.m
1 close all
2 clear all
3 clc
4 %Read the image
5 I = double(imread('test000.jpg'));
6 figure, imshow(I,[])
7
8 % Apply Gaussian blur
Command Window
New to MATLAB? See resources for Getting Started.
>>
Details
Workspace
Name
Value
2:18 AM
2/27/2019

The Harris Operator

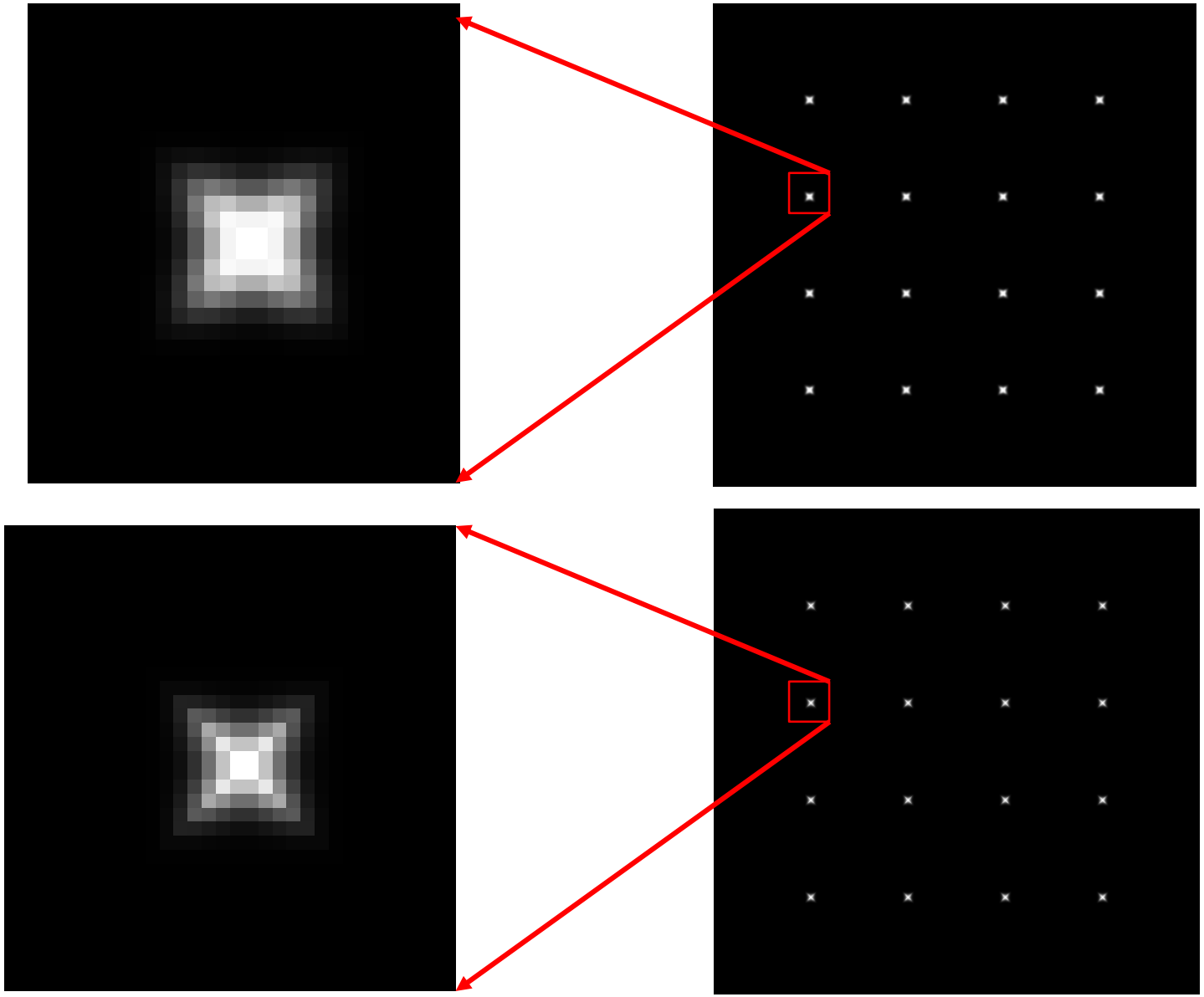
λ_{min} is a variant of the "Harris operator" for feature detection

$$f = \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2}$$

$$= \frac{determinant(H)}{trace(H)}$$

- The *trace* is the sum of the diagonals, i.e., *trace(H) = h_{11} + h_{22}*
- Very similar to λ_{min} but less expensive (no square root)
- Called the "Harris Corner Detector" or "Harris Operator"
- Lots of other detectors, this is one of the most popular

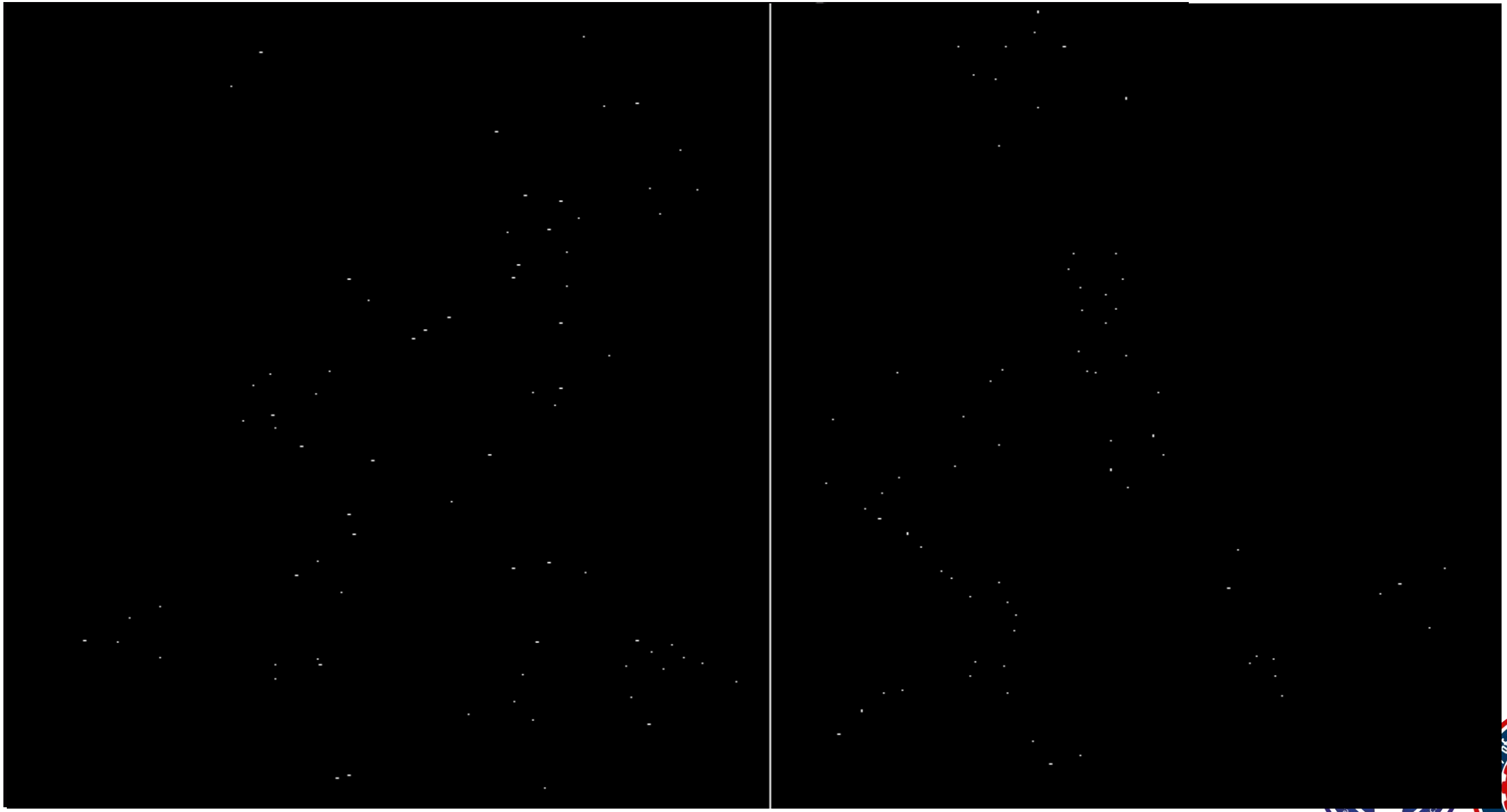
The Harris Operator



Harris operator

λ_{min}

Harris Detector Example



Find local maxima of f

Harris Features in Red

Code for Harris Corner Detection

```python
# organizing imports
import cv2
import numpy as np

# path to input image specified and
# image is loaded with imread command
image = cv2.imread('8.jpg')

# convert the input image into
# grayscale color space
operatedImage = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# modify the data type
# setting to 32-bit floating point
operatedImage = np.float32(operatedImage)

# apply the cv2.cornerHarris method
# to detect the corners with appropriate
# values as input parameters
dest = cv2.cornerHarris(operatedImage, 2, 5, 0.07)

# Results are marked through the dilated corners
dest = cv2.dilate(dest, None)

# Reverting back to the original image,
# with optimal threshold value
image[dest > 0.01 * dest.max()]=[0, 0, 255]

# the window showing output image with corners
cv2.imshow('Image with Borders', image)

# De-allocate any associated memory usage
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()

```

Results for Harris Corner Detection

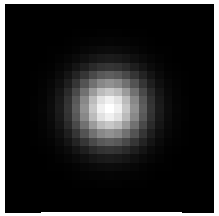
Weighting the derivatives

- In practice, using a simple window *W* doesn't work too well

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

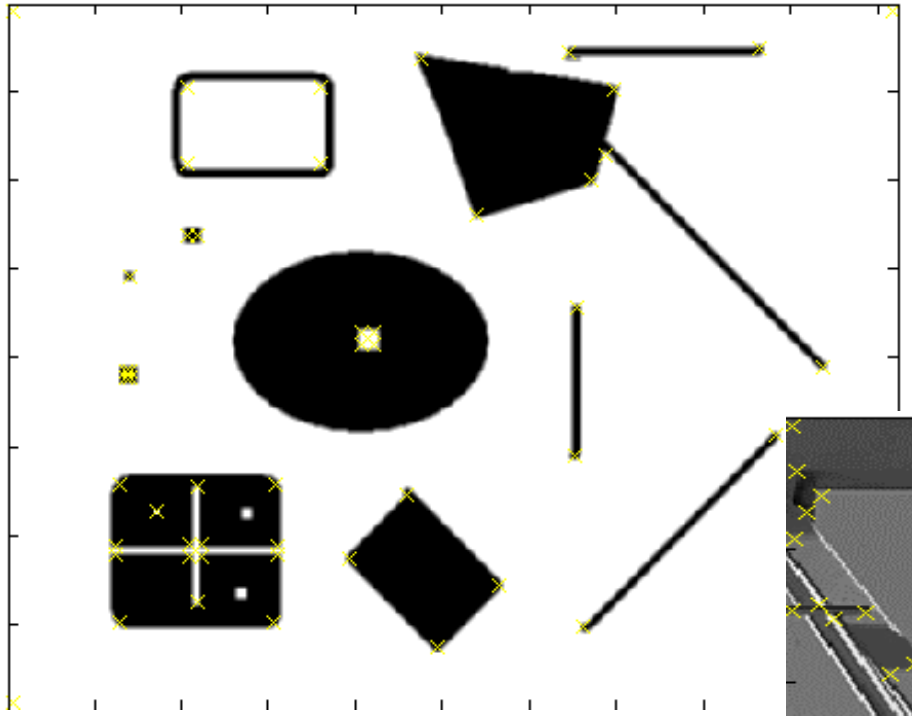
- Instead, we'll *weight* each derivative value based on its distance from the center pixel

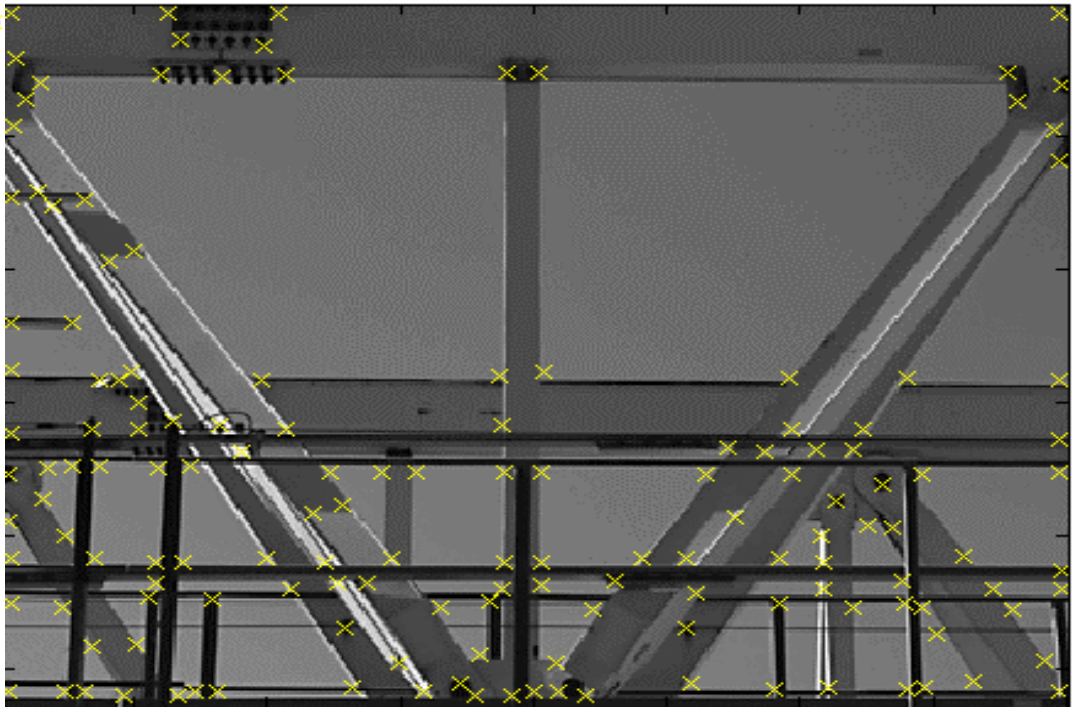
$$H = \sum_{(x,y)\in W} w_{x,y} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$



$w_{x,y}$

Harris Detector – Responses [Harris88]



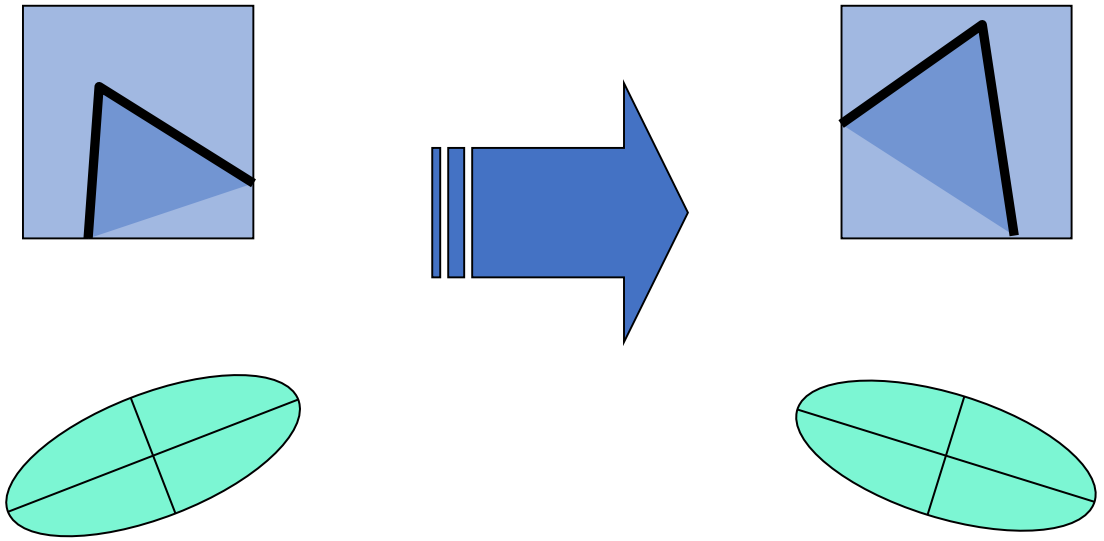


Effect: **A very precise corner detector.**

Harris Detector – Responses [Harris88]

