中山大學
SUN YAT-SEN UNIVERSITY

Lecture 5.
Feature Descriptors

Pattern Recognition and Computer Vision

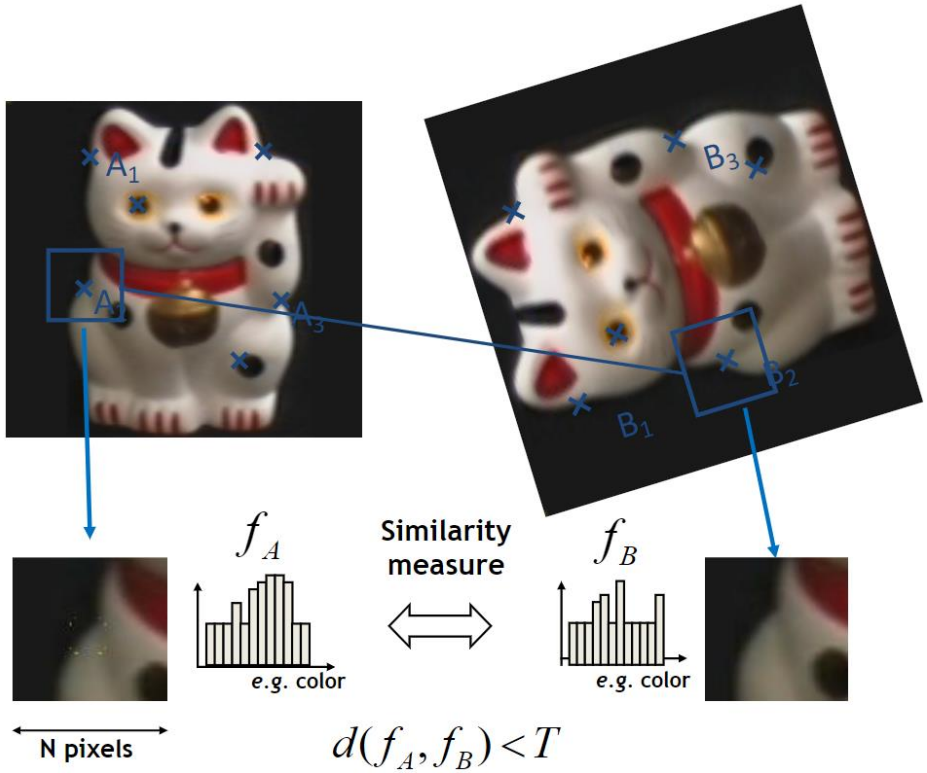
Guanbin Li,
School of Computer Science and Engineering, Sun Yat-Sen University

扫码签到

What will we learn today?

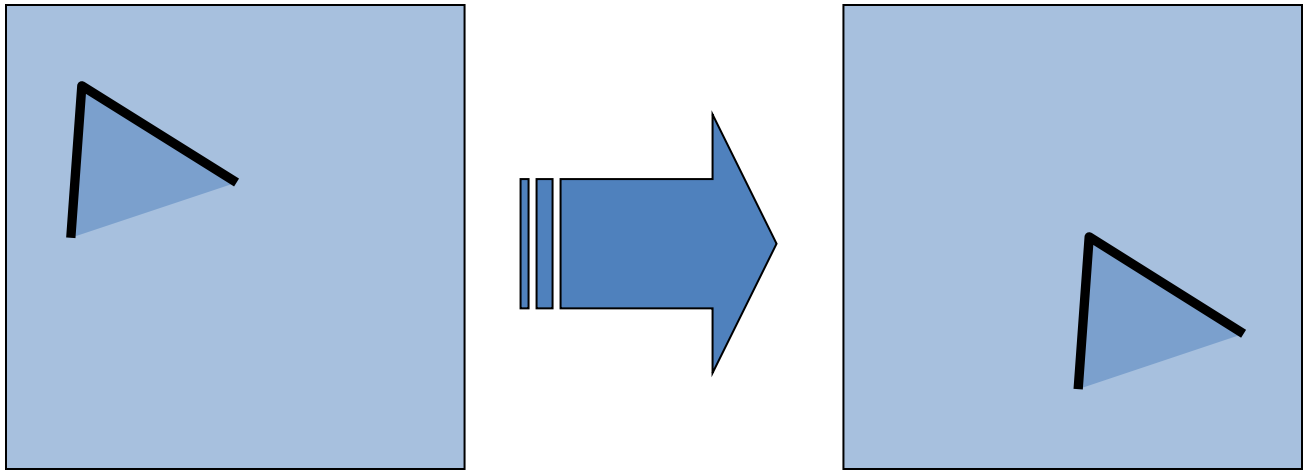
- A quick review
- Scale invariant region detection
  - Automatic scale selection
  - Difference-of-Gaussian (DoG) detector
- SIFT: an image region descriptor
- Application: Panorama

A quick review : General Approach



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

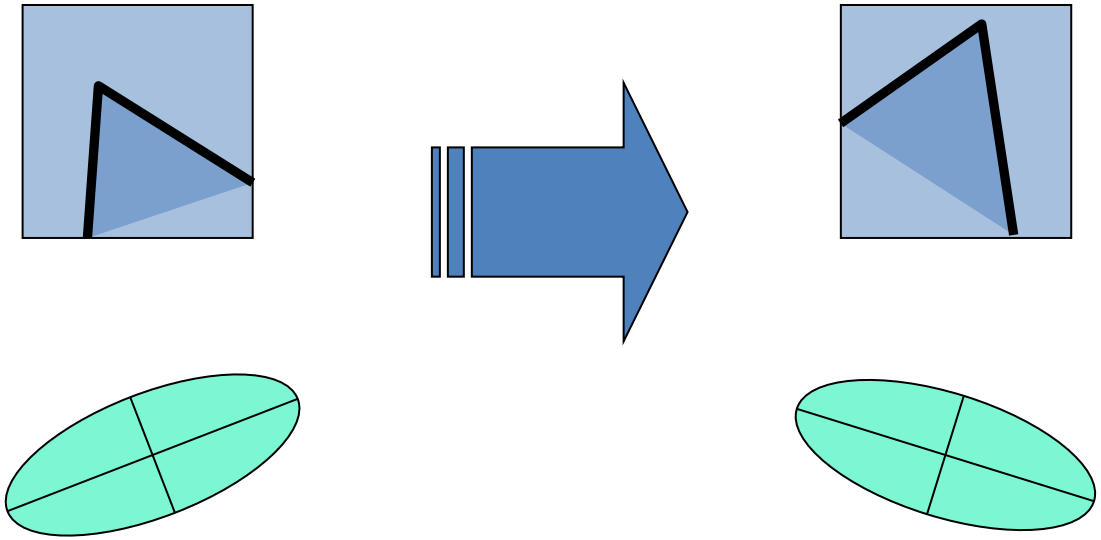
Quick review: Harris Corner Detector



- Derivatives and window function are equivariant

Corner location is equivariant w.r.t. translation

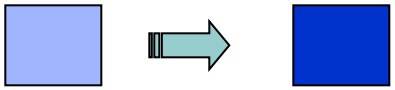
Quick review: Harris Corner Detector



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

Corner location is equivariant w.r.t. image rotation

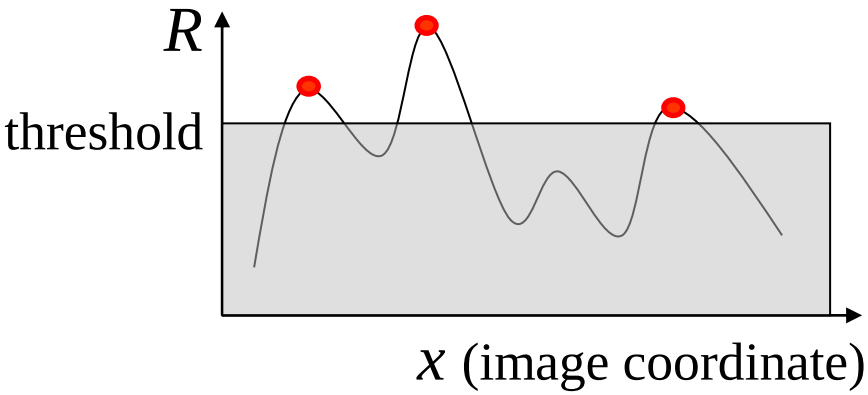
Quick review: Harris Corner Detector

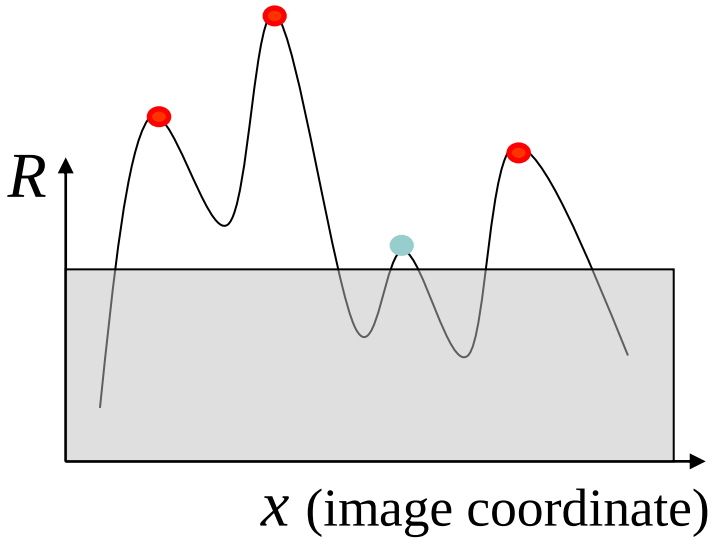


$$I \rightarrow a\,I + b$$

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

- Intensity scaling: $I \rightarrow a\,I$

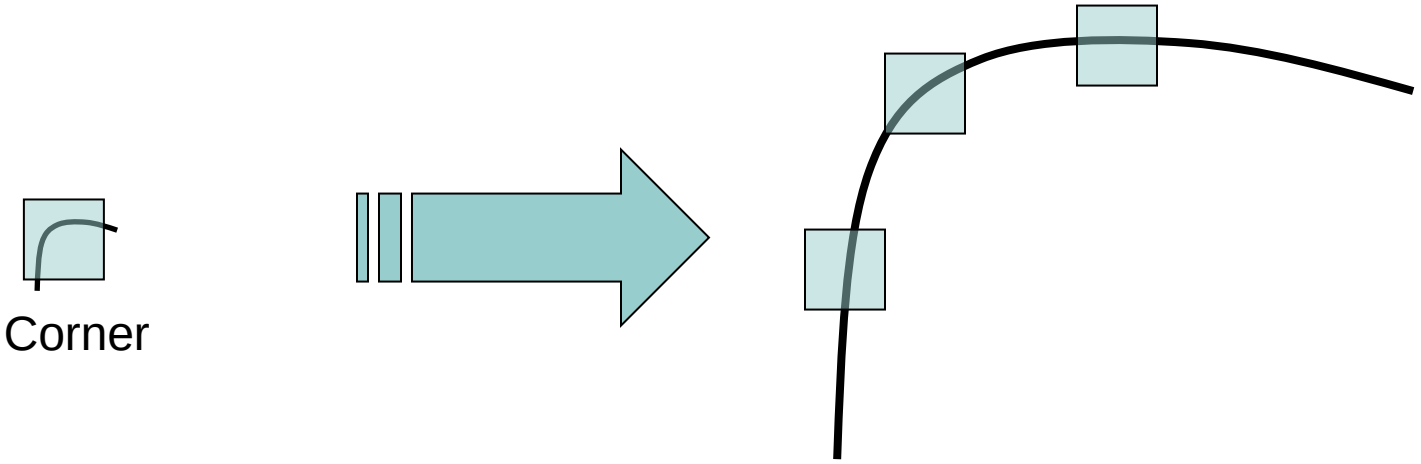




Partially invariant to affine intensity change

Quick review: Harris Corner Detector

- Scaling



All points will be classified as edges

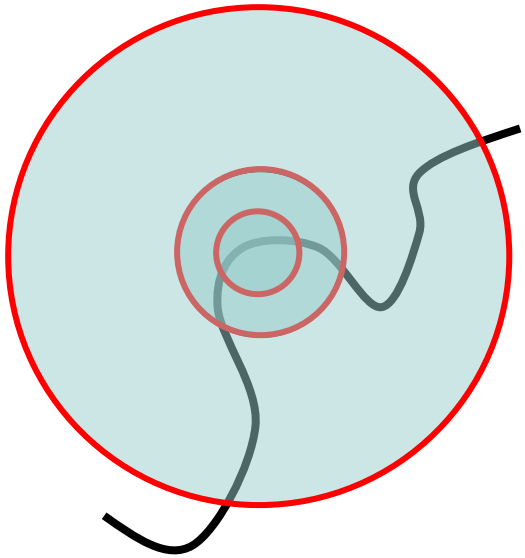
Neither invariant nor equivariant to scaling

What we will learn today?

- A quick review
- Scale invariant region detection
  - Automatic scale selection
  - Difference-of-Gaussian (DoG) detector
- SIFT: an image region descriptor
- HoG: another image region descriptor
- Application: Panorama

Scale invariant detection

Suppose you're looking for corners

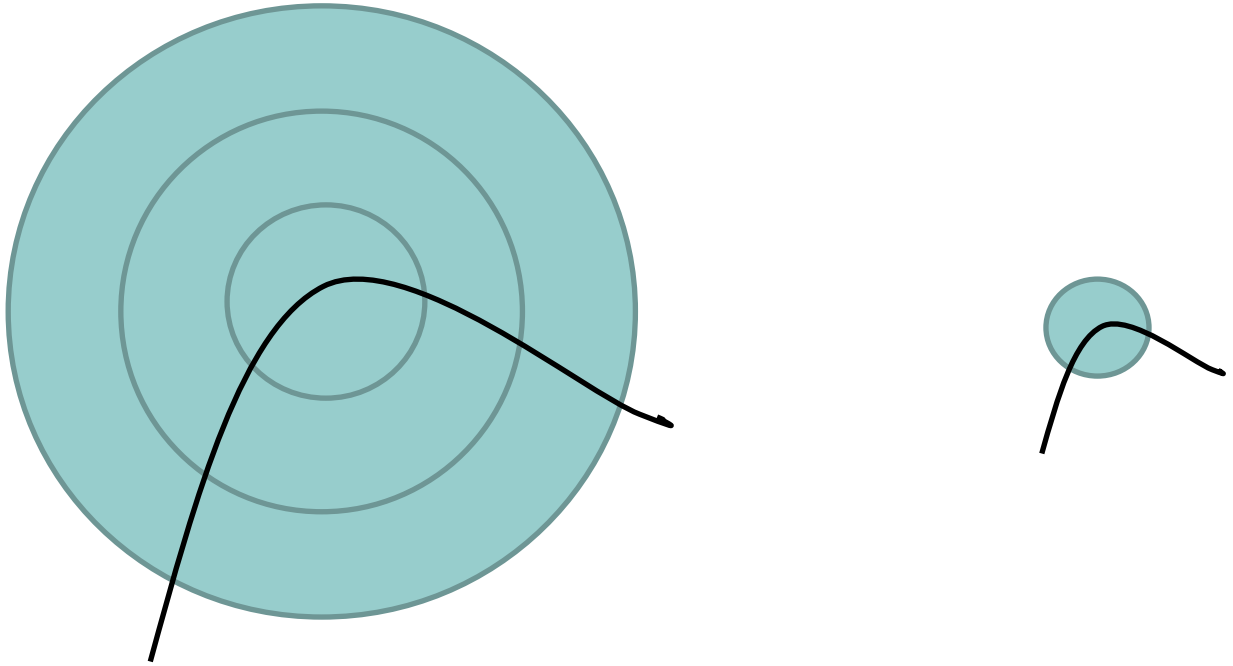


Key idea: find scale that gives local maximum of *f*

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

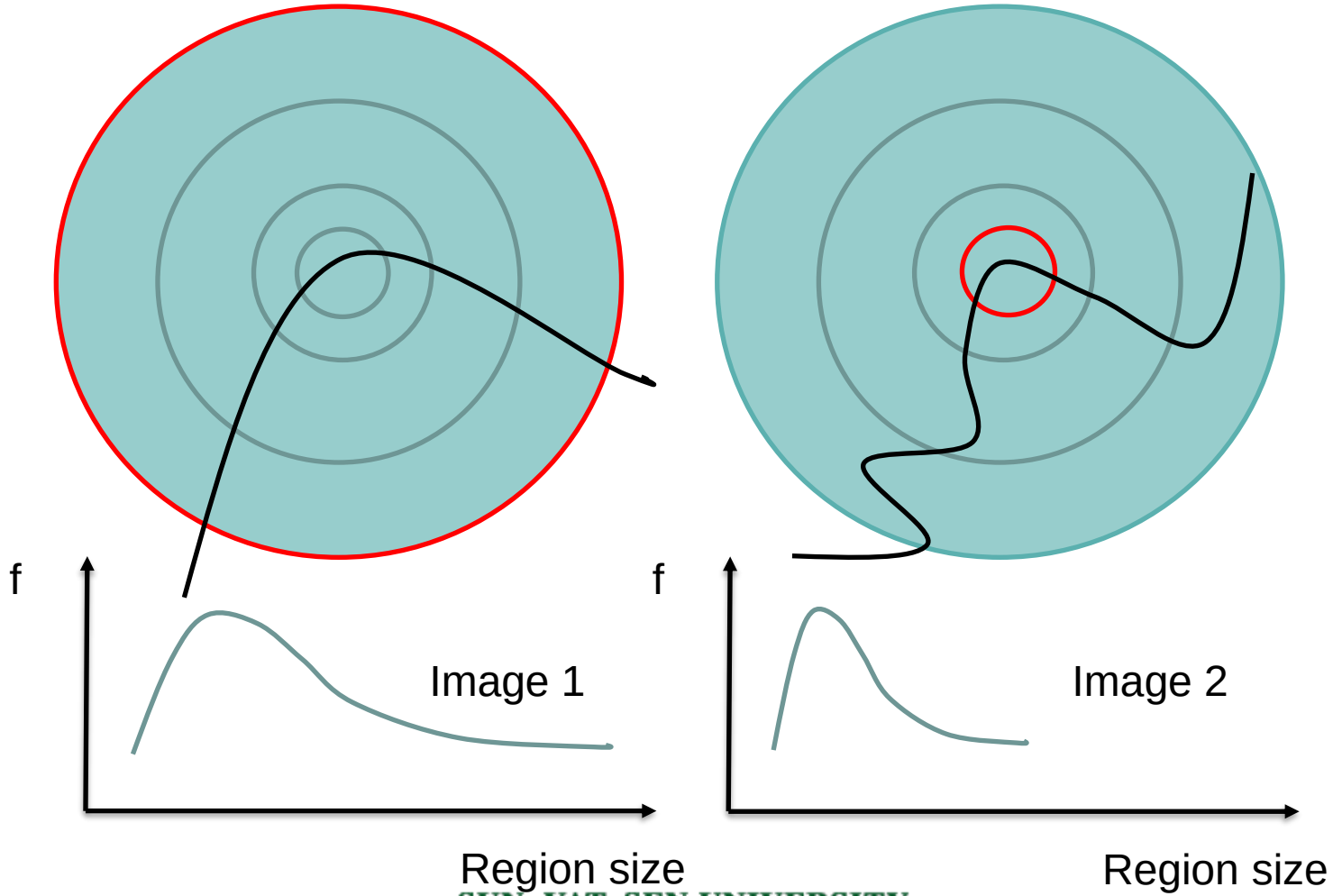
Scale invariant detection

- Scale Invariant Detection
  - Consider regions (e.g. circles) of different sizes around a point
  - What region size do we choose, so that the regions look the same in both images?



Scale invariant detection

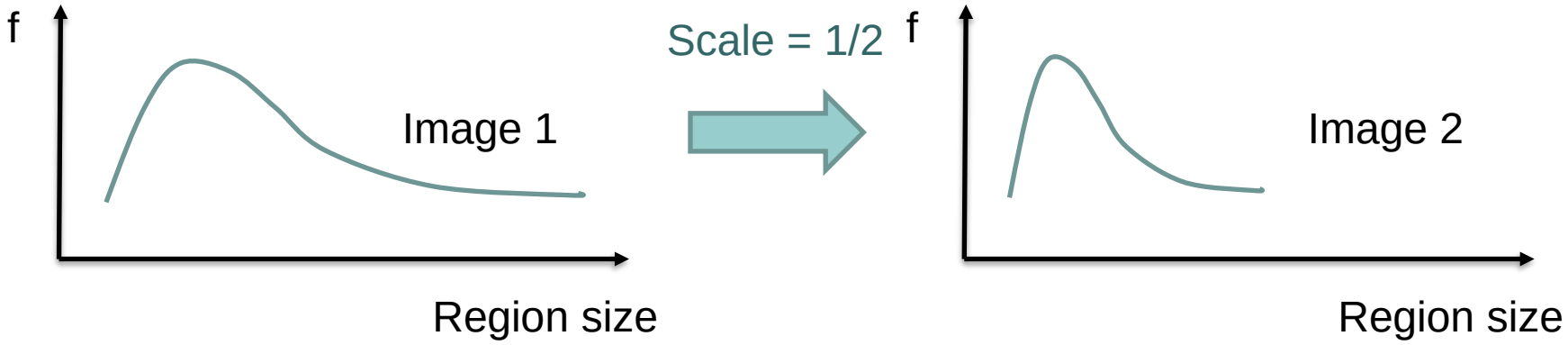
- The problem: how do we choose corresponding circles independently in each image?