Harris Detector: Invariance Properties

- Rotation



Ellipse rotates but its shape (i.e. eigenvalues) remains the same

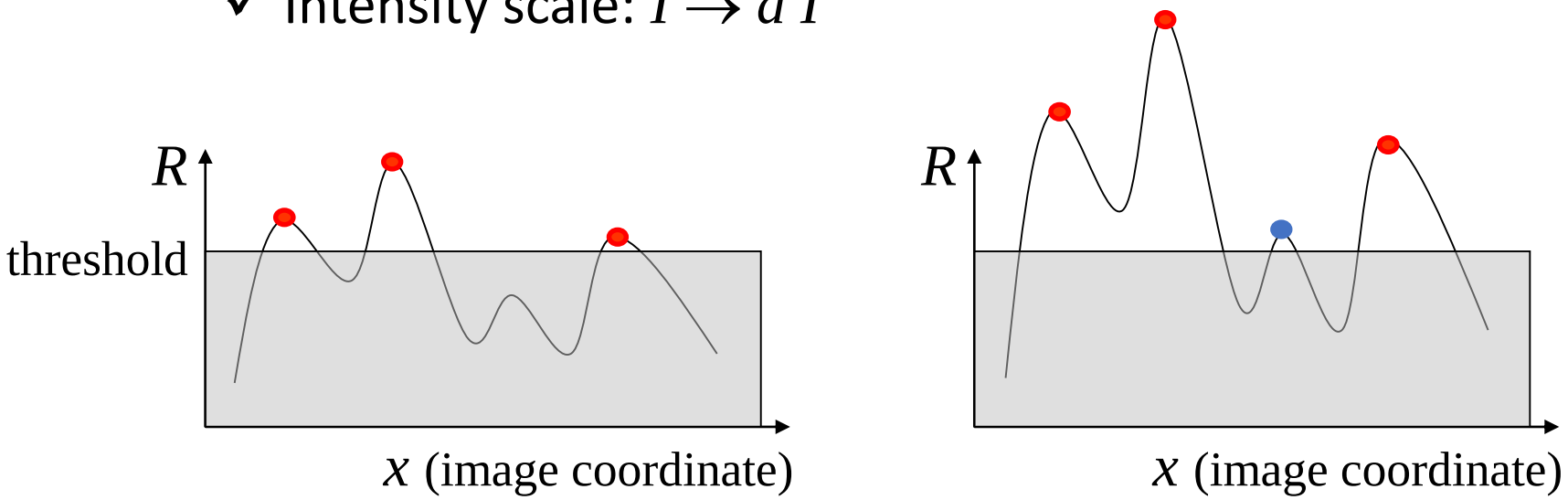
Corner response is invariant to image rotation

Harris Detector: Invariance Properties

- Affine intensity change: $I \rightarrow aI + b$

  ✓ Only derivatives are used =>
  invariance to intensity shift $I \rightarrow I + b$

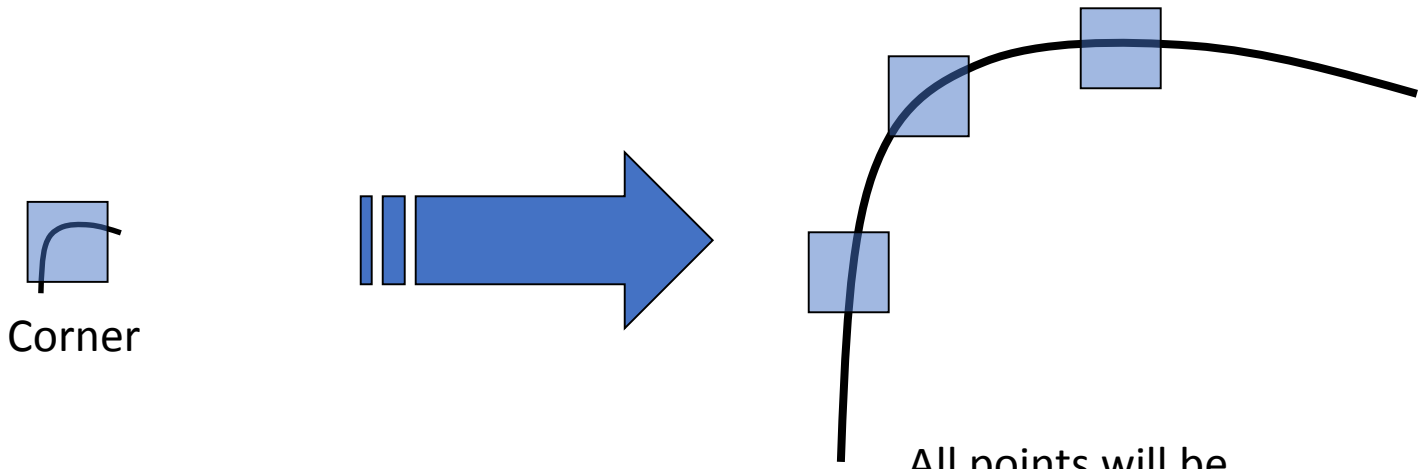
  ✓ Intensity scale: $I \rightarrow a\,I$



Partially invariant to affine intensity change

Harris Detector: Invariance Properties

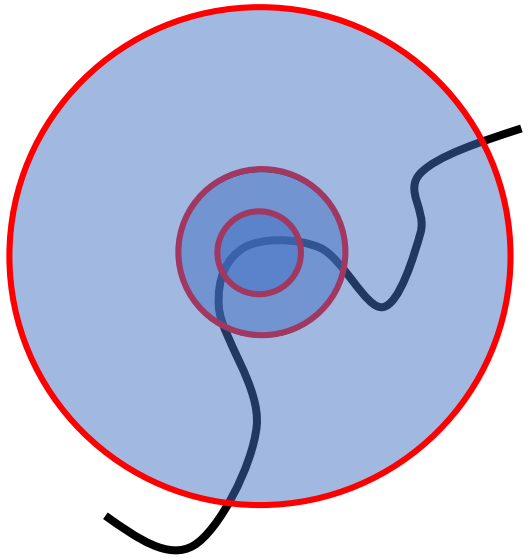
- Scaling



Not invariant to scaling

Scale Invariant Detection

Suppose you're looking for corners



Key idea: find scale that gives local maximum of *f*

- in both position and scale
- One definition of *f*: the Harris operator

Automatic Scale Selection

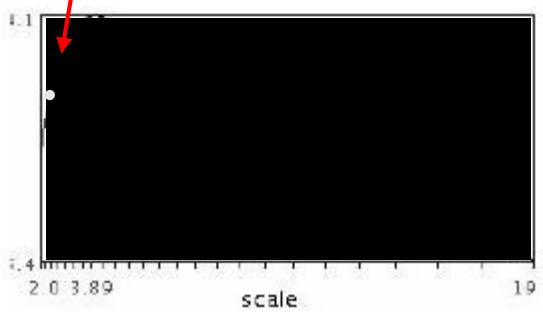
$$f(I_{i_1 \ldots i_m}(x,\sigma)) \quad = \quad f(I_{i_1 \ldots i_m}(x',\sigma'))$$

How to find corresponding patch sizes?

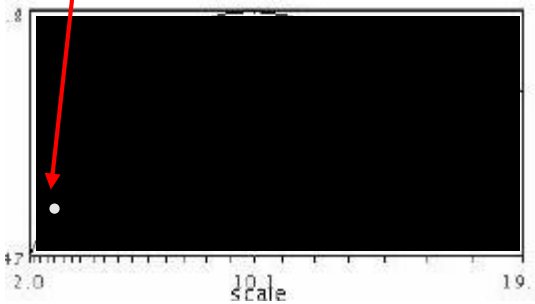
K. Grauman, B. Leibe

Automatic Scale Selection

- Function responses for increasing scale (scale signature)



$f(I_{i_1 \ldots i_m}(x, \sigma))$

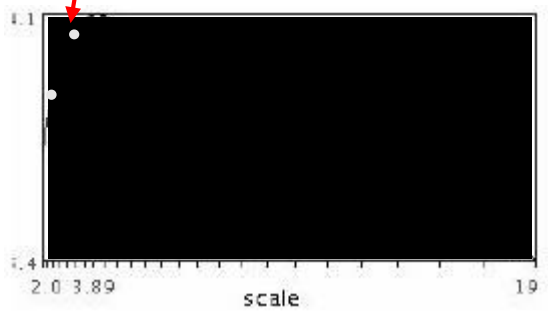


$f(I_{i_1 \ldots i_m}(x', \sigma))$

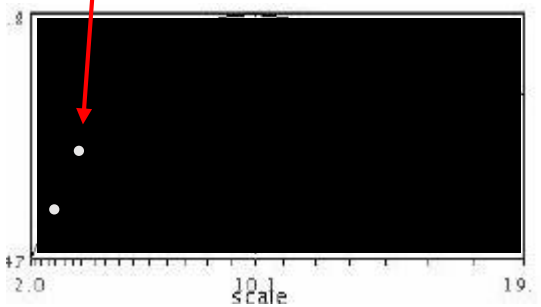
K. Grauman, B. Leibe

Automatic Scale Selection

- Function responses for increasing scale (scale signature)



$f(I_{i_1 \ldots i_m}(x,\sigma))$

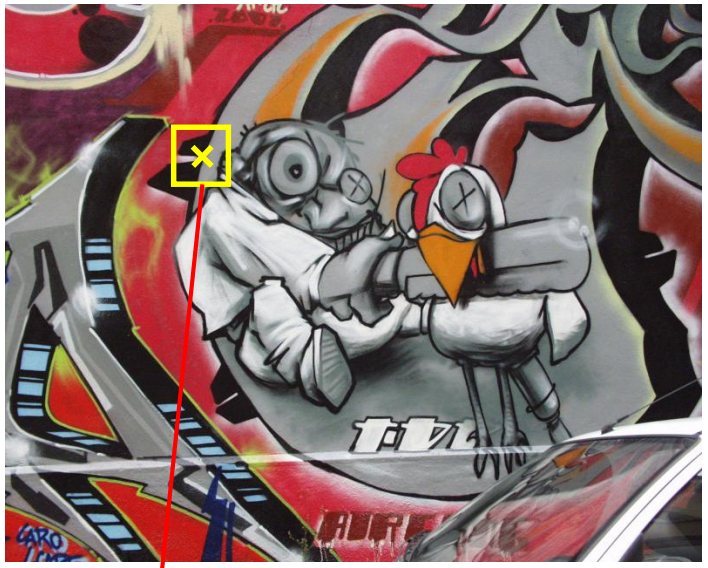


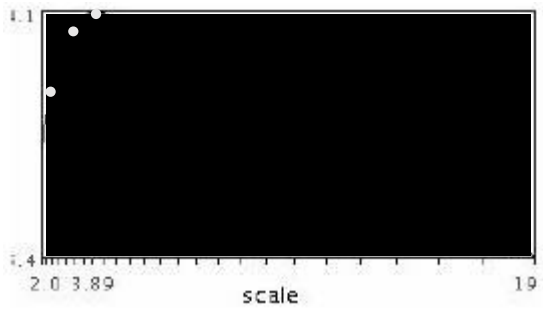
$f(I_{i_1 \ldots i_m}(x',\sigma))$

K. Grauman, B. Leibe

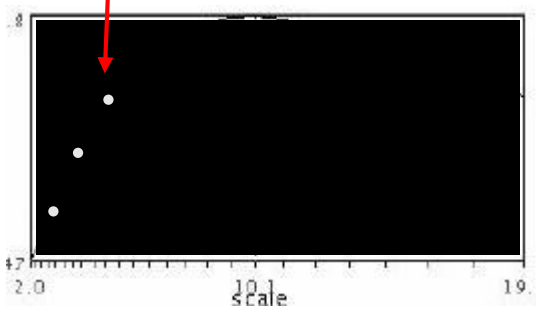
Automatic Scale Selection

- Function responses for increasing scale (scale signature)





$f(I_{i_1 \ldots i_m}(x,\sigma))$

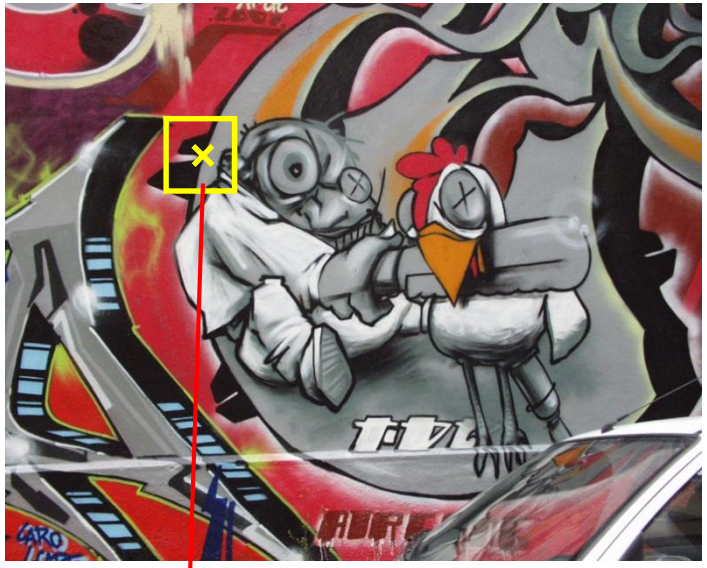


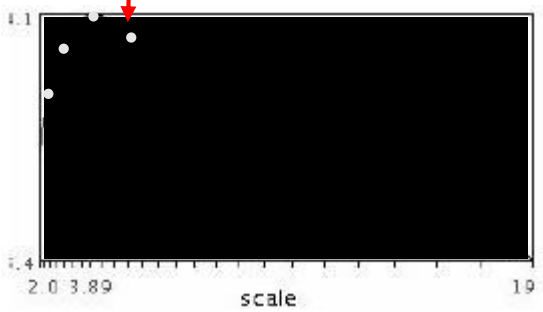
$f(I_{i_1 \ldots i_m}(x',\sigma))$

K. Grauman, B. Leibe

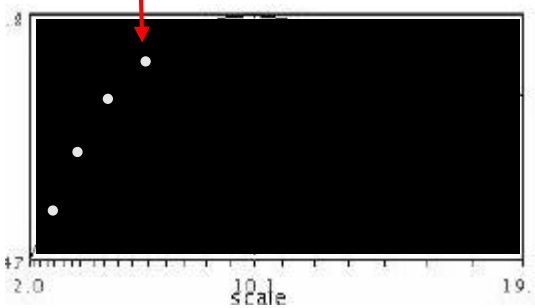
Automatic Scale Selection

- Function responses for increasing scale (scale signature)





$f(I_{i_1 \ldots i_m}(x, \sigma))$

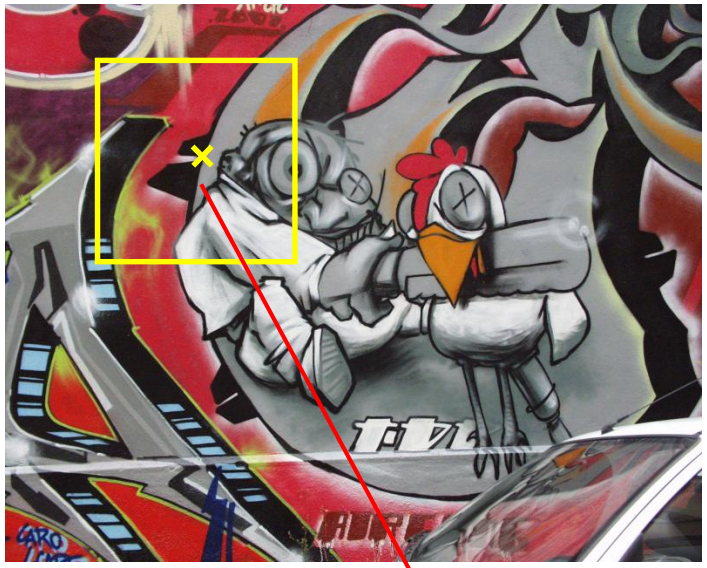


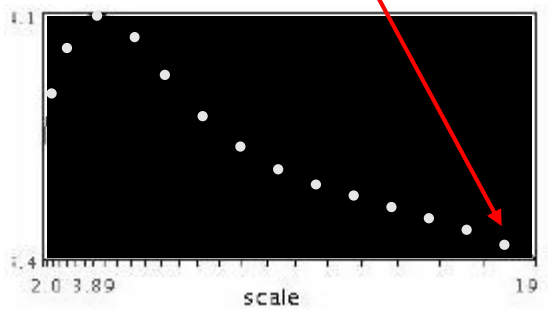
$f(I_{i_1 \ldots i_m}(x', \sigma))$

K. Grauman, B. Leibe

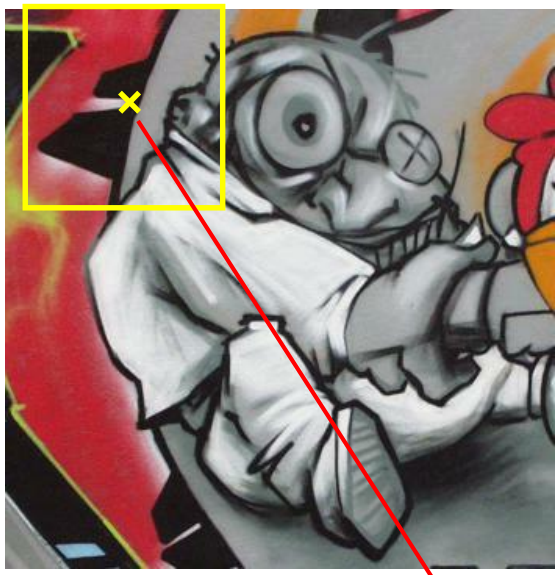
Automatic Scale Selection

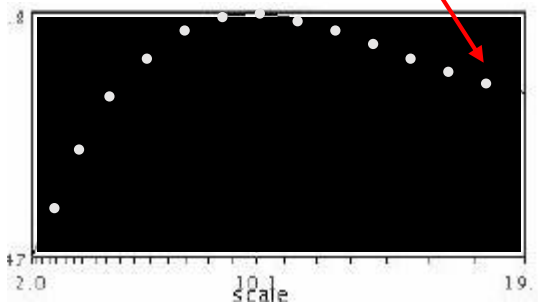
- Function responses for increasing scale (scale signature)