Scale invariant detection

- Scale Invariant Detection Solution:
  - Design a function on the region (circle), which is “scale invariant” (the same for corresponding regions, even if they are at different scales)

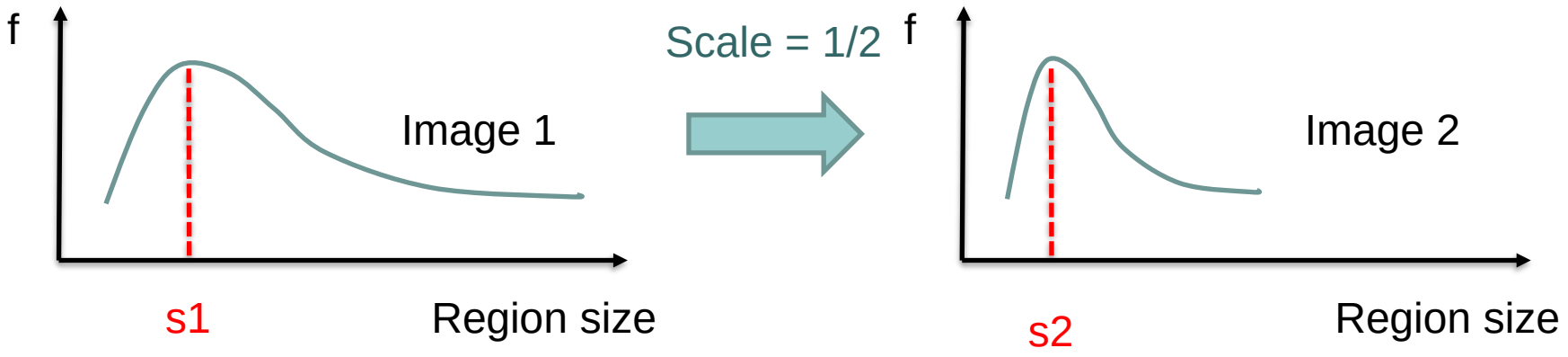
    Example: average intensity. For corresponding regions (even of different sizes) it will be the same.
  - For a point in one image, we can consider it as a function of region size (circle radius)



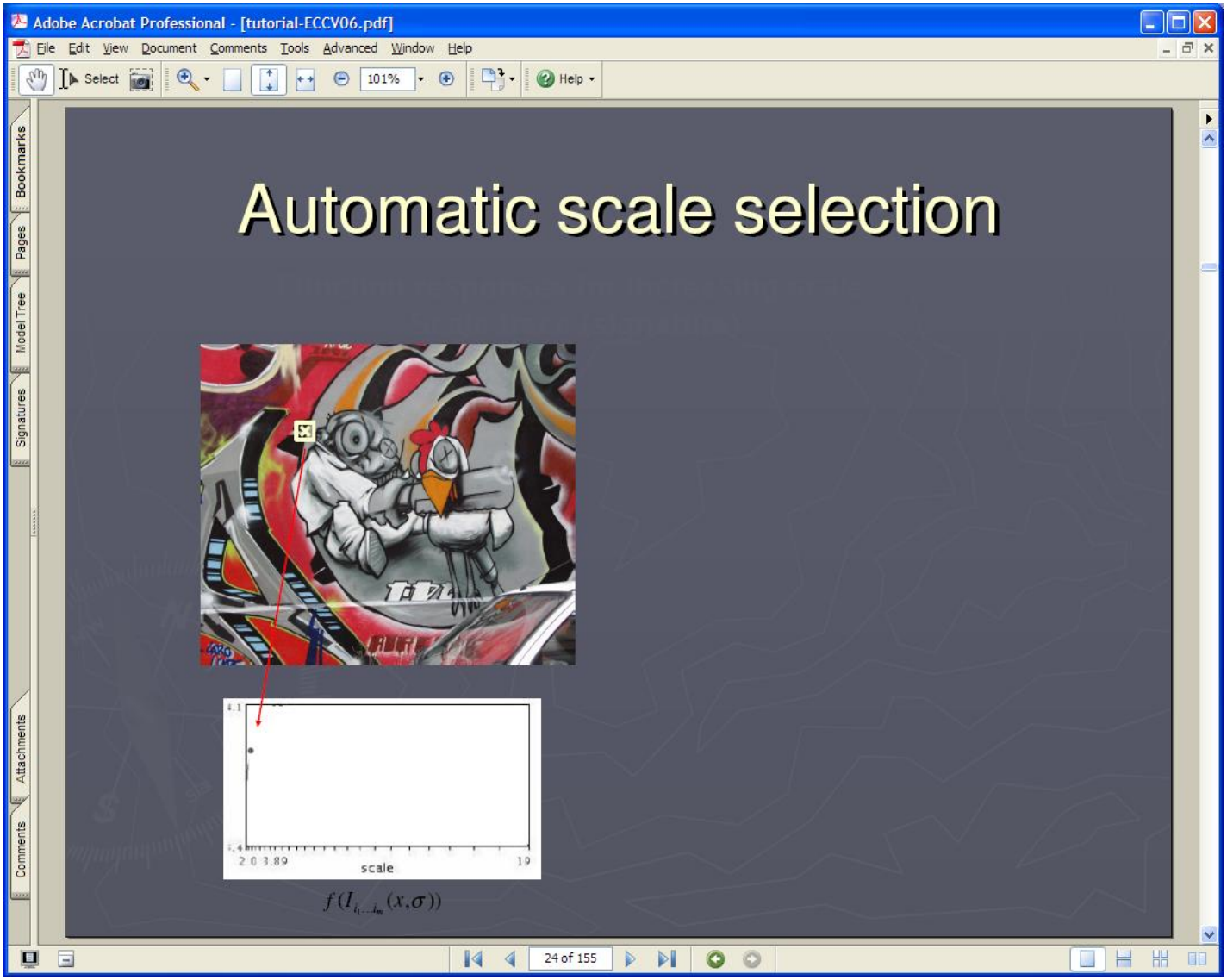
Scale invariant detection

- Common approach:
  - Take a local maximum of this function
- Observation: region size, for which the maximum is achieved, should be co-variant with image scale.

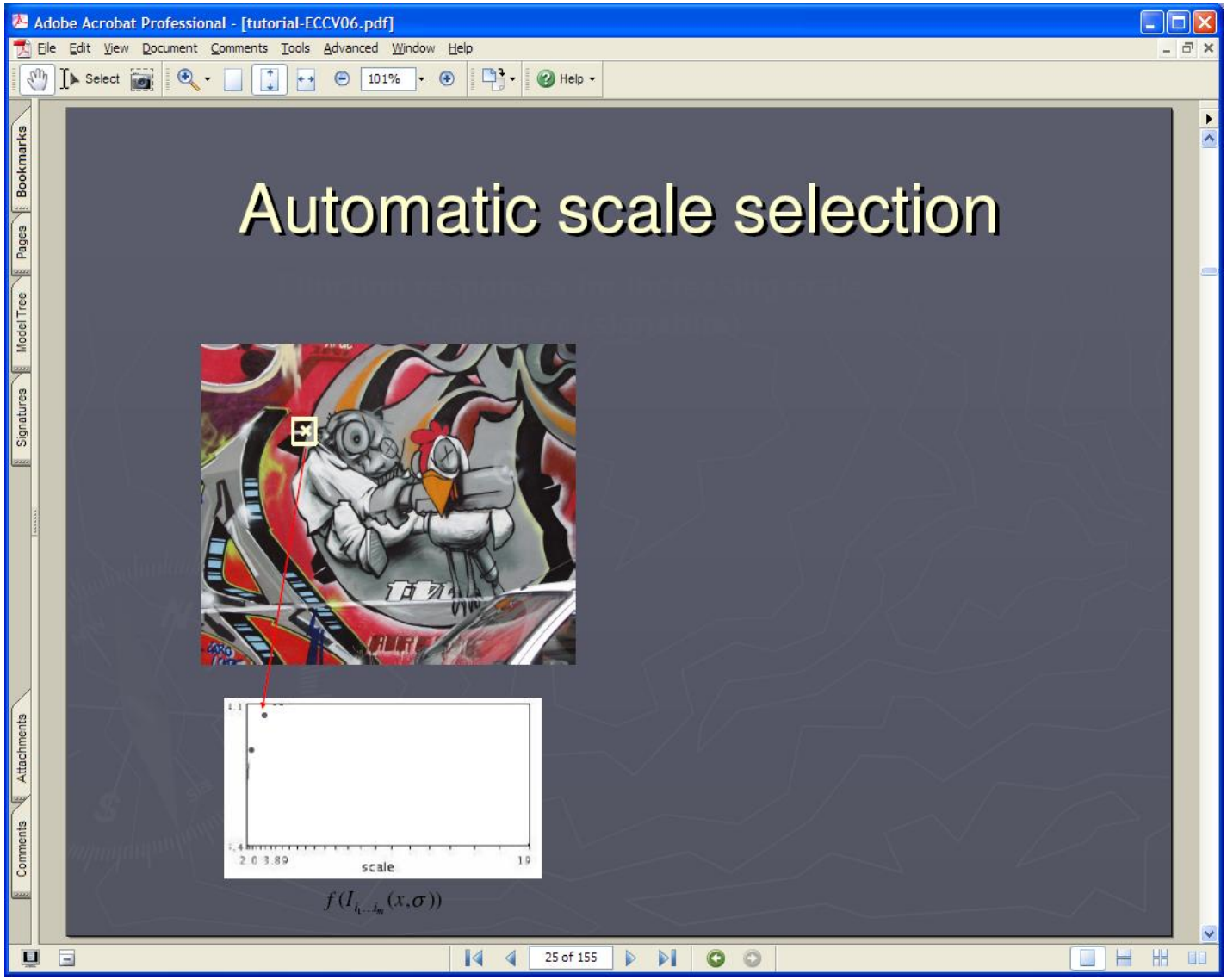
Important: this scale invariant region size is found in each image independently!



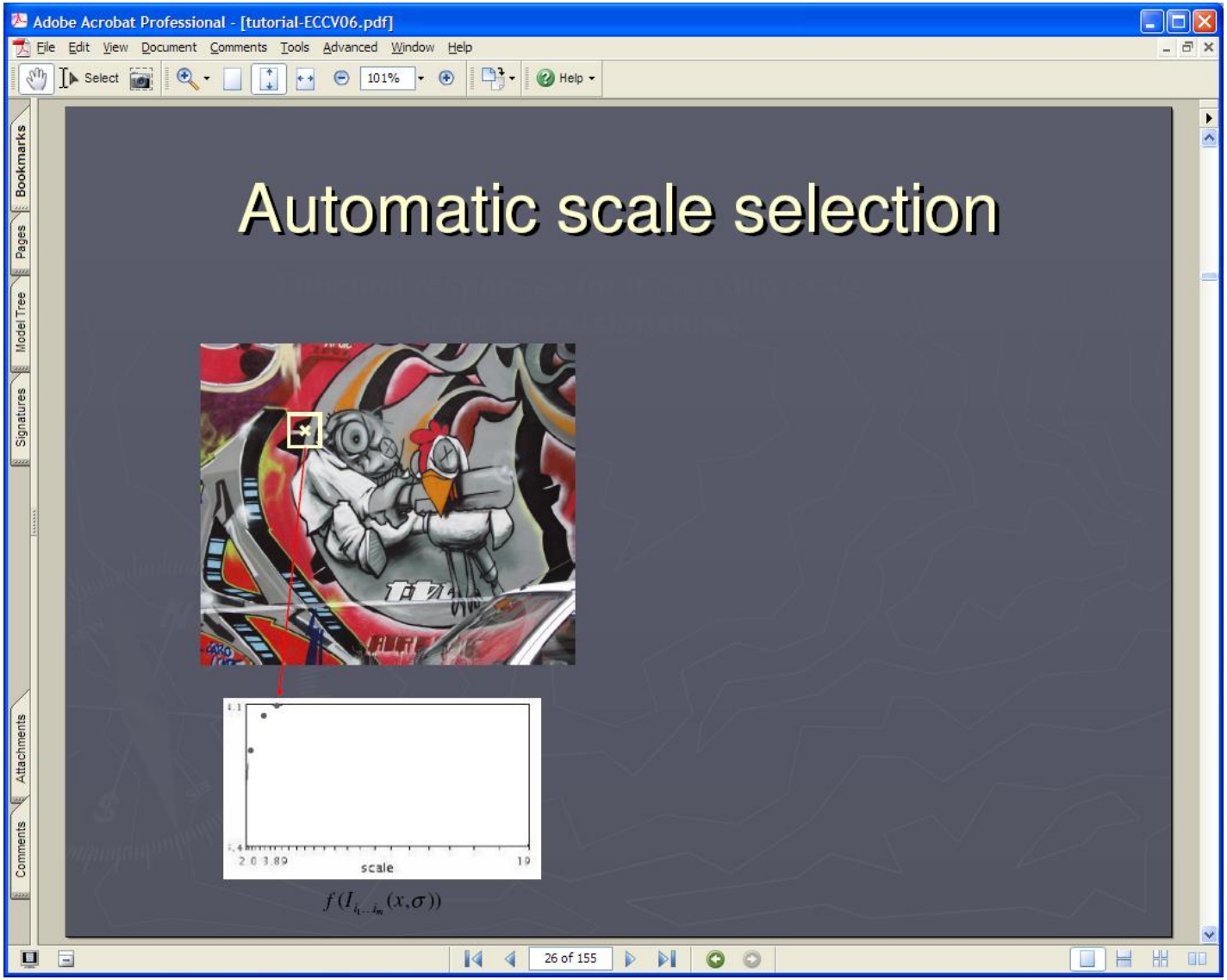
Scale invariant detection



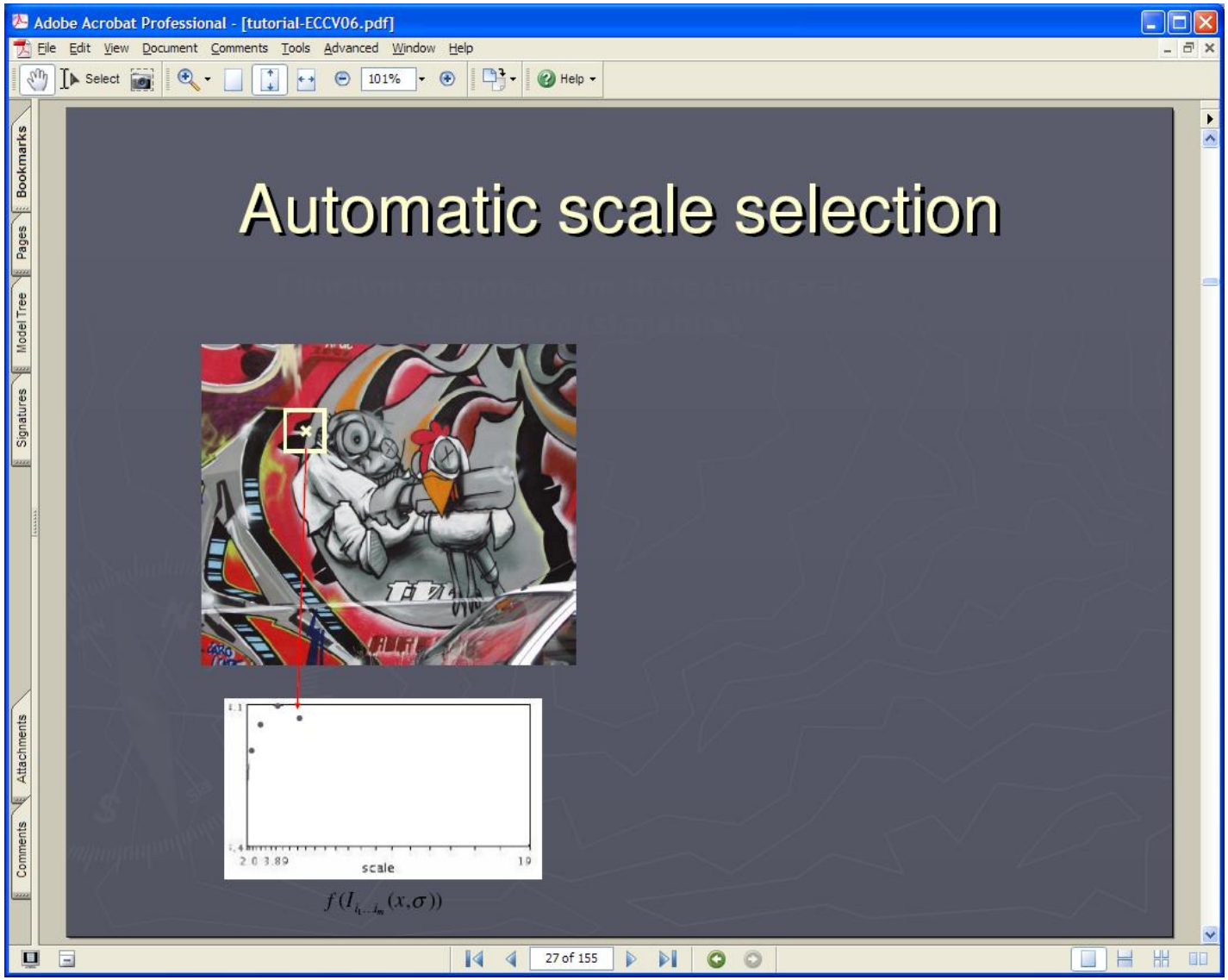
Scale invariant detection



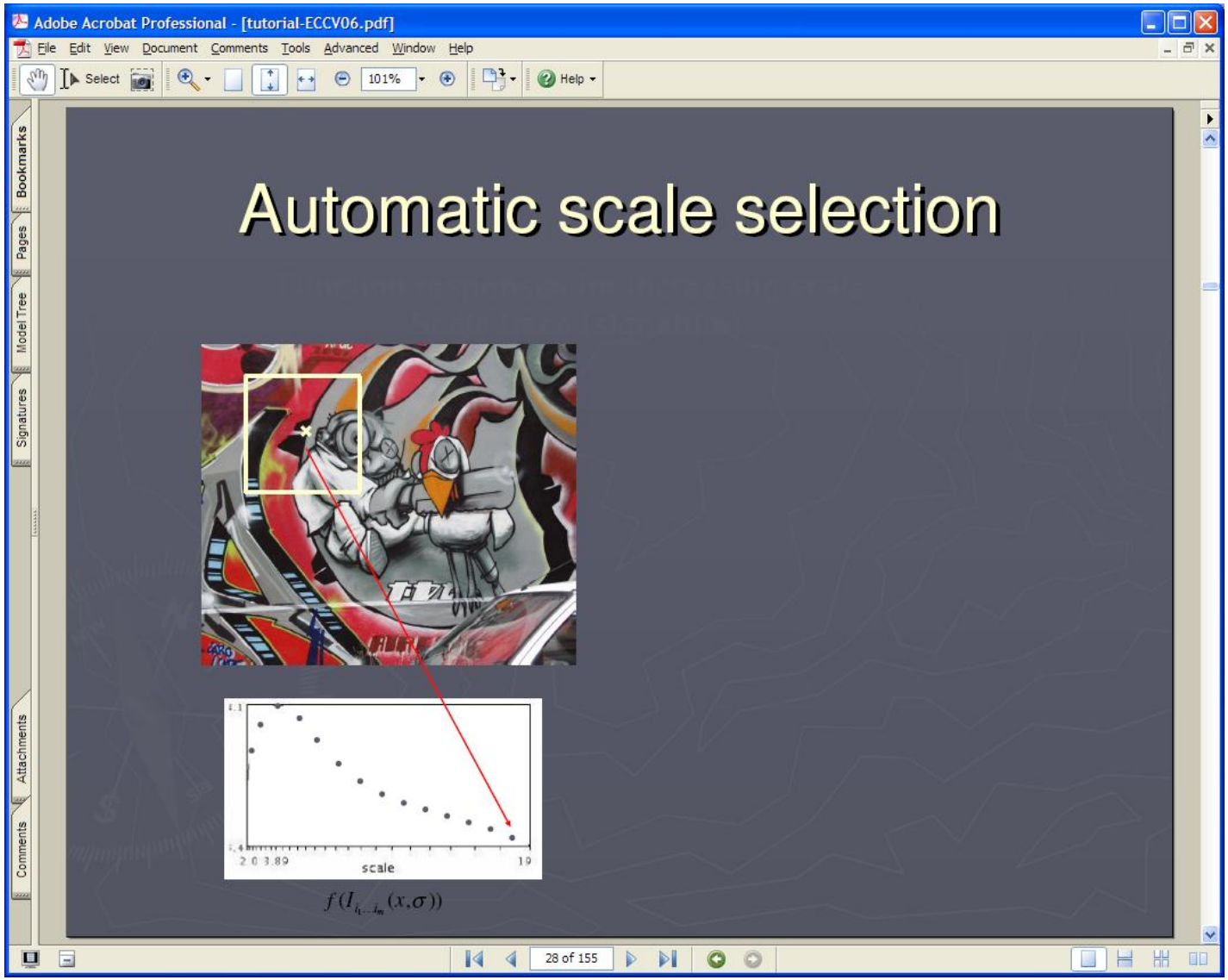
Scale invariant detection



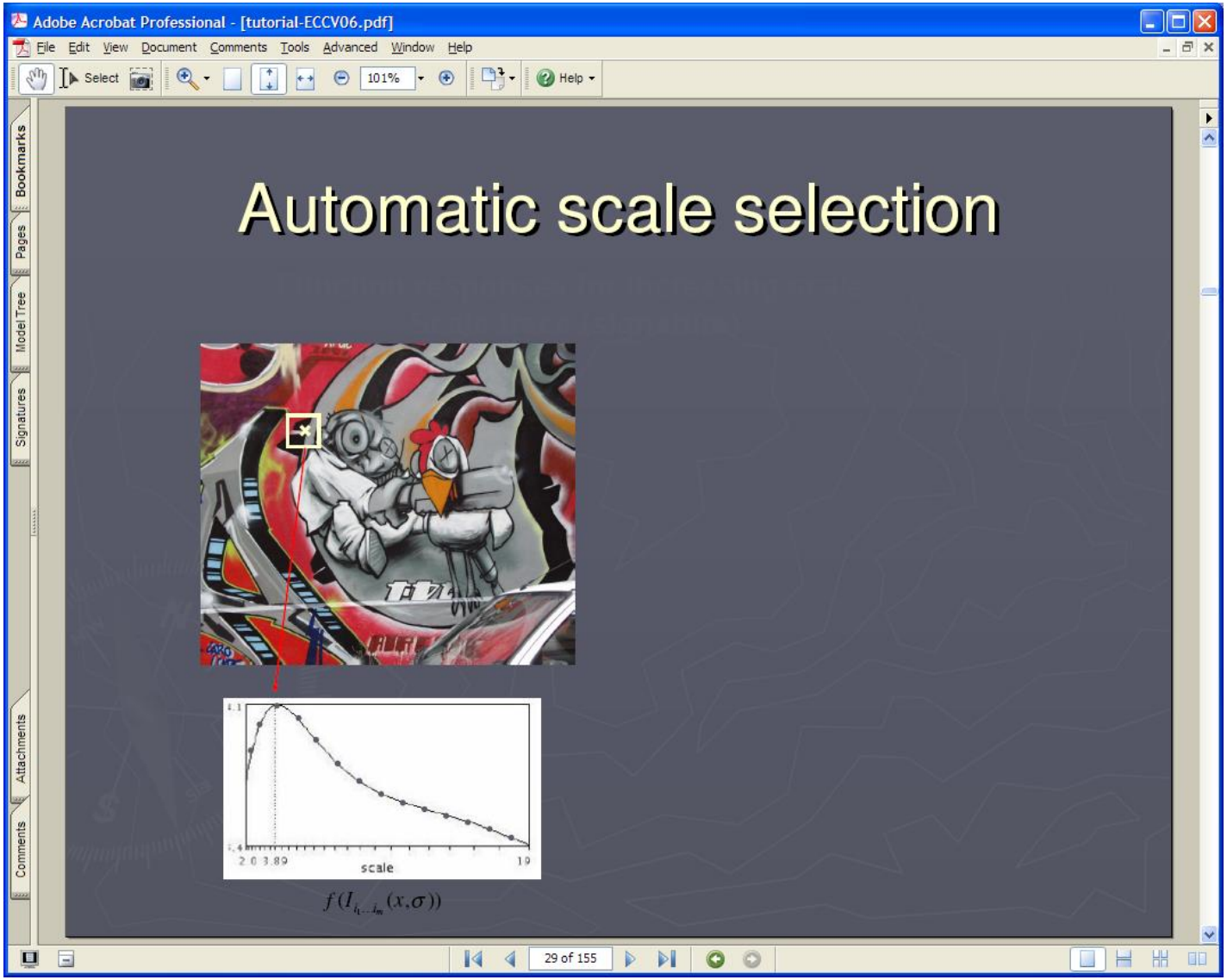
Scale invariant detection



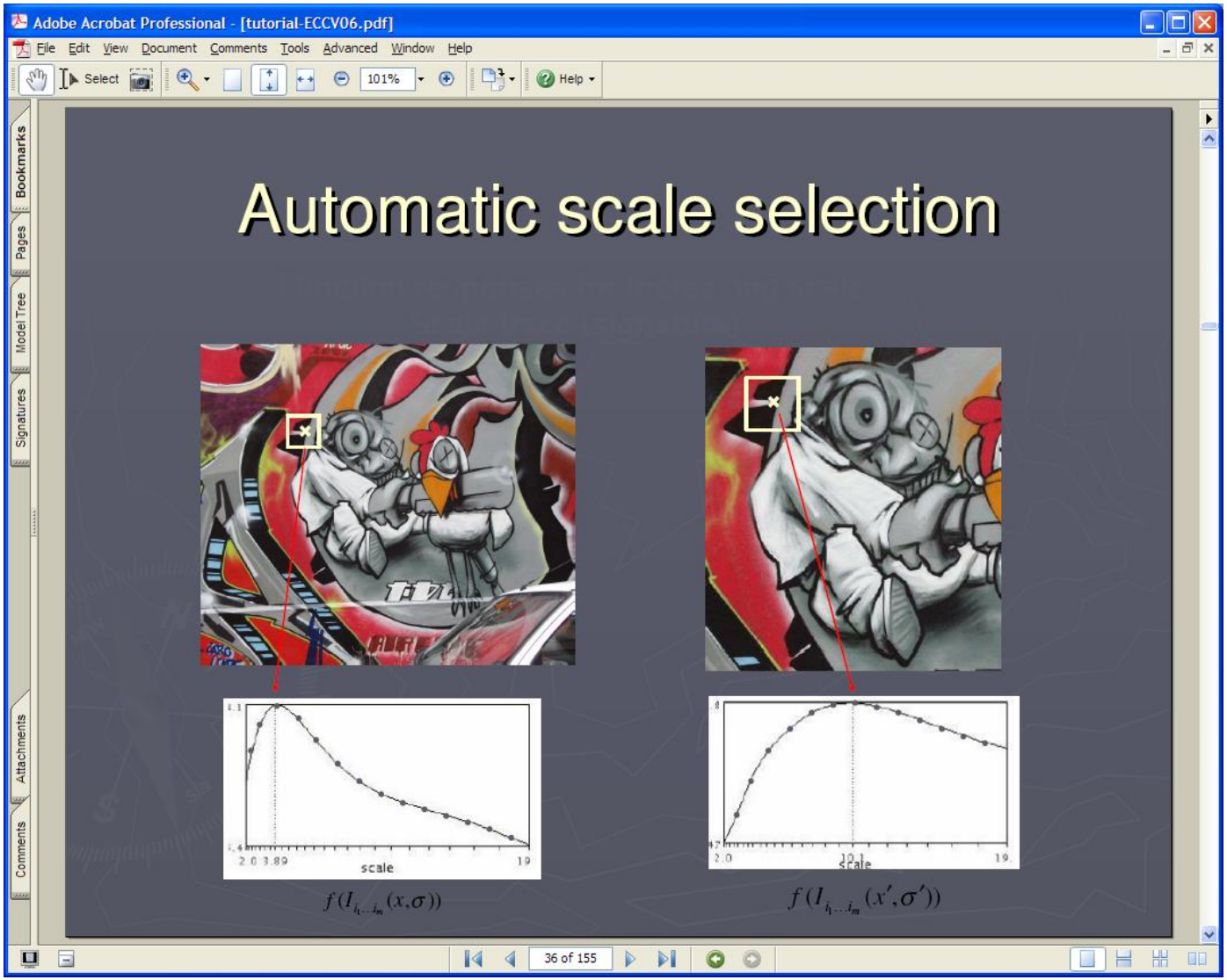
Scale invariant detection



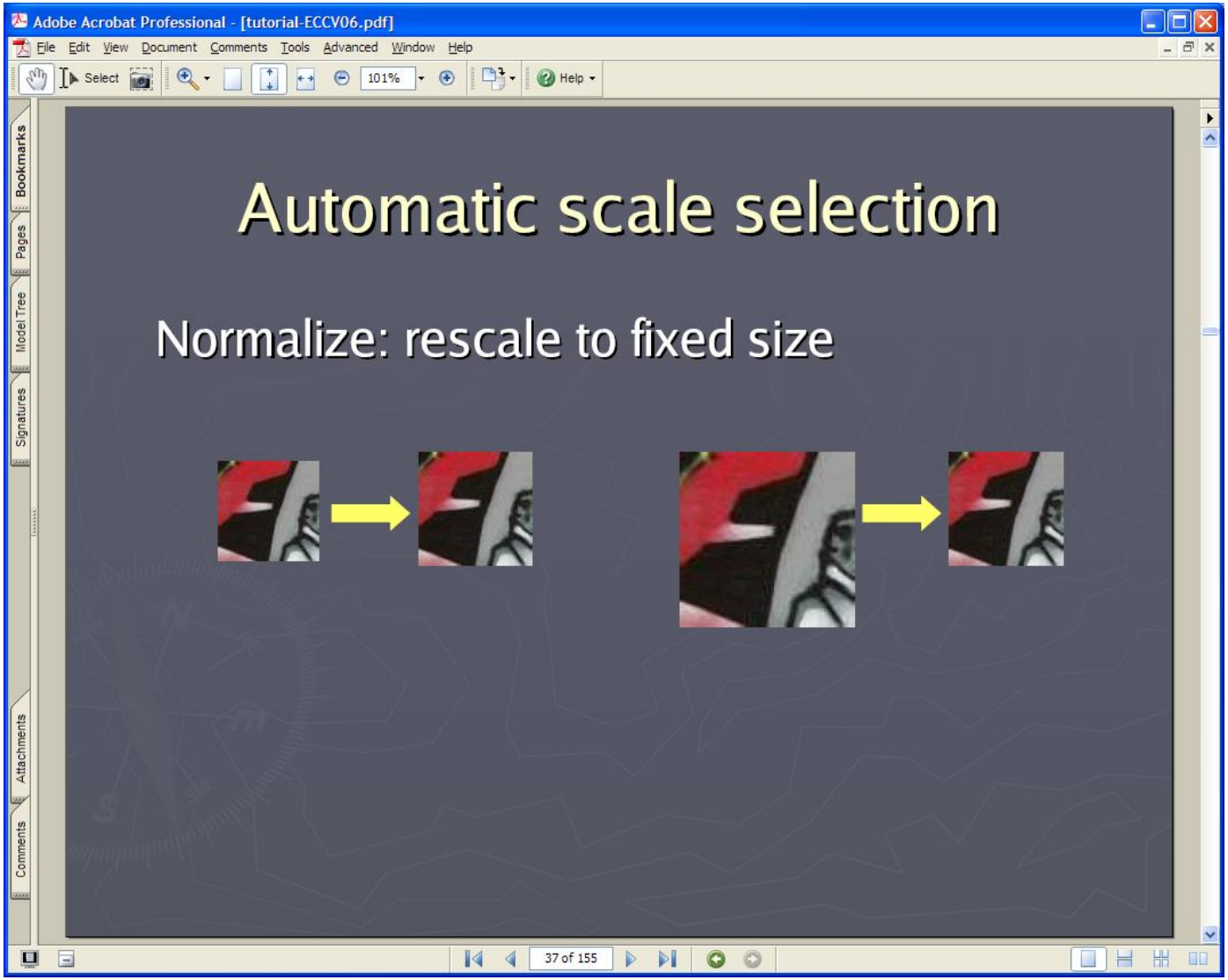
Scale invariant detection



Scale invariant detection

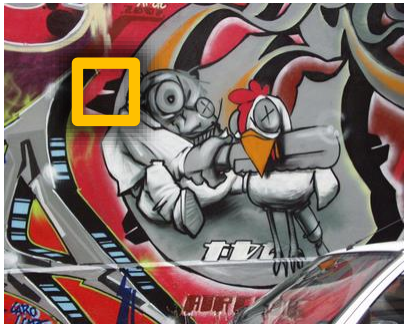


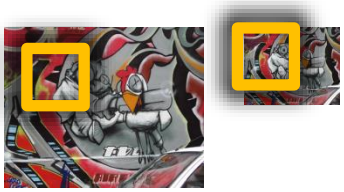
Scale invariant detection