$f(I_{i_1 \ldots i_m}(x,\sigma))$



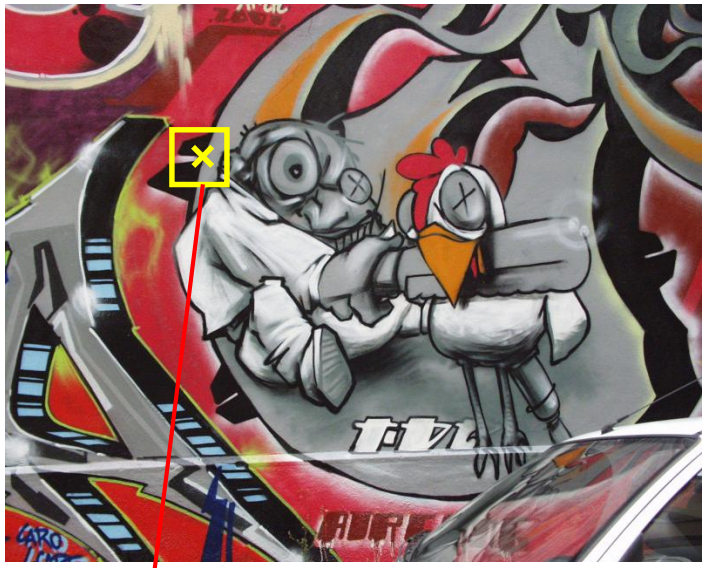


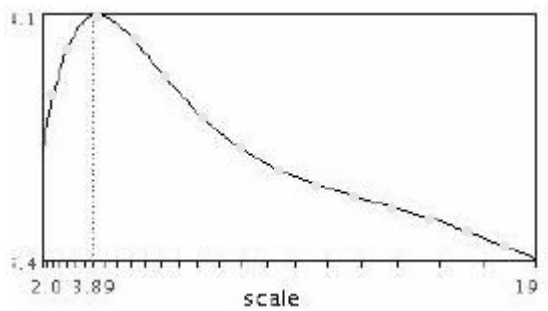
$f(I_{i_1 \ldots i_m}(x',\sigma))$

K. Grauman, B. Leibe

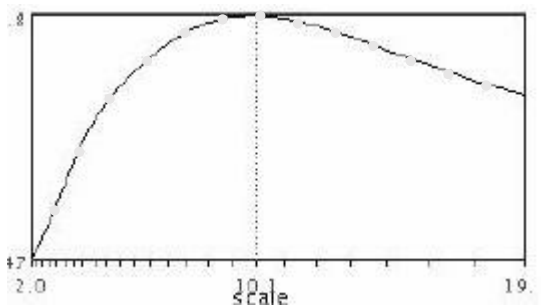
Automatic Scale Selection

- Function responses for increasing scale (scale signature)





$f(I_{i_1 \ldots i_m}(x, \sigma))$

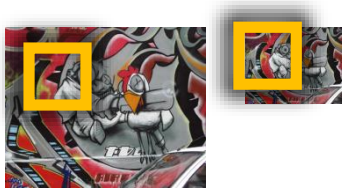


$f(I_{i_1 \ldots i_m}(x', \sigma'))$

K. Grauman, B. Leibe

Implementation

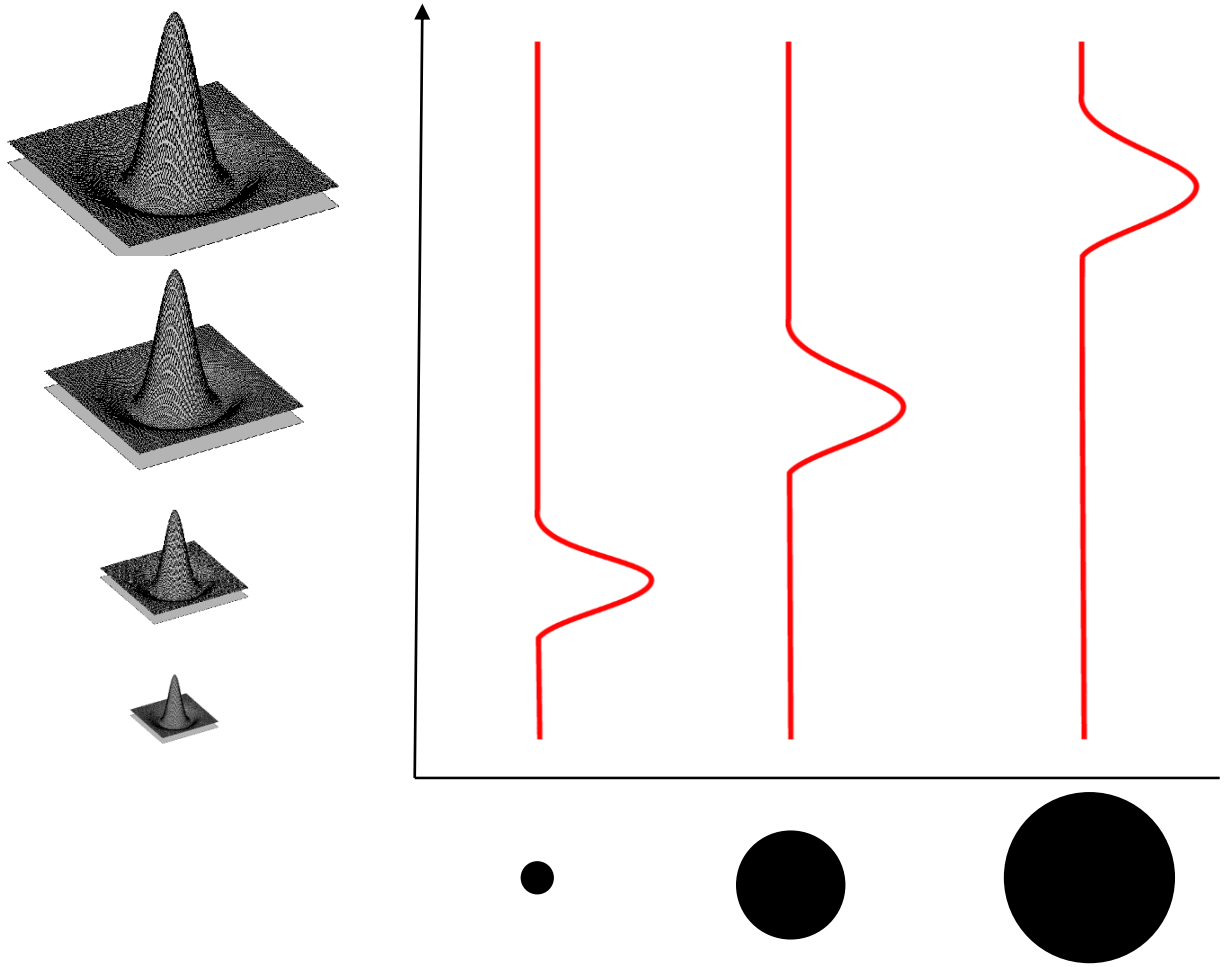
- Instead of computing *f* for larger and larger windows, we can implement using a fixed window size with a Gaussian pyramid



(sometimes need to create in-between levels, e.g. a ¾-size image)