Implementation

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

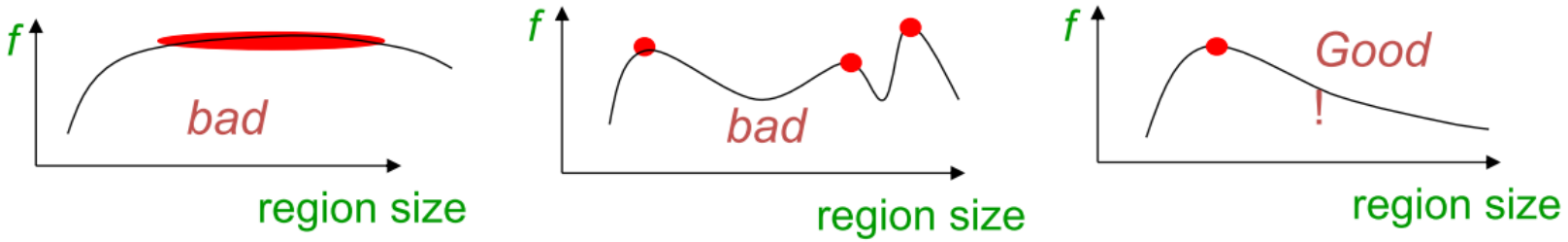




(sometimes need to create in-between levels, e.g. a ¾-size image)

Scale invariant detection

- Scale Invariant Detection
  - A “good” function for scale detection: has one stable sharp peak



  - For usual images: a good function would be one which responds to contrast (sharp local intensity change)

Scale invariant detection

- Scale Invariant Detection
  - Functions for determining scale

$$f = \text{Kernel} * \text{Image}$$

  - Laplacian Kernels:

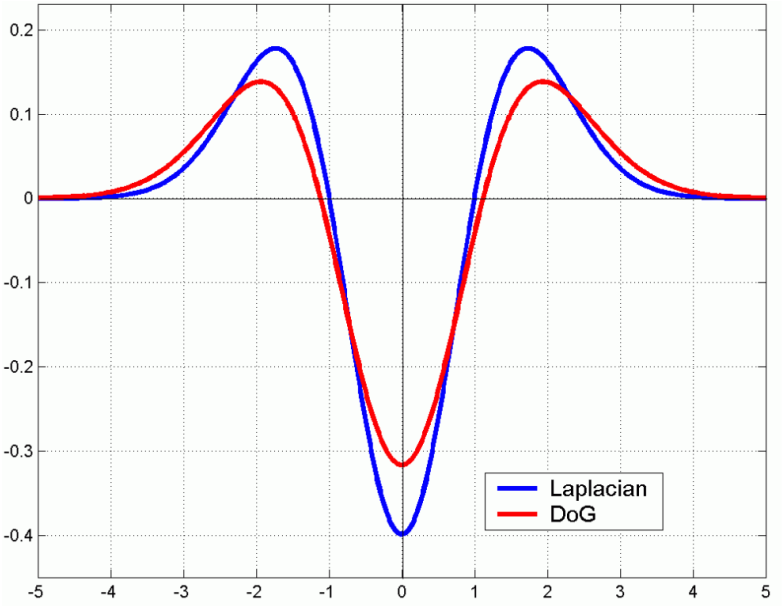
$$L = \sigma^2(G_{xx}(x, y, \sigma) + G_{yy}(x, y, \sigma))$$

  - Difference of Gaussians:

$$DoG = G(x, y, k\sigma) - G(x, y, \sigma)$$

  - Gaussian

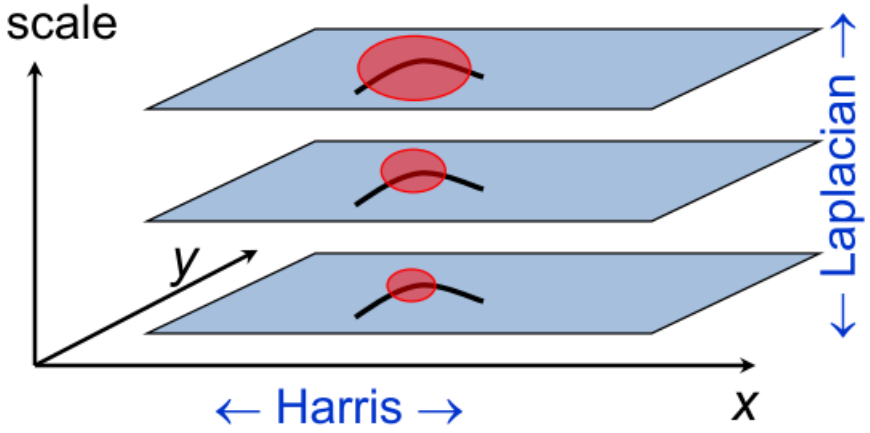
$$G(x, y, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



Scale invariant detectors

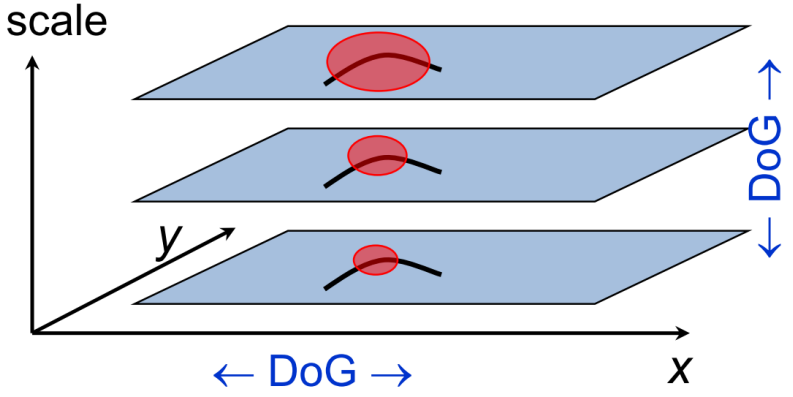
- Scale Invariant Detectors
  - Harris-Laplacian

    Find local maximum of:
    - Harris corner detector in space (image coordinates)
    - Laplacian in scale