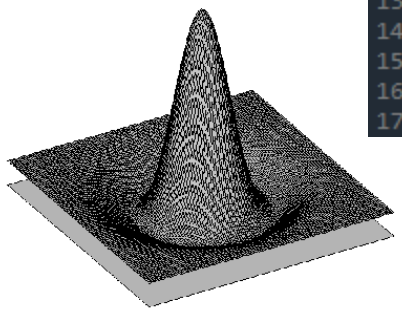
What is a useful signature function

- Difference-of-Gaussian = “blob” detector



K. Grauman, B. Leibe

Difference of Gaussian



```python
import cv2

img = cv2.imread('54.jpg')

# Apply 3x3 and 7x7 Gaussian blur
low_sigma = cv2.GaussianBlur(img,(3,3),0)
high_sigma = cv2.GaussianBlur(img,(5,5),0)

# Calculate the DoG by subtracting
dog = low_sigma - high_sigma

# the window showing output image with corners
cv2.imshow('Difference of Gaussian', dog)

# De-allocate any associated memory usage
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```

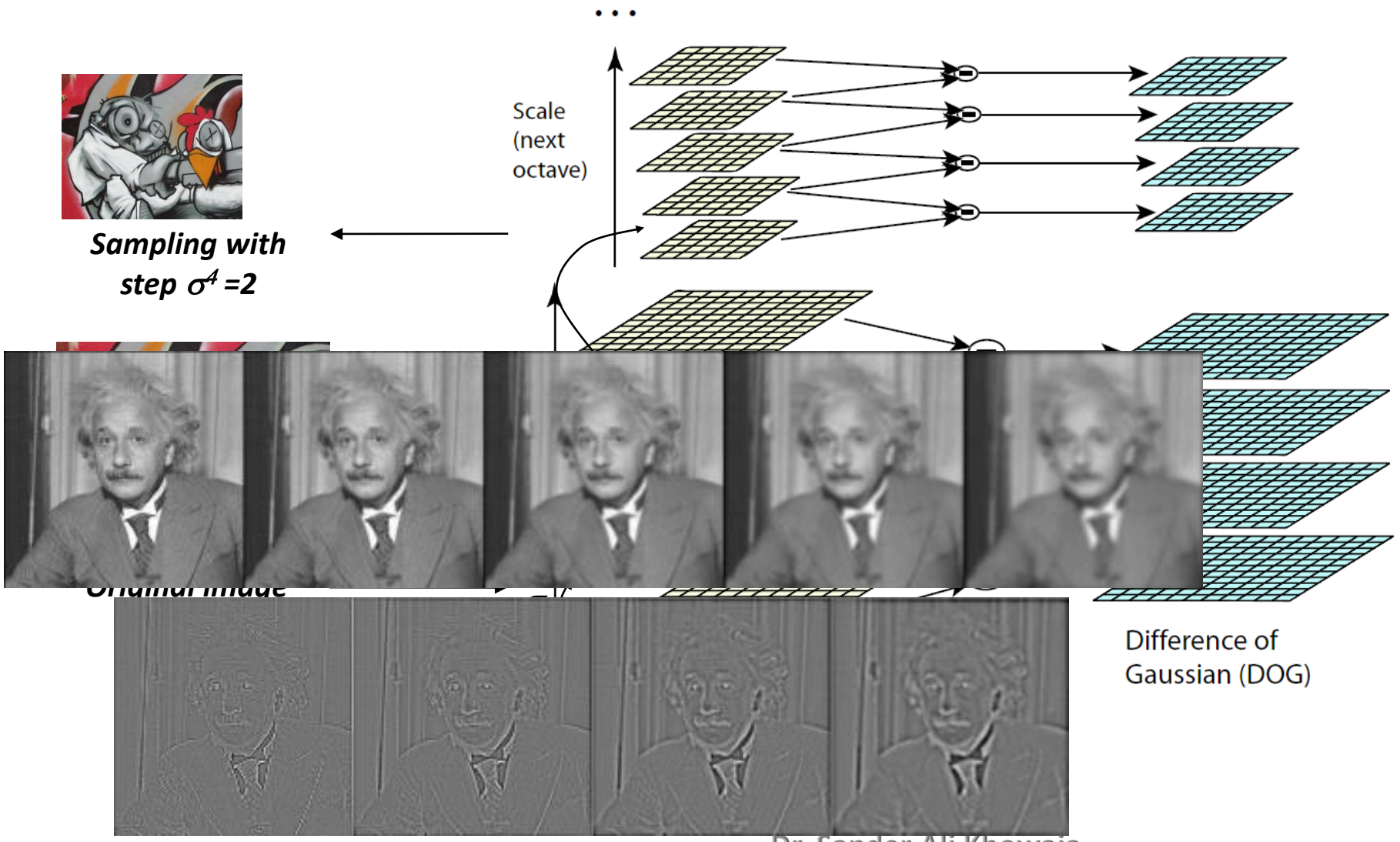
-

=

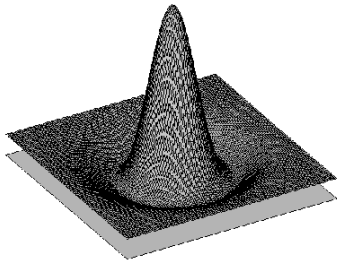
K. Grauman, B. Leibe

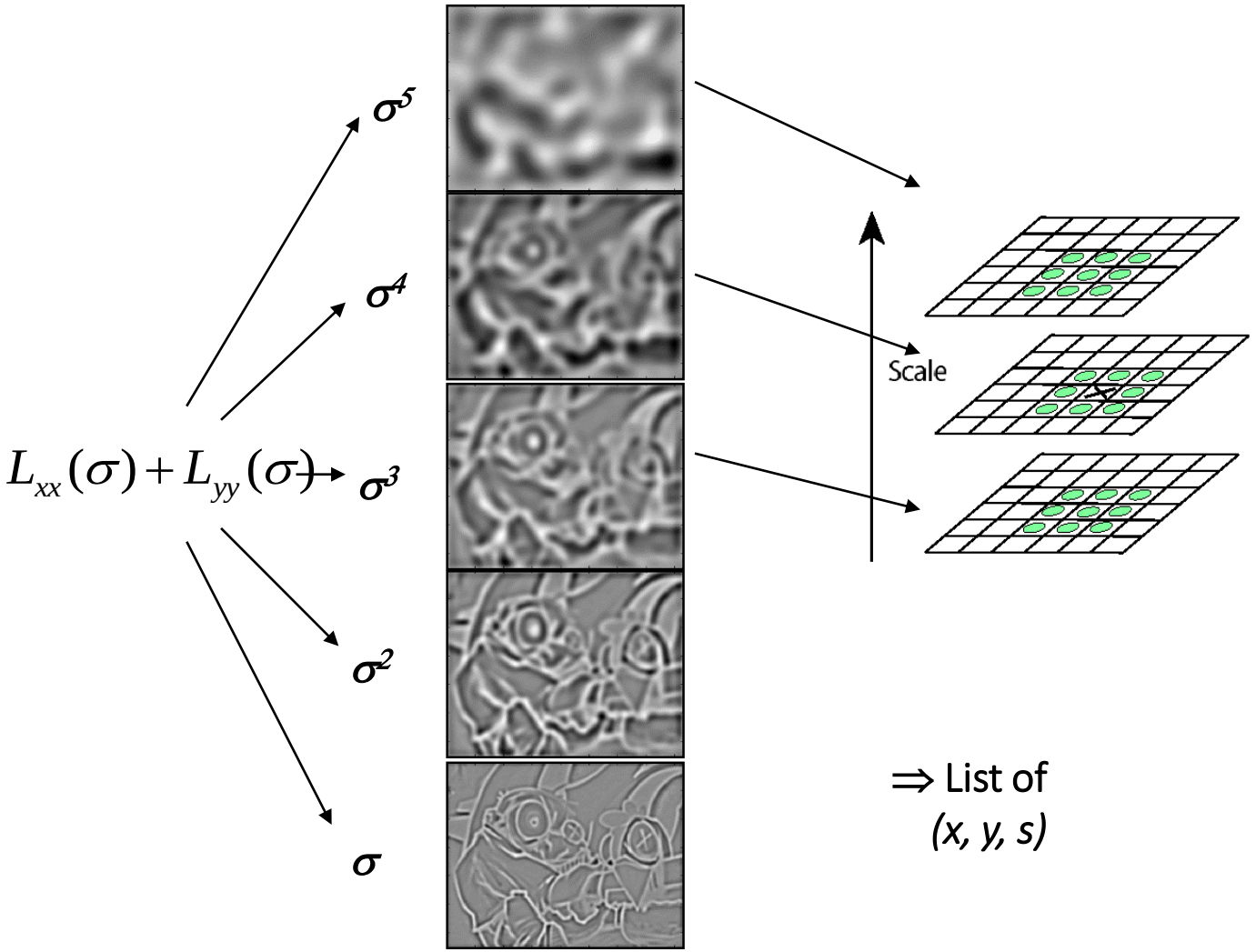
Difference of Gaussian – Efficient Computation

- Computation in Gaussian scale pyramid



Find Local Maxima in Position-scale space of Difference-of-Gaussian



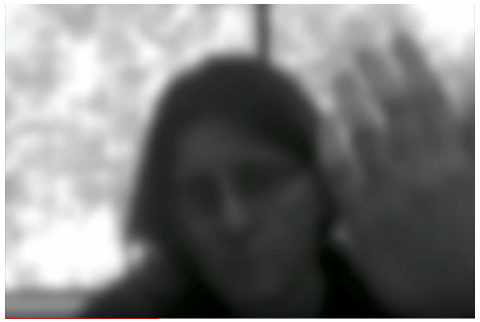


⇒ List of
(x, y, s)

K. Grauman, B. Leibe

Example of Gaussian Kernel

Original video



Blurred with a gaussian kernel

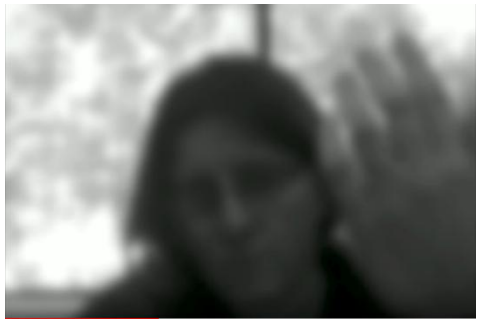
Blurred with a different gaussian kernel

What happens if you subtract one blurred image from another?

source

Difference of Gaussians

Original video



Blurred with a gaussian kernel: k_1

Blurred with a different gaussian kernel: k_2

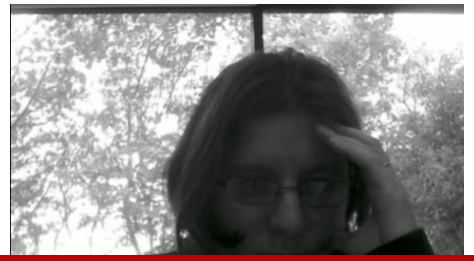
DoG: $k_1 - k_2$

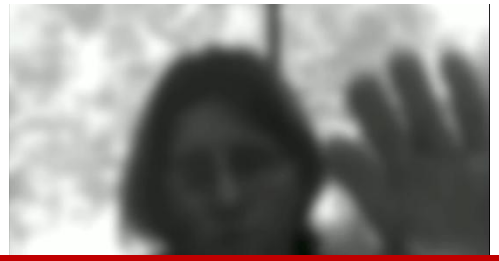
DoG: $k_1 - k_3$

DoG: $k_1 - k_4$

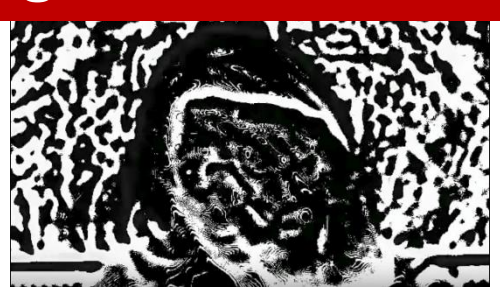
Source: https://www.youtube.com/watch?v=oTud1De_W4s

Difference of Gaussians





At different resolutions of kernel size, we see different fine details of the image. In other words, we can capture keypoints at varying scales.

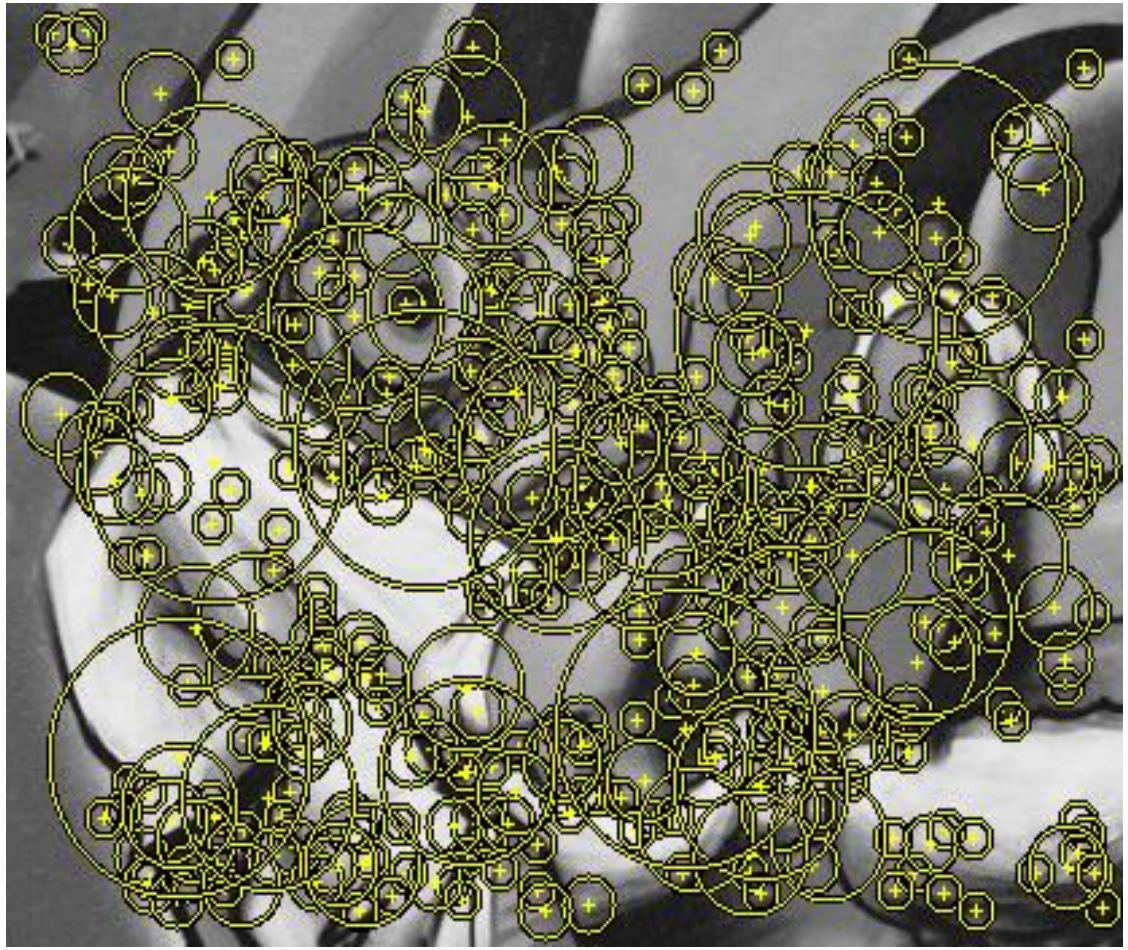


DoG: $k_1 - k_2$

DoG: $k_1 - k_3$

DoG: $k_1 - k_4$

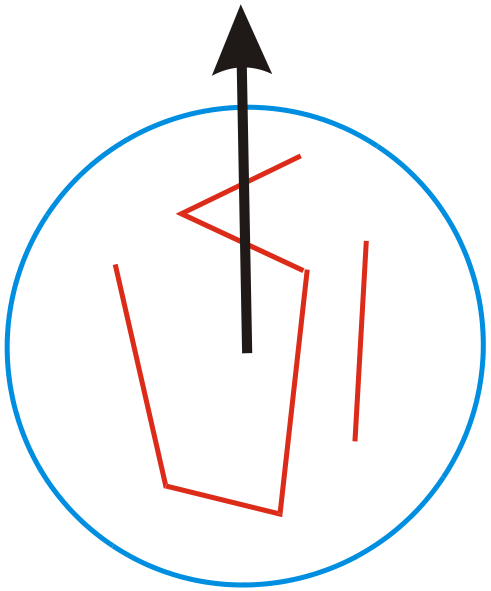
Results of Interest points using Difference of Gaussian

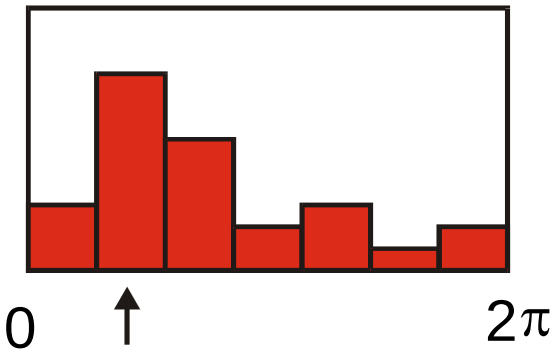


Orientation Normalization

[Lowe, SIFT, 1999]

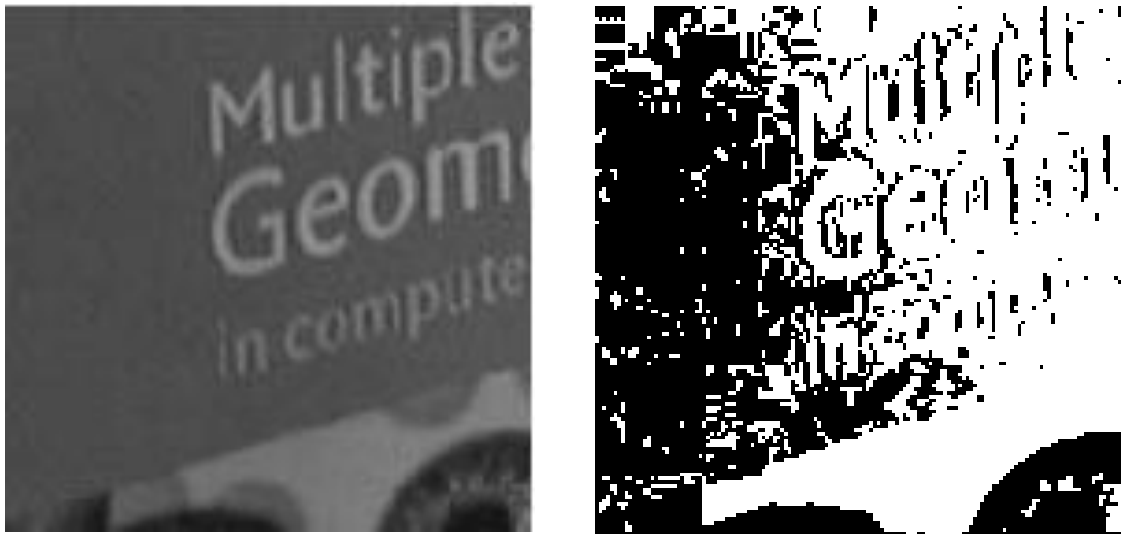
- Compute orientation histogram
- Select dominant orientation
- Normalize: rotate to fixed orientation





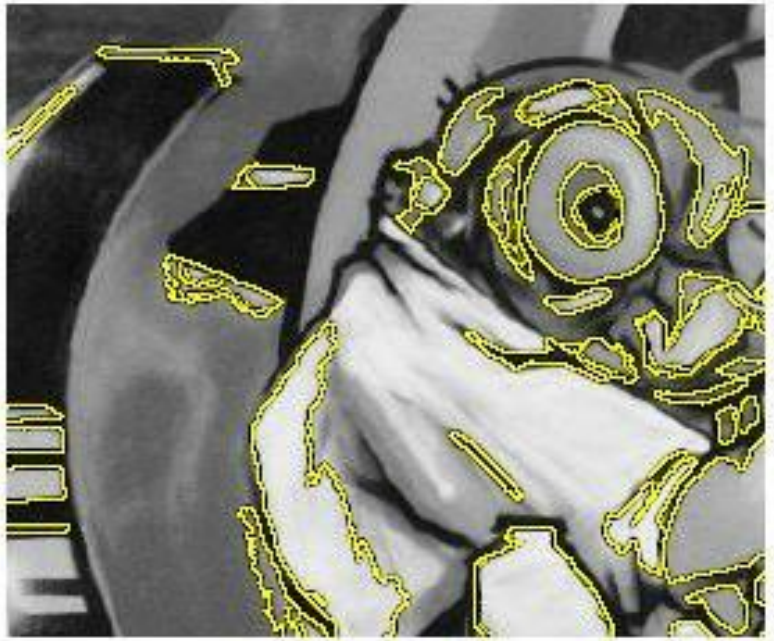
Maximally Stable Extremal Regions [Matas '02]

- Based on Watershed segmentation algorithm
- Select regions that stay stable over a large parameter range



K. Grauman, B. Leibe

Example Results: MSER



K. Grauman, B. Leibe

Feature Detector codes

- For most local feature detectors, executables are available online:
  - http://www.robots.ox.ac.uk/~vgg/research/affine
  - http://www.cs.ubc.ca/~lowe/keypoints/
  - http://www.vision.ee.ethz.ch/~surf

Local Descriptors

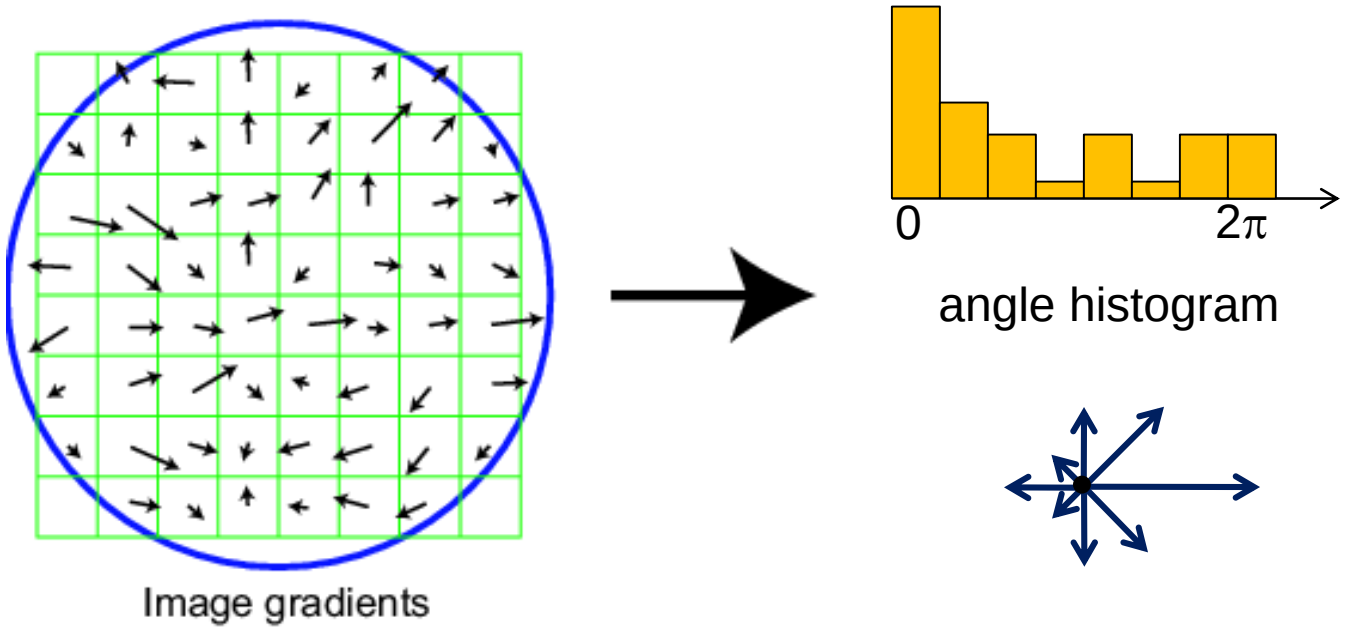
- The ideal descriptor should be
  - Robust
  - Distinctive
  - Compact
  - Efficient

- Most available descriptors focus on edge/gradient information
  - Capture texture information
  - Color rarely used

Scale Invariant Feature Transform (SIFT)

Basic idea:

- Take 16x16 square window around detected feature
- Compute edge orientation (angle of the gradient - 90°) for each pixel
- Throw out weak edges (threshold gradient magnitude)
- Create histogram of surviving edge orientations

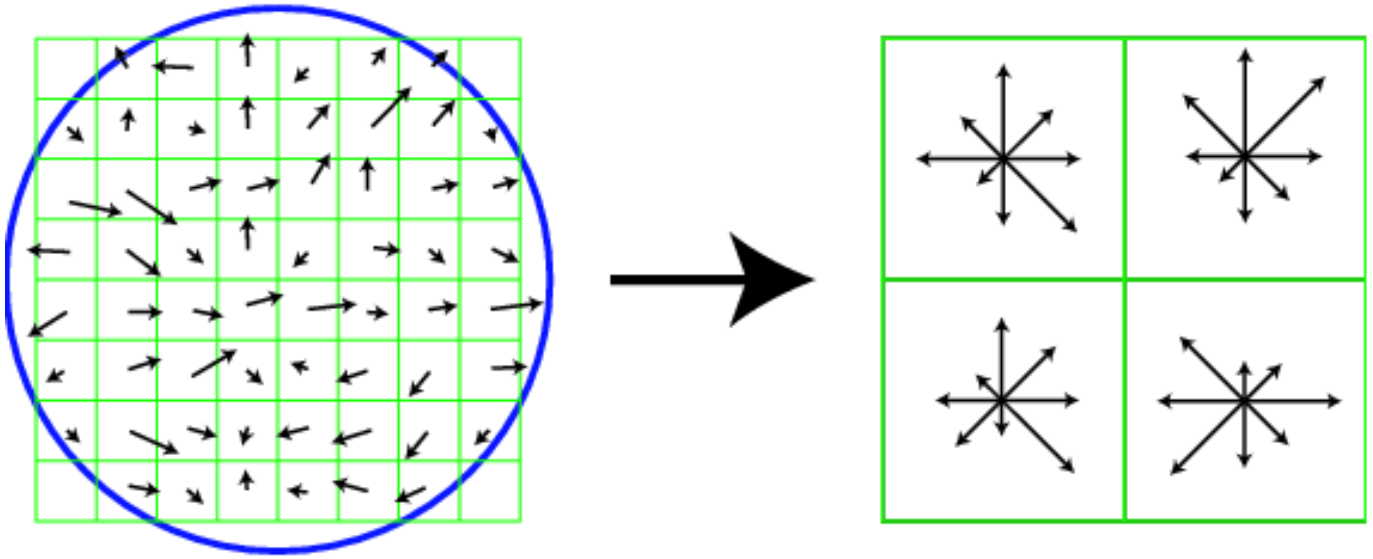


Adapted from slide by David Lowe

SIFT Descriptor

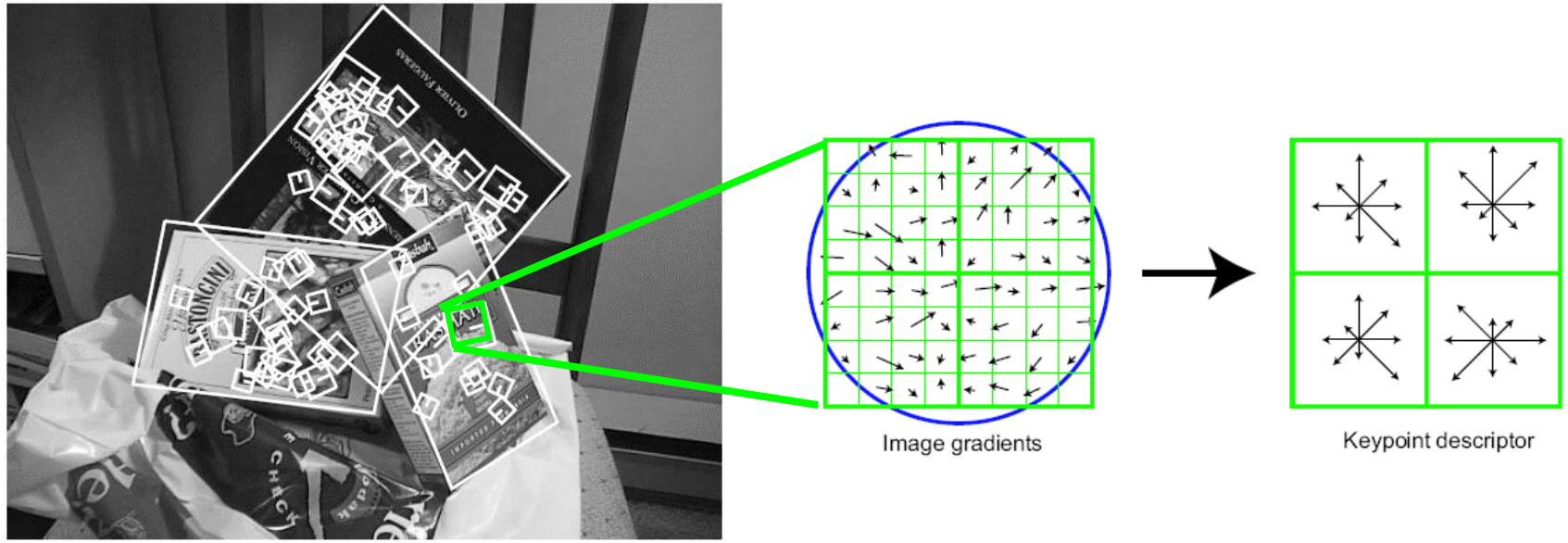
Full version

- Divide the 16x16 window into a 4x4 grid of cells (2x2 case shown below)
- Compute an orientation histogram for each cell
- 16 cells * 8 orientations = 128 dimensional descriptor



Adapted from slide by David Lowe

Local Descriptors: SIFT Descriptor



Histogram of oriented gradients

- **Captures important texture information**
- **Robust to small translations / affine deformations**

[Lowe, ICCV 1999]

K. Grauman, B. Leibe

Details of Lowe's SIFT Algorithm

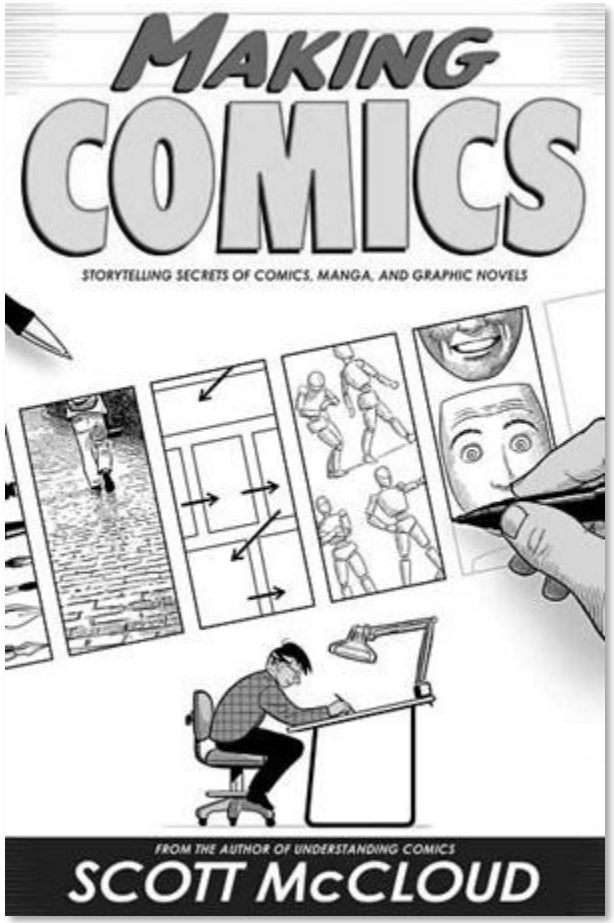
$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

$$\frac{\mathrm{Tr}(\mathbf{H})^2}{\mathrm{Det}(\mathbf{H})} < \frac{(r+1)^2}{r}$$

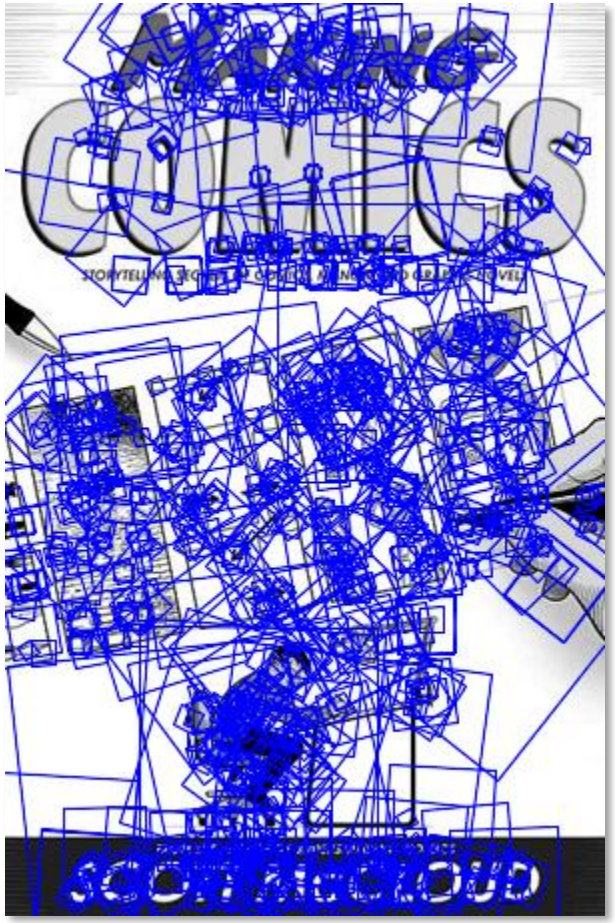
- Run DoG detector
  - Find maxima in location/scale space
  - Remove edge points
- Find all major orientations
  - Bin orientations into 36 bin histogram
    - Weight by gradient magnitude
    - Weight by distance to center (Gaussian-weighted mean)
  - Return orientations within 0.8 of peak
    - Use parabola for better orientation fit
- For each (x,y,scale,orientation), create descriptor:
  - Sample 16x16 gradient mag. and rel. orientation
  - Bin 4x4 samples into 4x4 histograms
  - Threshold values to max of 0.2, divide by L2 norm
  - Final descriptor: 4x4x8 normalized histograms

Lowe IJCV 2004

SIFT Example



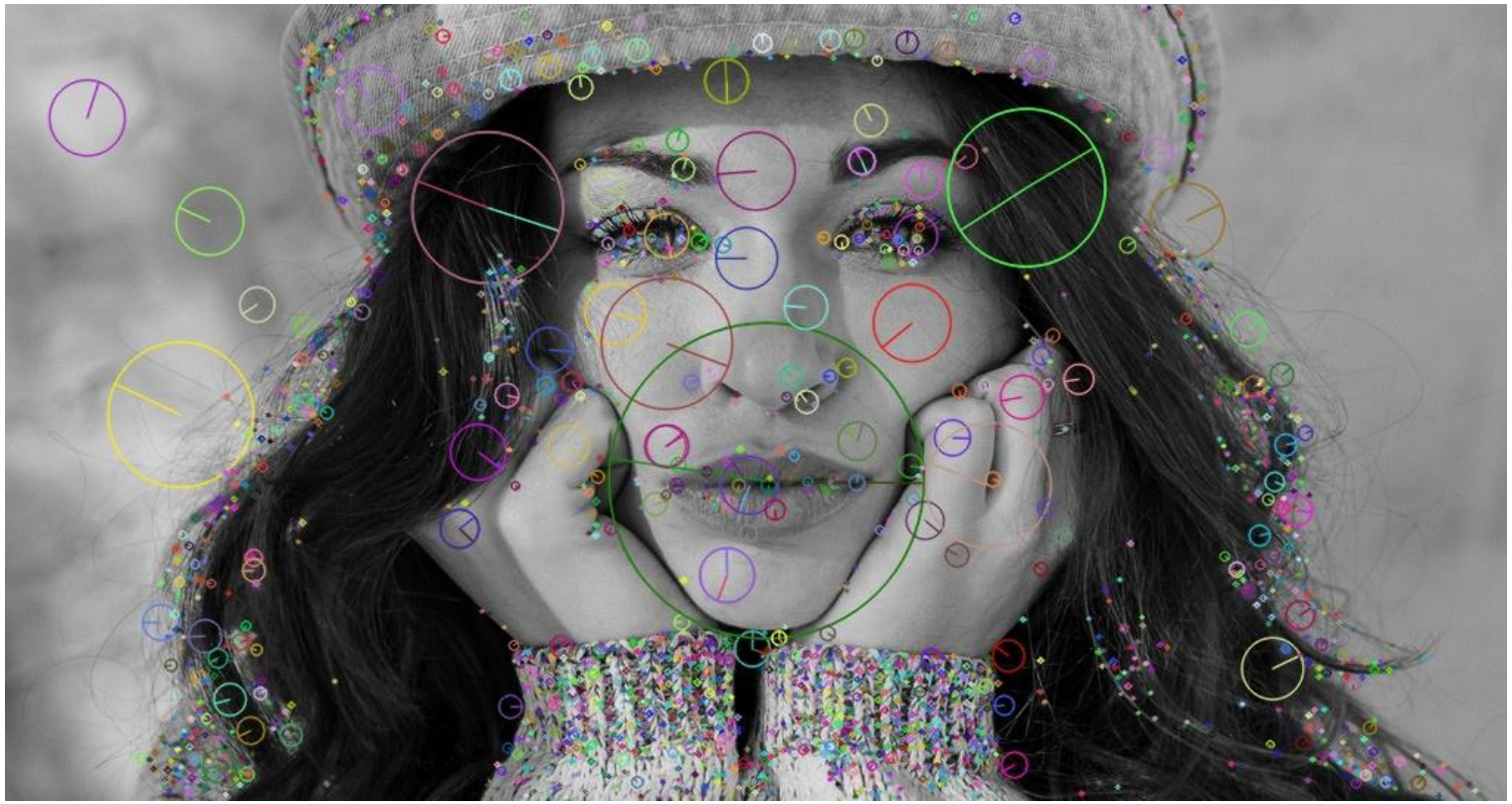
sift



868 SIFT features

SIFT Python