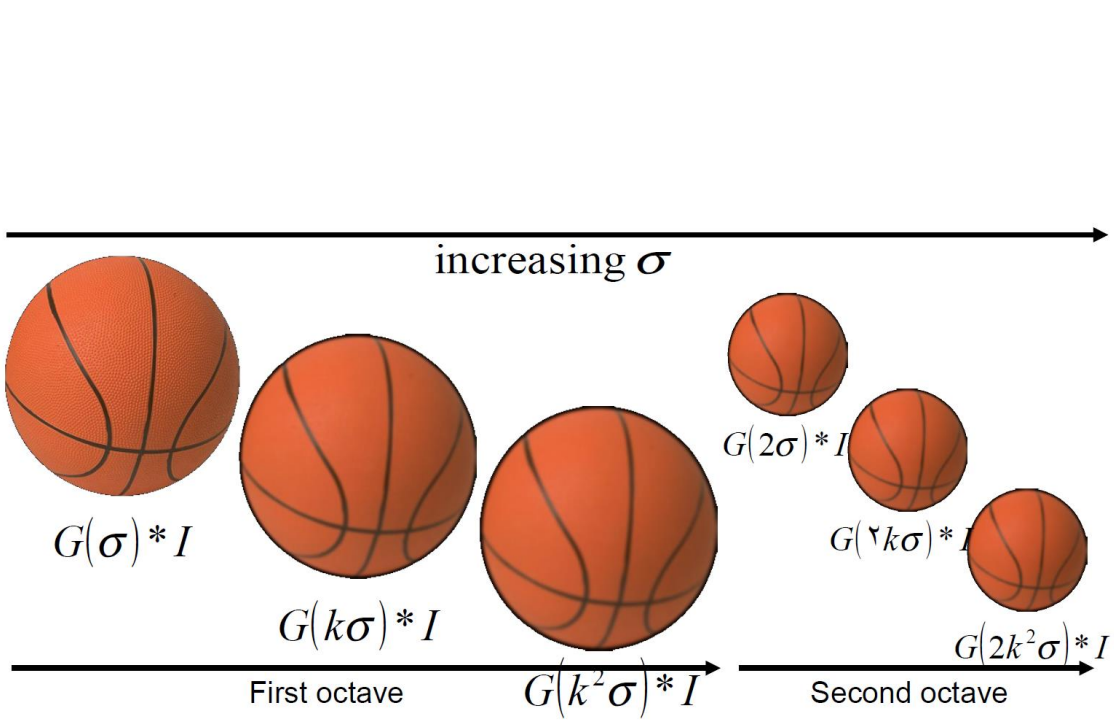


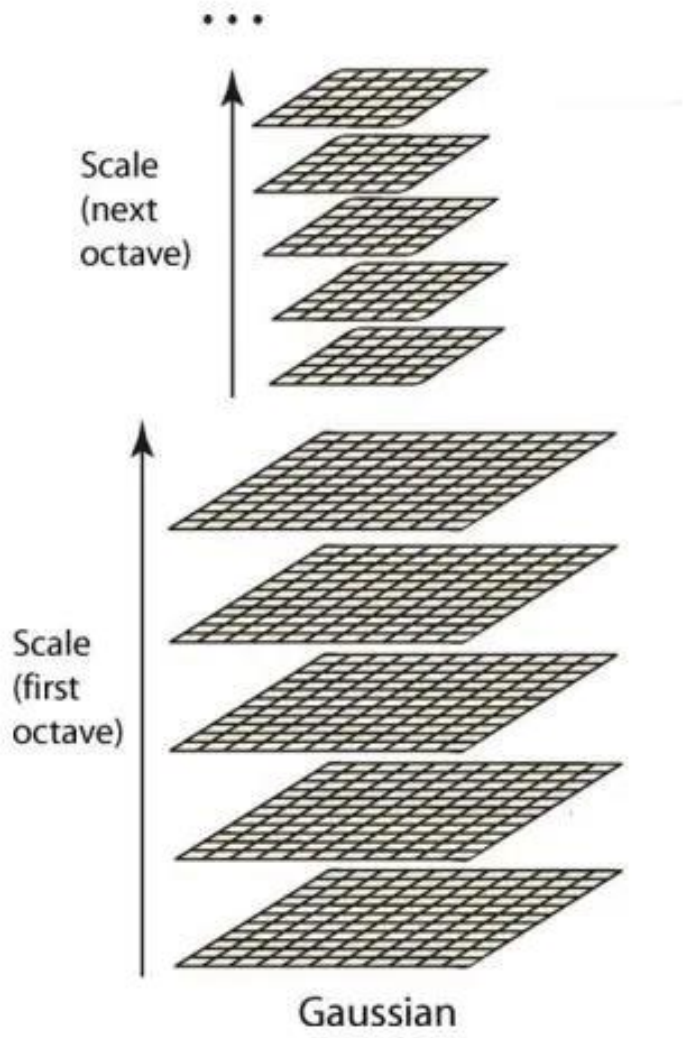
  - DoG (from SIFT by Lowe)

    Find local maximum of:
    - Difference of Gaussians in space and scale



Scale-Space Construction

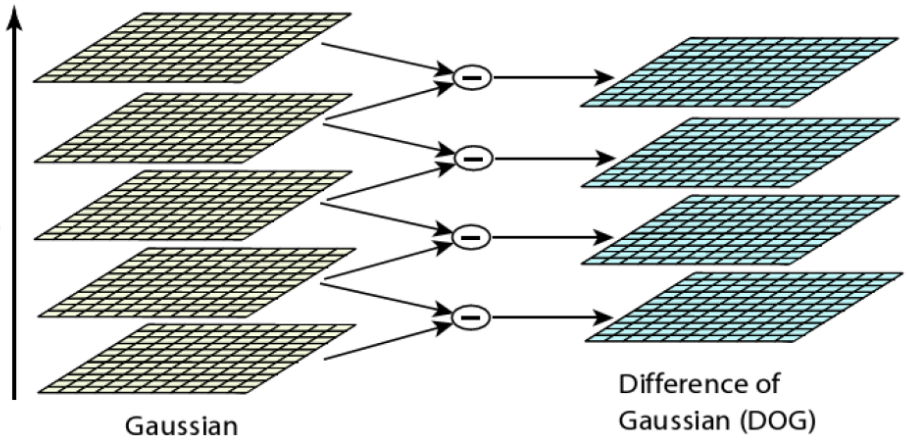




Scale invariant detectors

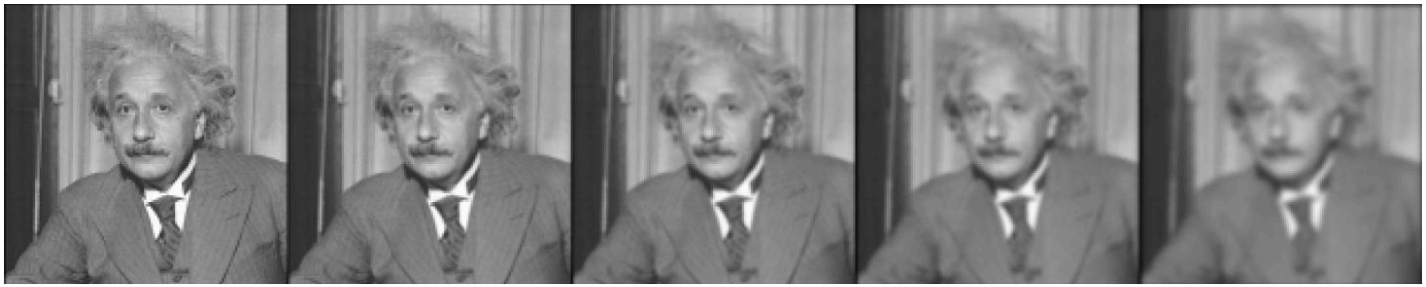
- Difference-of-Gaussians

$G(k^2\sigma)*$
$G(k\sigma)*I$
$G(\sigma)*I$



$$D(\sigma) \equiv (G(k\sigma) - G(\sigma)) * I$$

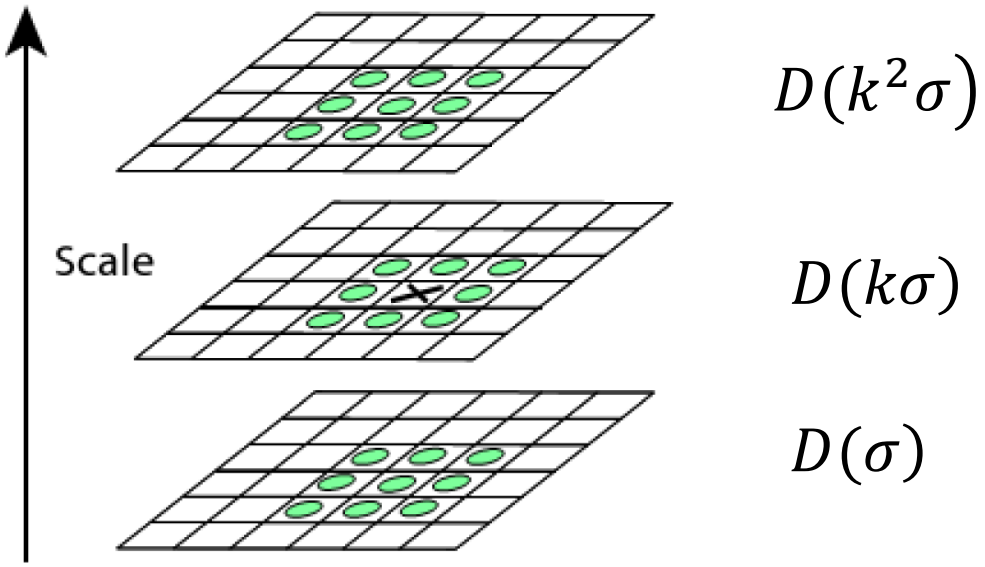
Gaussian:



DoG:

Scale invariant detectors

- Scale-Space Extrema
  - Choose all extrema within 3x3x3 neighborhood.

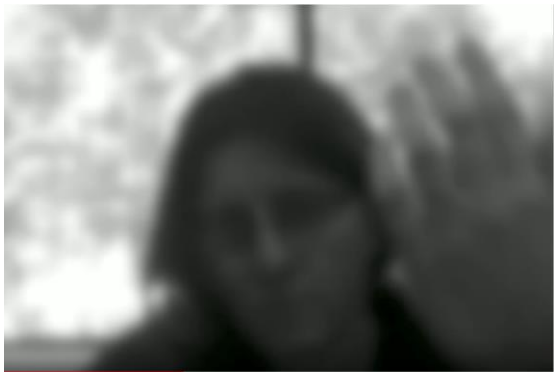


X is selected if it is larger or smaller than all 26 neighbors

Scale invariant detectors

- Example of Gaussian kernel

Original video



Blurred with a gaussian kernel

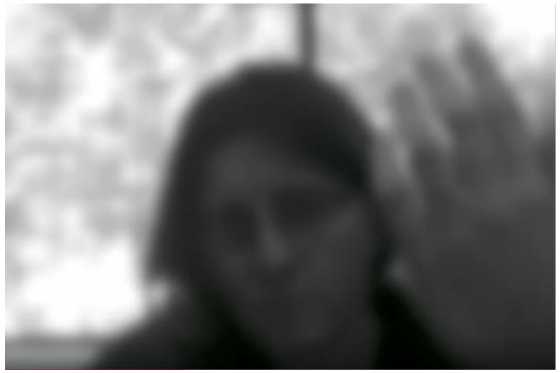
Blurred with a different gaussian kernel

What happens if you subtract one blurred image from another?