```python
# Important NOTE: Use opencv <= 3.4.2.16 as
# SIFT is no longer available in
# opencv > 3.4.2.16
import cv2

# Loading the image
img = cv2.imread('8.jpg')

# Converting image to grayscale
gray= cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

# Applying SIFT detector
sift = cv2.xfeatures2d.SIFT_create()
kp = sift.detect(gray, None)

# Marking the keypoint on the image using circles
img=cv2.drawKeypoints(gray ,
                      kp ,
                      img ,
                      flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

cv2.imwrite('image-with-keypoints.jpg', img)
```



Feature Matching

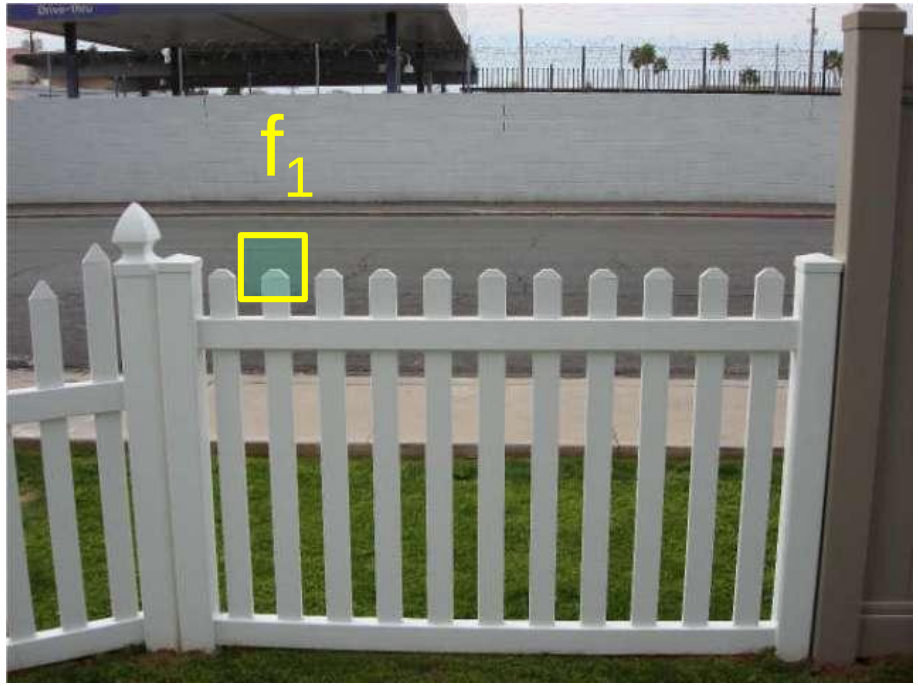
Given a feature in I_1, how to find the best match in I_2?

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

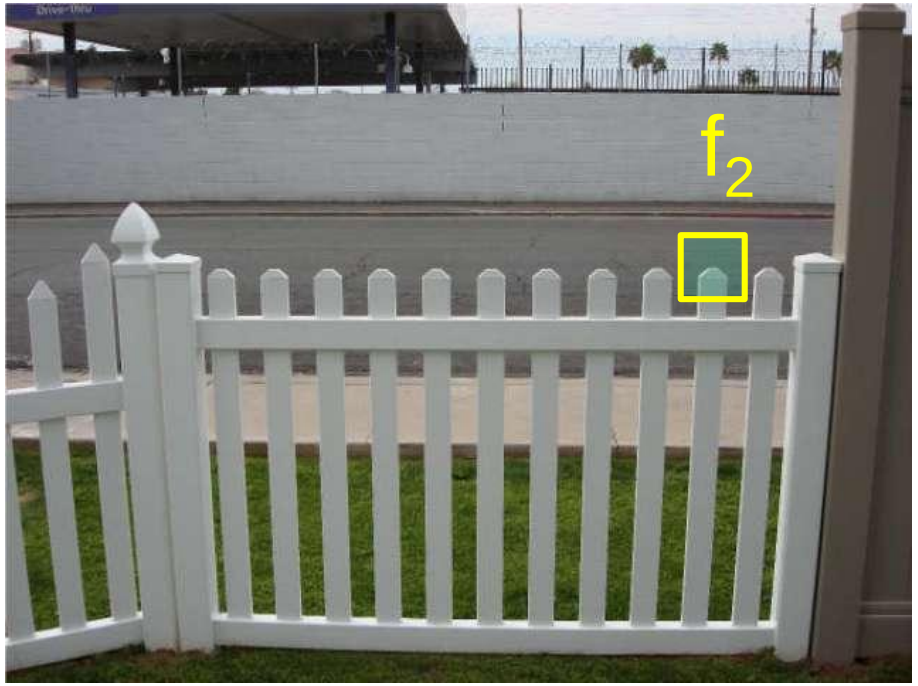
Feature Distance

How to define the difference between two features f_1, f_2?

- Simple approach: L_2 distance, $||f_1 - f_2||$
- can give good scores to ambiguous (incorrect) matches



I_1

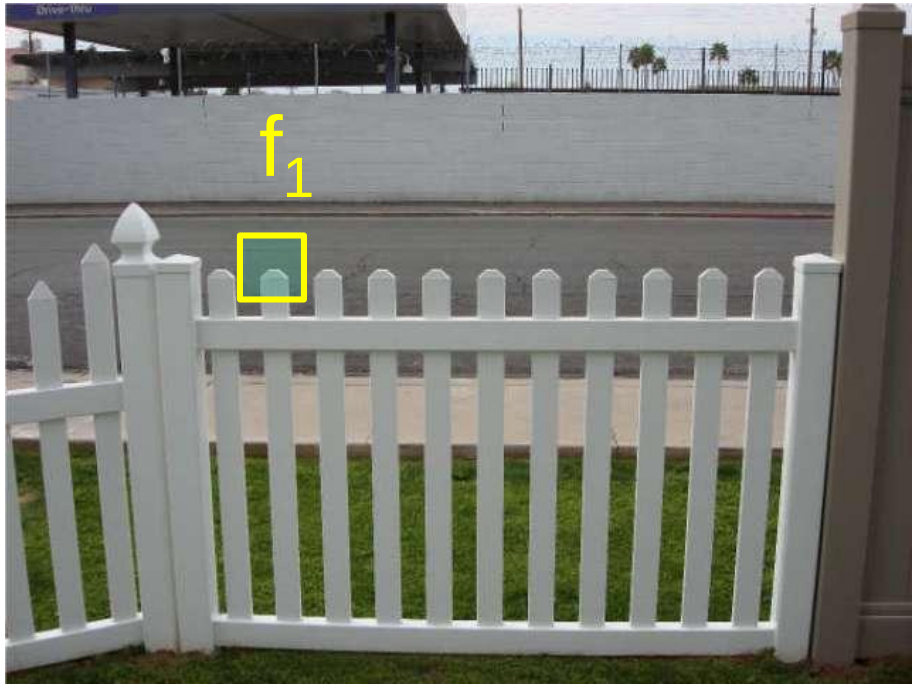


I_2

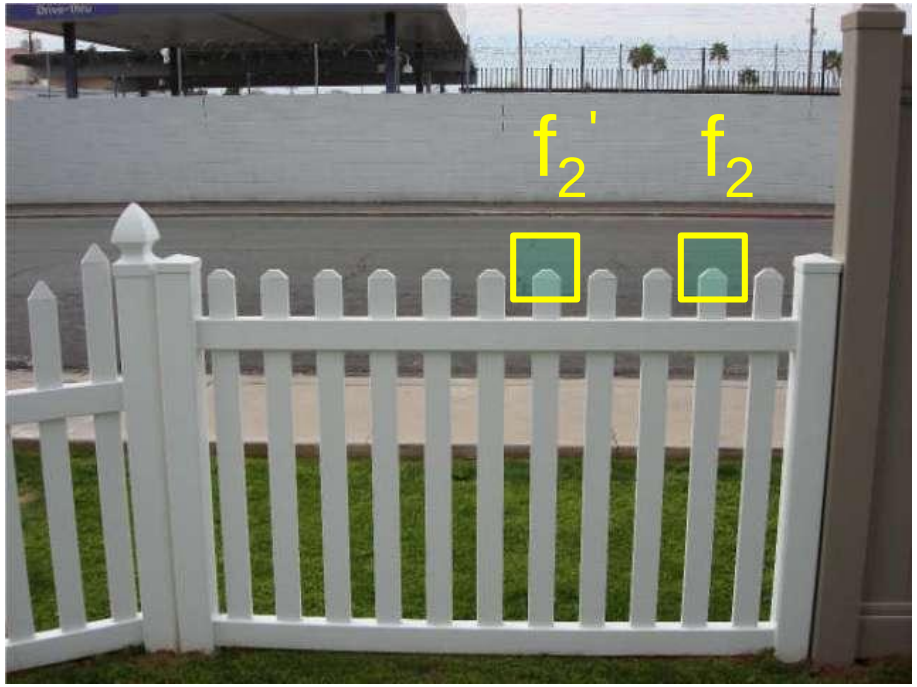
Feature Distance

How to define the difference between two features f_1, f_2?

- Better approach: ratio distance = $||f_1 - f_2|| / ||f_1 - f_2'||$
  - f_2 is best SSD match to f_1 in I_2
  - f_2' is 2nd best SSD match to f_1 in I_2
  - gives large values for ambiguous matches

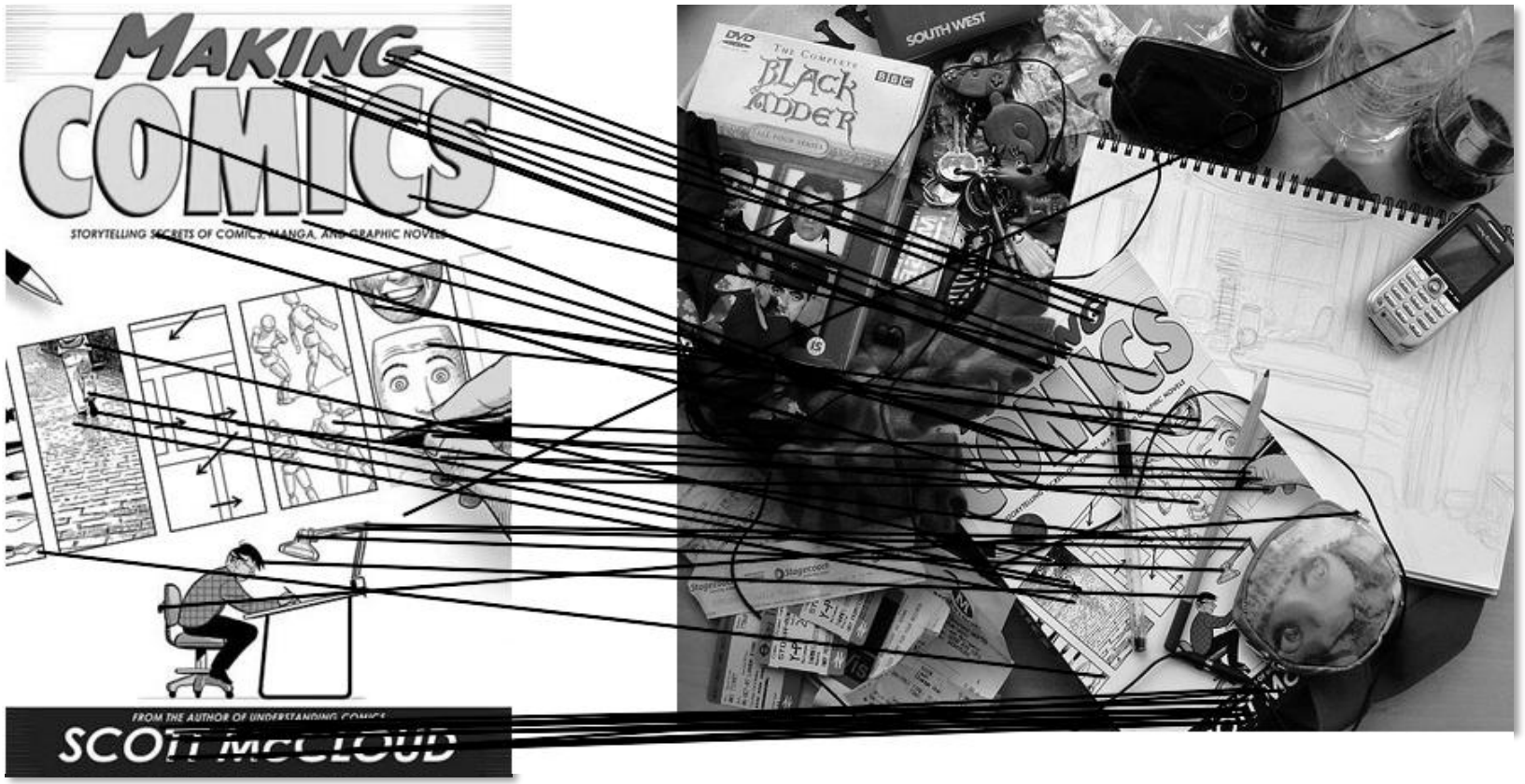


I_1



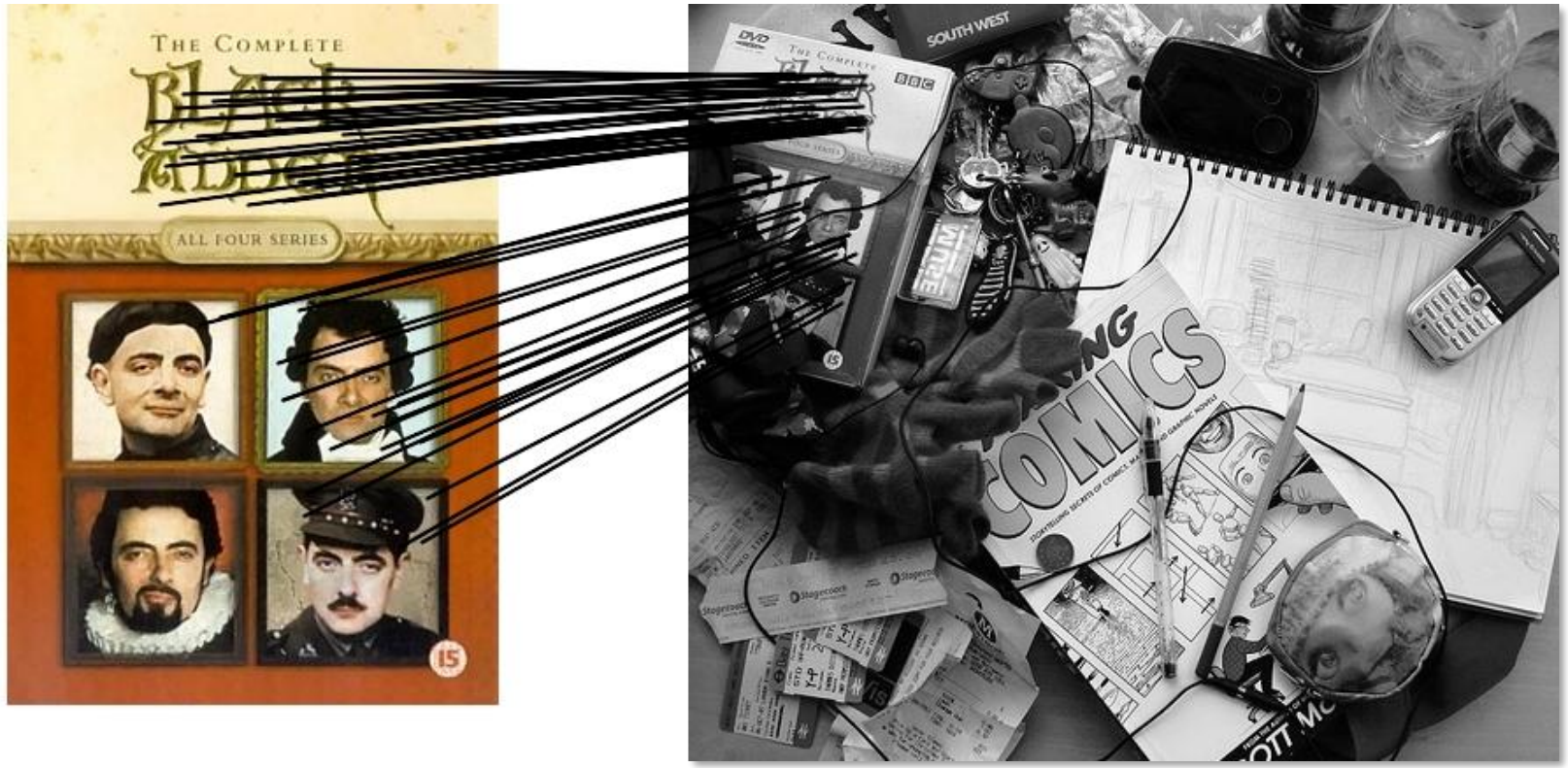
I_2

Feature Matching Example



51 matches

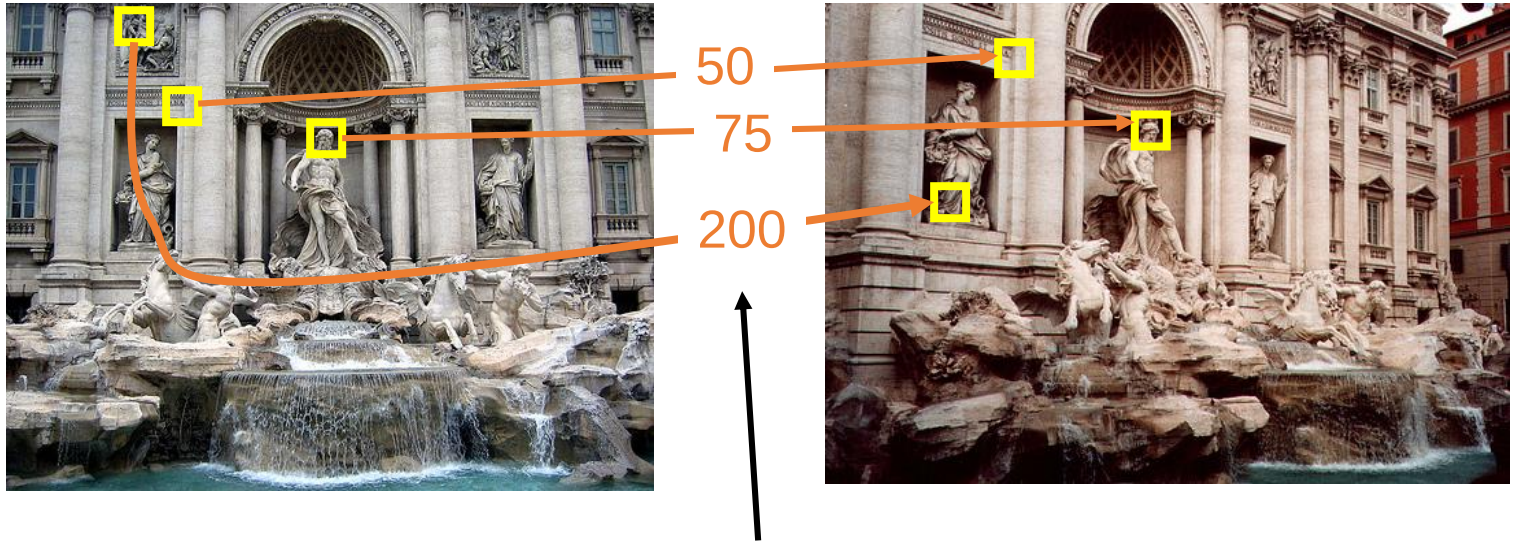
Feature Matching Example



58 matches

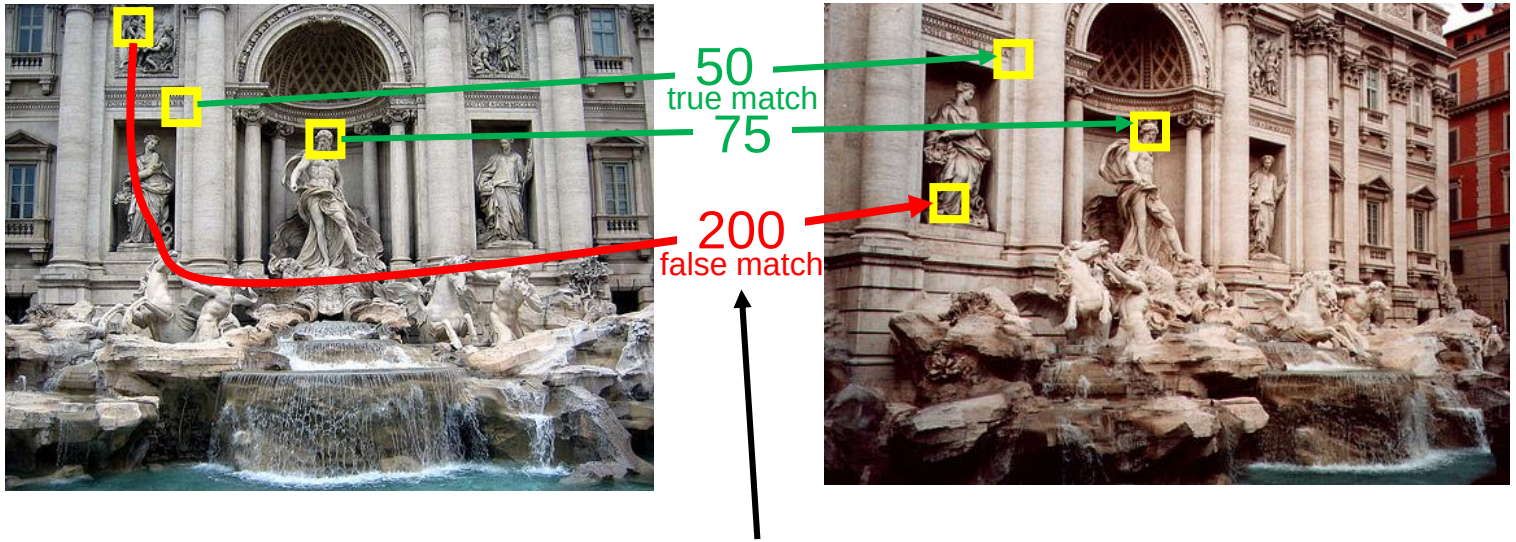
Evaluating the results

How can we measure the performance of a feature matcher?



True/False Positives

How can we measure the performance of a feature matcher?



The distance threshold affects performance

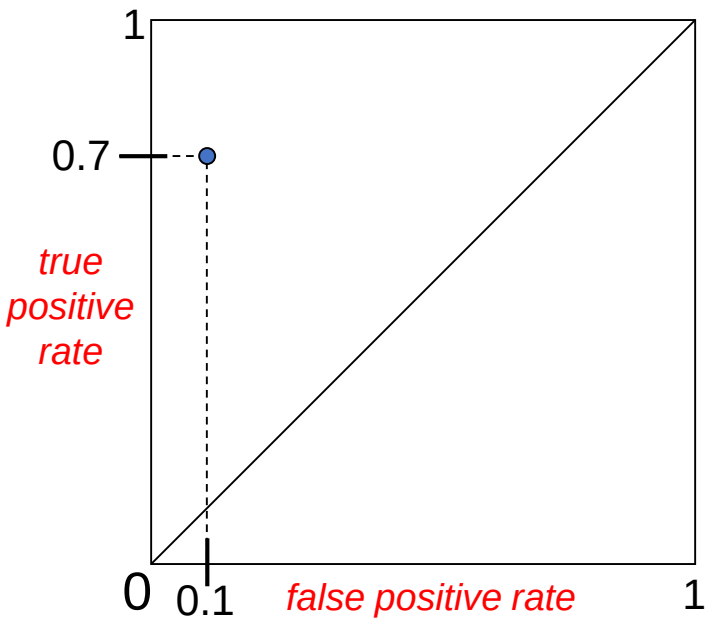
- True positives = # of detected matches that are correct
  - Suppose we want to maximize these—how to choose threshold?
- False positives = # of detected matches that are incorrect
  - Suppose we want to minimize these—how to choose threshold?

Evaluating the results

How can we measure the performance of a feature matcher?

$$\frac{\text{\# true positives}}{\text{\# correctly matched features (positives)}}$$

"recall"



$$\frac{\text{\# false positives}}{\text{\# incorrectly matched features (negatives)}}$$

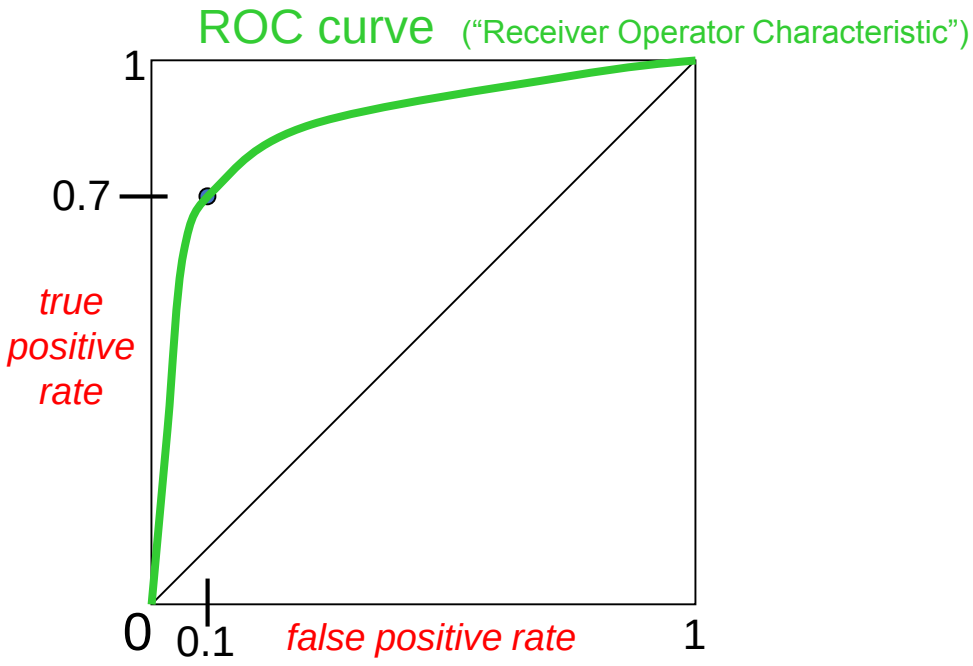
1 - "precision"

Evaluating the results

How can we measure the performance of a feature matcher?

$$\frac{\text{\# true positives}}{\text{\# correctly matched features (positives)}}$$

"recall"

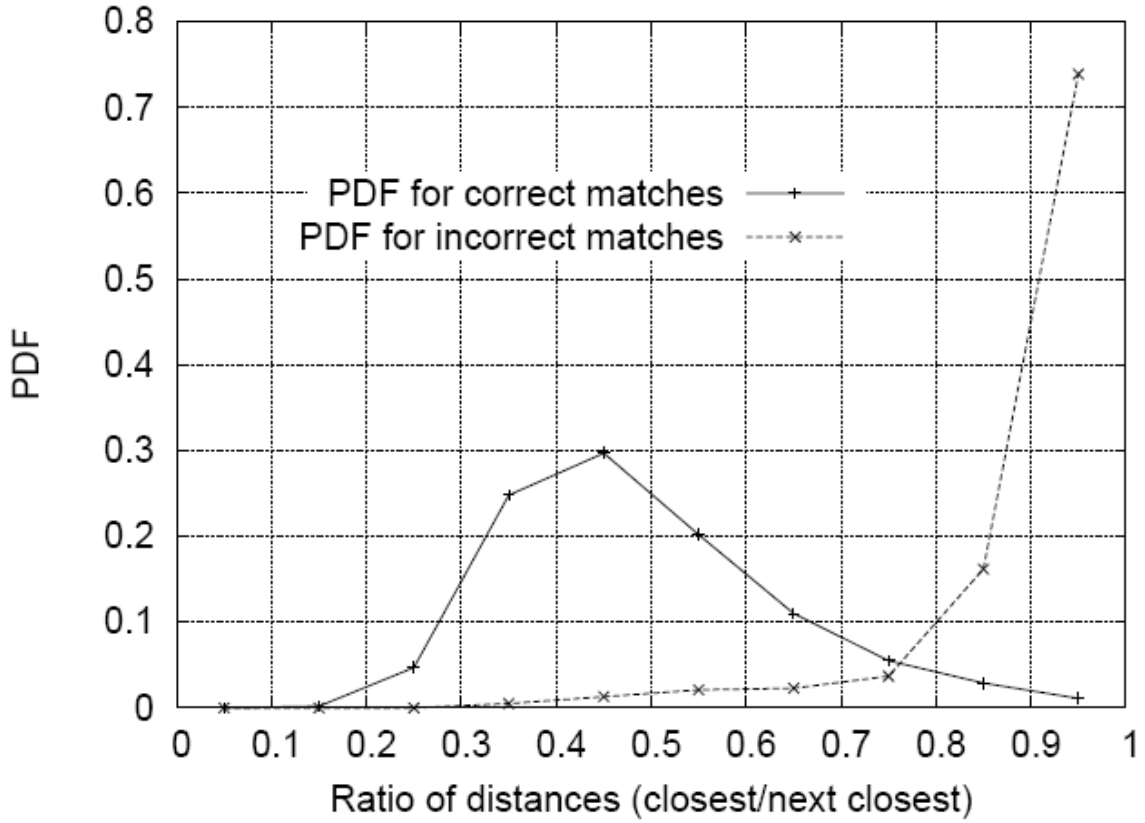


$$\frac{\text{\# false positives}}{\text{\# incorrectly matched features (negatives)}}$$

1 - "precision"

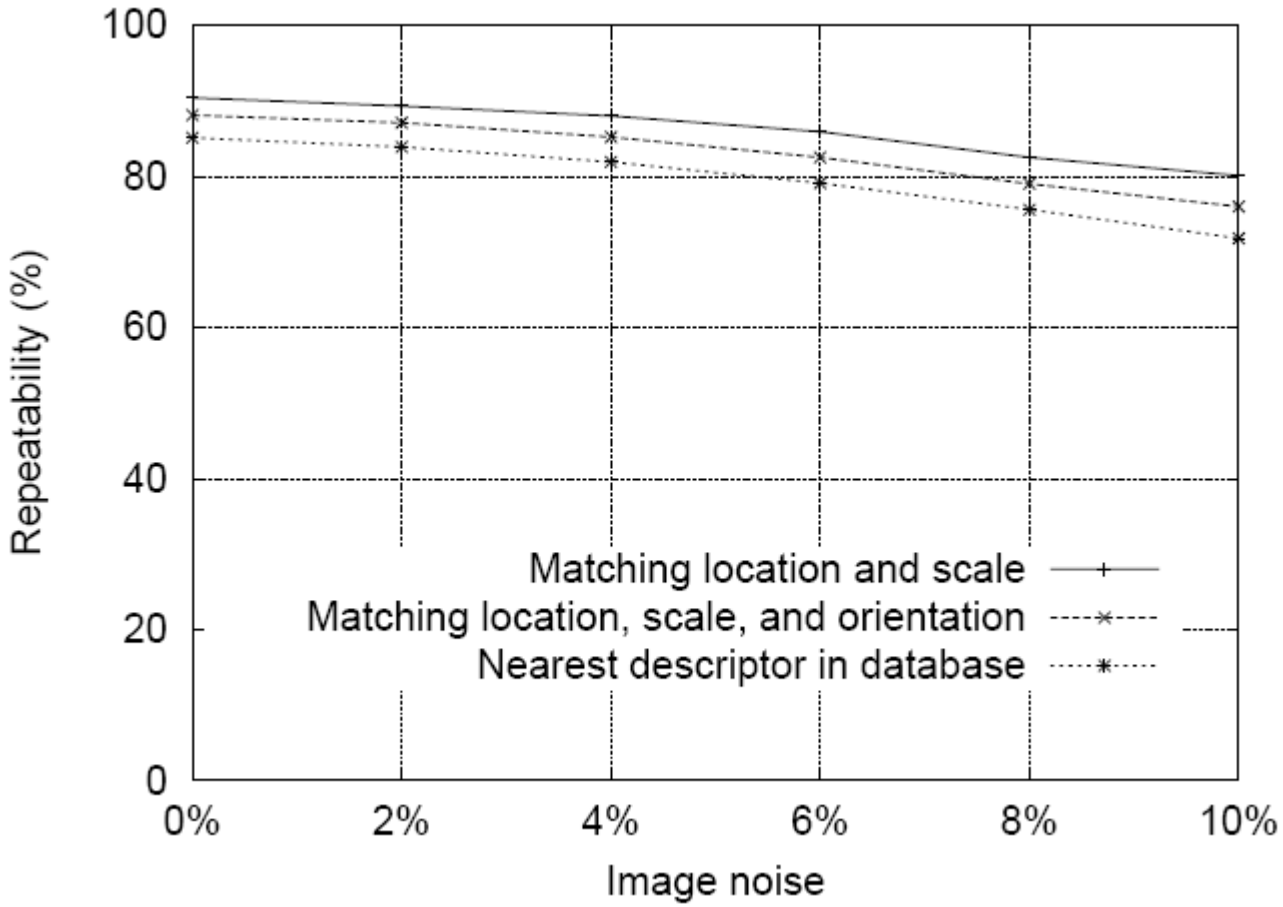
Matching SIFT Descriptors

- Nearest neighbor (Euclidean distance)
- Threshold ratio of nearest to 2nd nearest descriptor



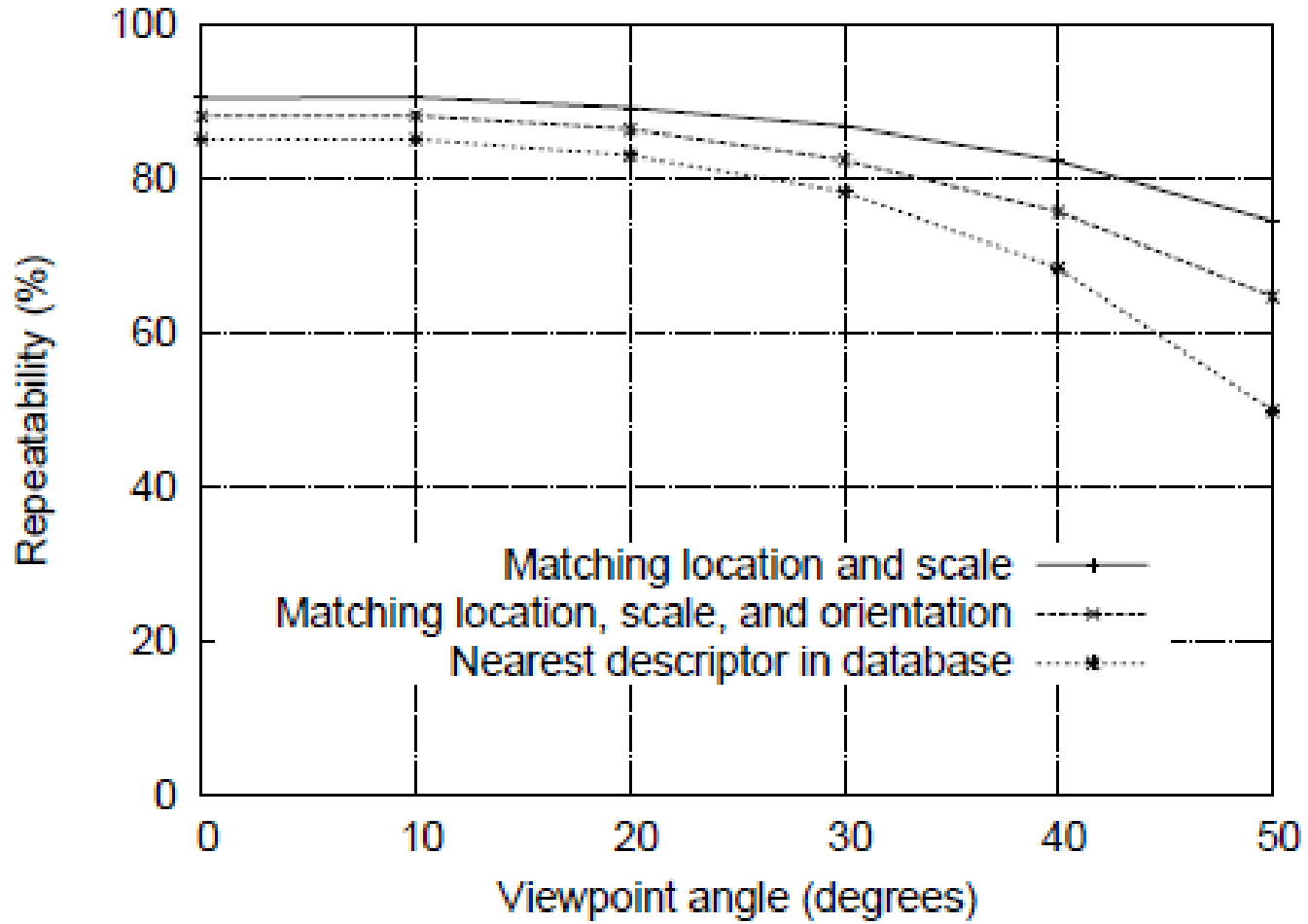
Lowe IJCV 2004

SIFT Repeatability

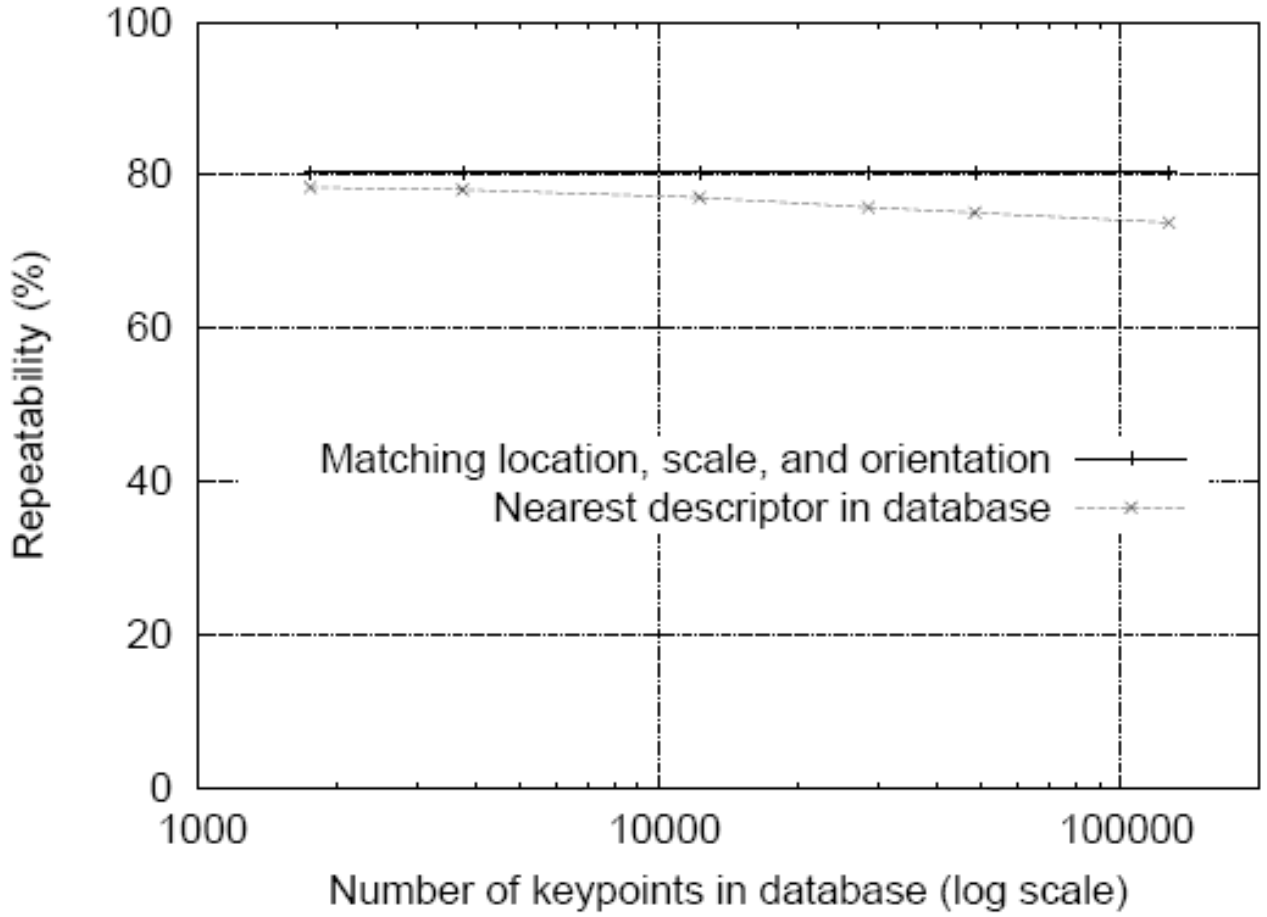


Lowe IJCV 2004

SIFT Repeatability



SIFT Repeatability

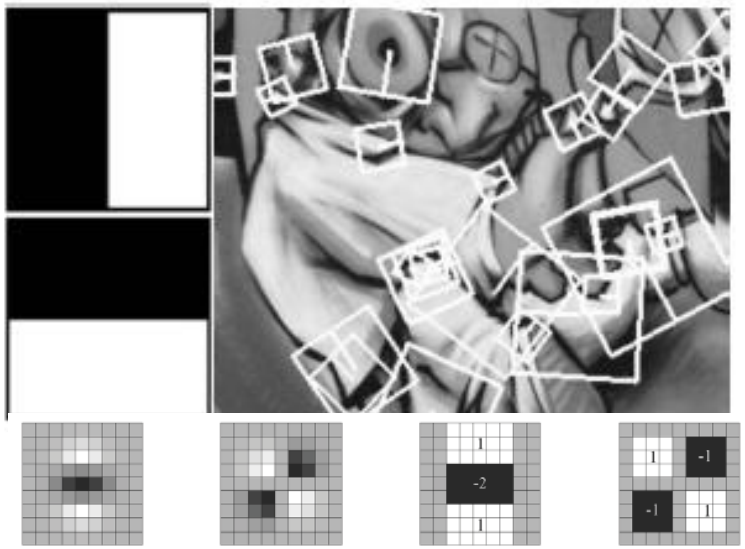


Lowe IJCV 2004

SIFT Feature Matching Python

```python
# import required libraries
import numpy as np
import cv2
import matplotlib.pyplot as plt

# read two input images as grayscale
img1 = cv2.imread('0.jpg',0)
img2 = cv2.imread('65.jpg',0)

# Initiate SIFT detector
sift = cv2.SIFT_create()

# detect and compute the keypoints and descriptors with SIFT
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)

# create BFMatcher object
bf = cv2.BFMatcher()

# Match descriptors.
matches = bf.match(des1,des2)

# sort the matches based on distance
matches = sorted(matches, key=lambda val: val.distance)

# Draw first 50 matches.
out = cv2.drawMatches(img1, kp1, img2, kp2, matches[:50], None, flags=2)
plt.imshow(out), plt.show()
```

Local Descriptors: SURF



Fast approximation of SIFT idea

Efficient computation by 2D box filters & integral images

$\Rightarrow$ 6 times faster than SIFT

Equivalent quality for object identification

GPU implementation available

Feature extraction @ 200Hz (detector + descriptor, 640×480 img)

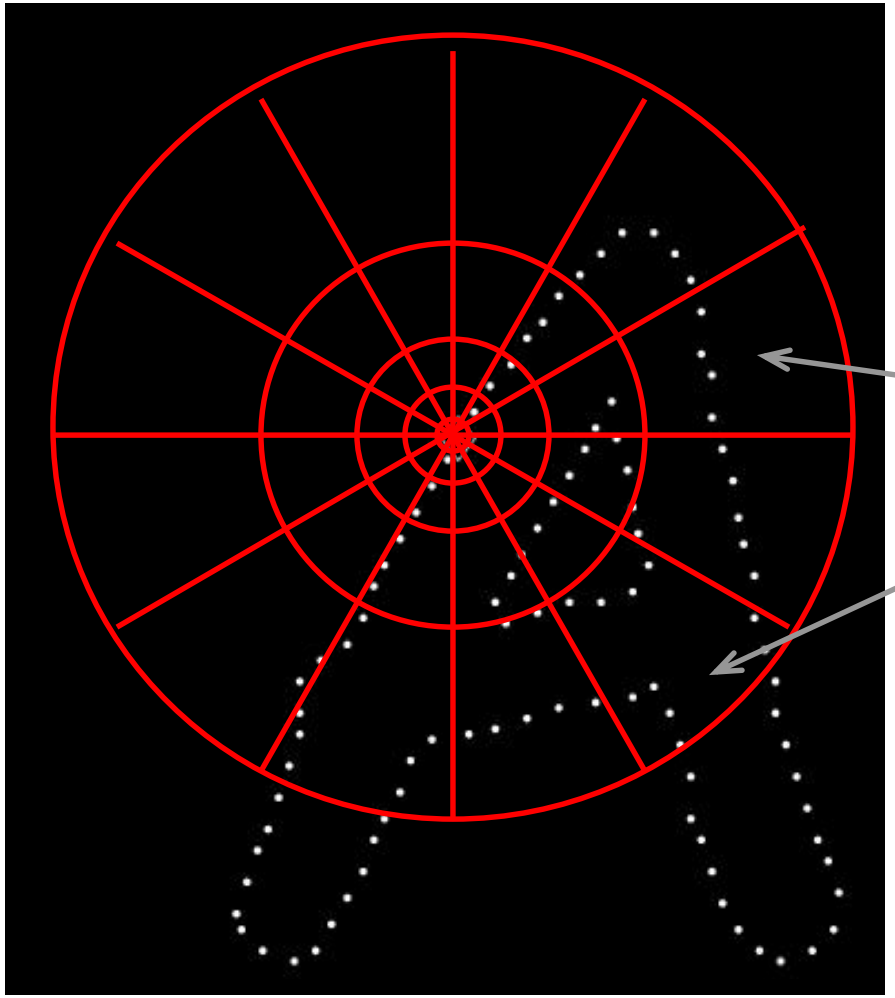
http://www.vision.ee.ethz.ch/~surf

Many other efficient descriptors are also available

[Bay, ECCV'06], [Cornelis, CVGPU'08]

K. Grauman, B. Leibe

Local Descriptors: Shape Context



Count the number of points inside each bin, e.g.:

Count = 4

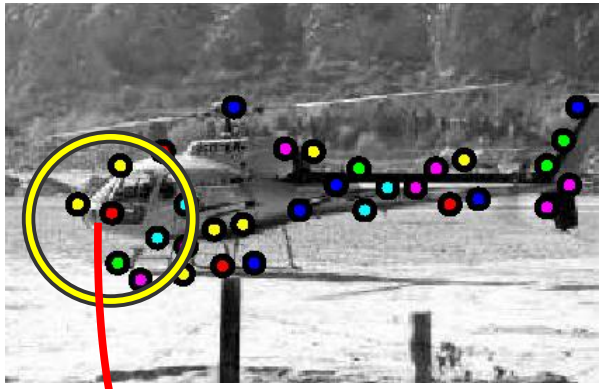
⋮

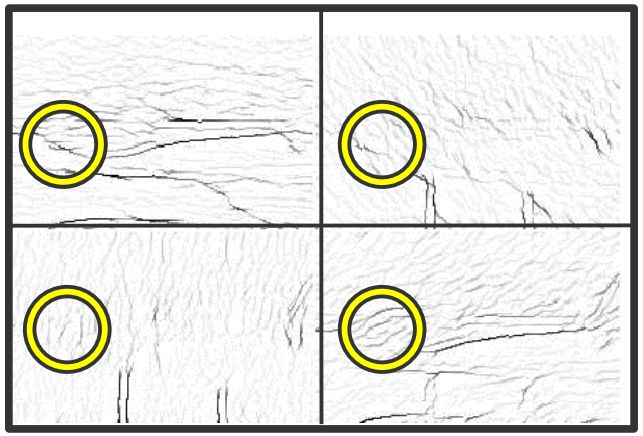
Count = 10

Log-polar binning: more precision for nearby points, more flexibility for farther points.

Belongie & Malik, ICCV 2001

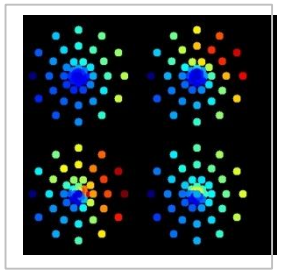
Local Descriptors: Geometric Blur



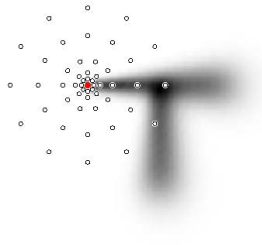


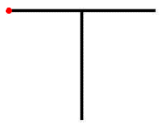
Compute edges at four orientations

Extract a patch in each channel



Example descriptor





Apply spatially varying blur and sub-sample

(Idealized signal)

Berg & Malik, CVPR 2001

Choosing a Detector

- What do you want it for?
  - Precise localization in x-y: Harris
  - Good localization in scale: Difference of Gaussian
  - Flexible region shape: MSER
- Best choice often application dependent
  - Harris-/Hessian-Laplace/DoG work well for many natural categories
  - MSER works well for buildings and printed things
- Why choose?
  - Get more points with more detectors
- There have been extensive evaluations/comparisons
  - [Mikolajczyk et al., IJCV'05, PAMI'05]
  - All detectors/descriptors shown here work well

Comparison of Key Point Detectors

Table 7.1 Overview of feature detectors.

Feature Detector	Corner	Blob	Region	Rotation invariant	Scale invariant	Affine invariant	Repeatability	Localization accuracy	Robustness	Efficiency
Harris	√			√			+++	+++	+++	++
Hessian		√		√			++	++	++	+
SUSAN	√			√			++	++	++	+++
Harris-Laplace	√	(√)		√	√		+++	+++	++	+
Hessian-Laplace	(√)	√		√	√		+++	+++	+++	+
DoG	(√)	√		√	√		++	++	++	++
SURF	(√)	√		√	√		++	++	++	+++
Harris-Affine	√	(√)		√	√	√	+++	+++	++	++
Hessian-Affine	(√)	√		√	√	√	+++	+++	+++	++
Salient Regions	(√)	√		√	√	(√)	+	+	++	+
Edge-based	√			√	√	√	+++	+++	+	+
MSER			√	√	√	√	+++	+++	++	+++
Intensity-based			√	√	√	√	++	++	++	++
Superpixels			√	√	(√)	(√)	+	+	+	+

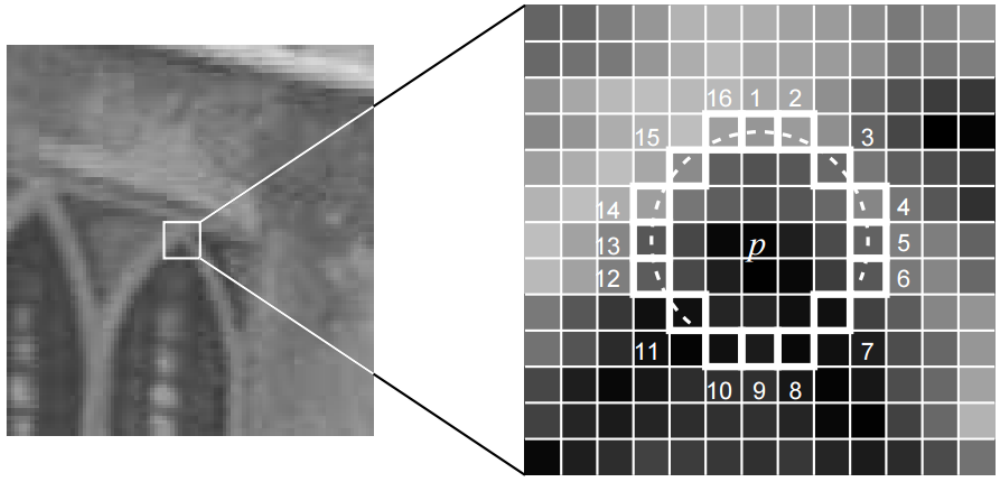
Tuytelaars Mikolajczyk 2008

Choosing a Descriptor

- Again, need not stick to one

- For object instance recognition or stitching, SIFT or variant is a good choice

Recent advances in Interest Points



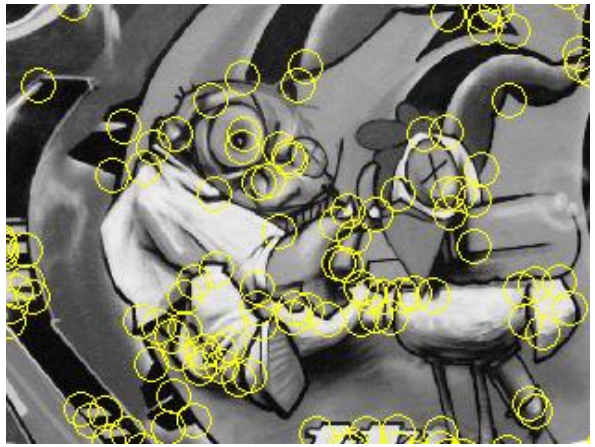
Features from Accelerated Segment Test, ECCV 06

Binary feature descriptors

- BRIEF: Binary Robust Independent Elementary Features, ECCV 10
- ORB (Oriented FAST and Rotated BRIEF), CVPR 11
- BRISK: Binary robust invariant scalable keypoints, ICCV 11
- Freak: Fast retina keypoint, CVPR 12
- LIFT: Learned Invariant Feature Transform, ECCV 16

Things to Remember

- Keypoint detection: repeatable and distinctive
  - Corners, blobs, stable regions
  - Harris, DoG



- Descriptors: robust and selective
  - spatial histograms of orientation
  - SIFT