Scale invariant detectors

- Difference of Gaussians (DoG)

Original video



Blurred with a gaussian kernel k_1

Blurred with a different gaussian kernel k_2

DoG: $k_1 - k_2$

DoG: $k_1 - k_3$

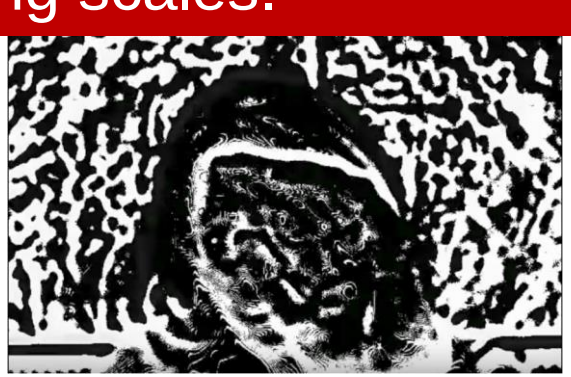
DoG: $k_1 - k_4$

Scale invariant detectors

- Difference of Gaussians (DoG)

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: $k_1 - k_3$

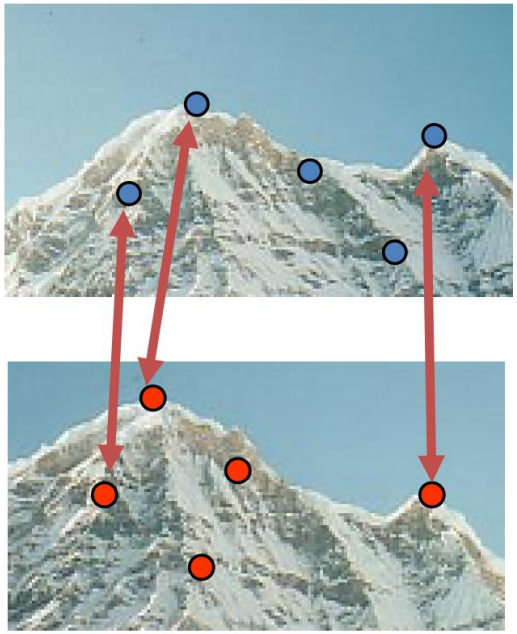
DoG: $k_1 - k_4$

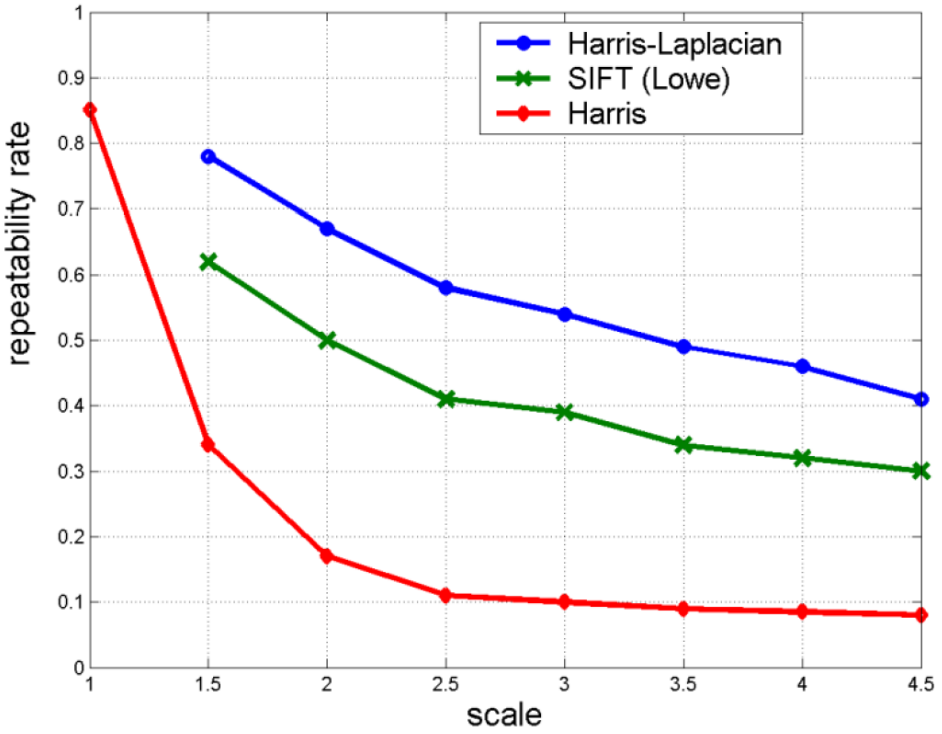
Scale invariant detection

- Scale Invariant Detectors
  - Experimental evaluation of detectors w.r.t. scale change

Repeatability rate:

$$\frac{correspondences}{possible\ correspondences}$$





Scale invariant detection

- Scale Invariant Detection: Summary
  - Given: two images of the same scene with a large scale difference between them
  - Goal: find the same interest points independently in each image
  - Solution: search for maxima of suitable functions in scale and in space (over the image)

Methods:

1. Harris-Laplacian [Mikolajczyk, Schmid]: maximize Laplacian over scale, Harris' measure of corner response over the image
2. SIFT [Lowe]: maximize Difference of Gaussians over scale and space

What's next?

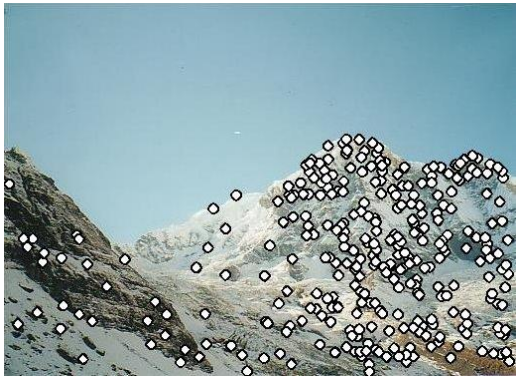
- So we can have detect keypoints at varying scales. But what can we do with those keypoints?
- Things we would like to do:
  - Search:
    - We would need to find similar key points in other images
  - Panorama
    - Match keypoints from one image to another.
  - Etc…
- For all such applications, we need a way of `describing` the keypoints.

What we will learn today?

- Scale invariant keypoint detection
  - Automatic scale selection
  - Difference-of-Gaussian (DoG) detector
- SIFT: an image region descriptor
- Application: Panorama

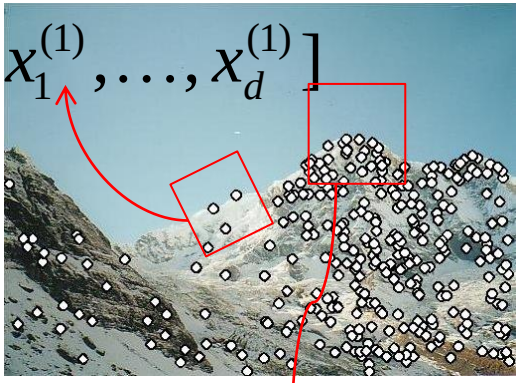
Local features: main components

1) Detection: Identify the interest points



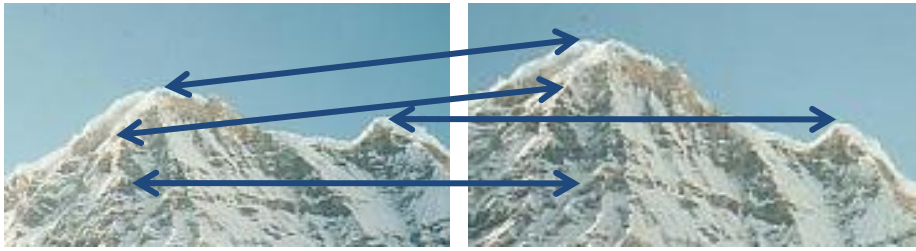
2) **Description: Extract vector feature descriptor surrounding each interest point.**

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



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

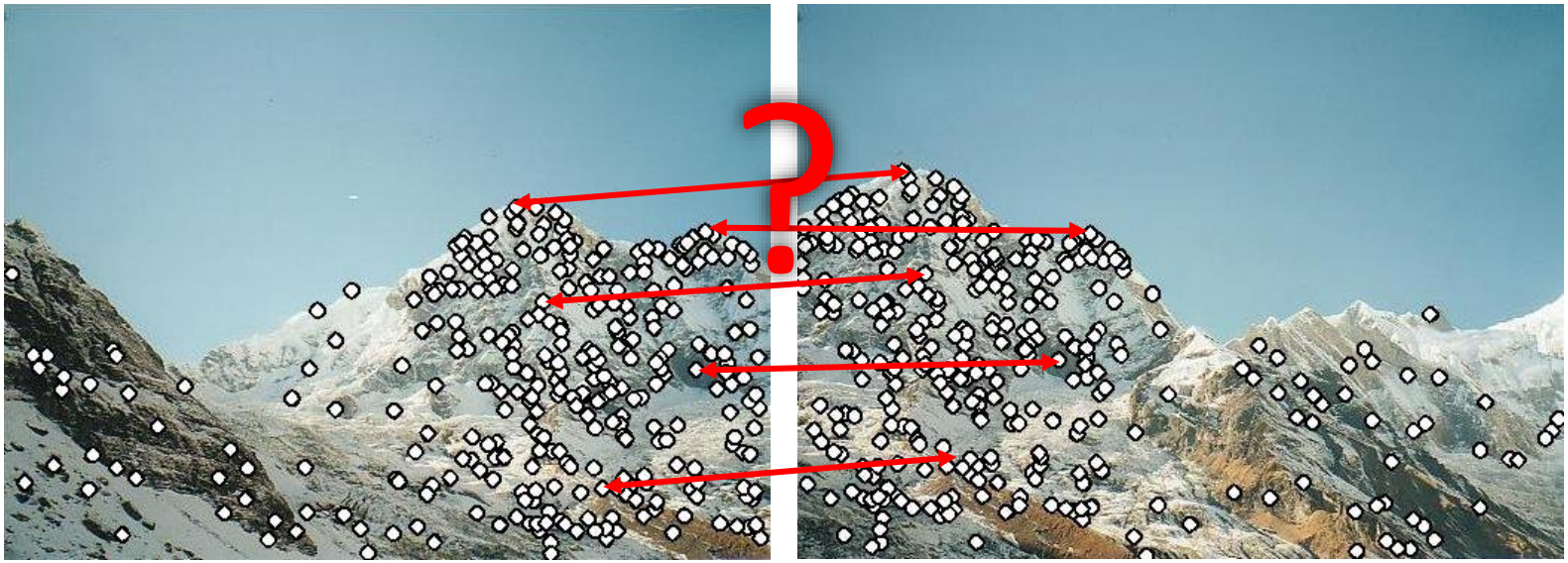
3) Matching: Determine correspondence between descriptors in two views



Feature descriptors

We know how to detect good points

Next question: **How to match them?**

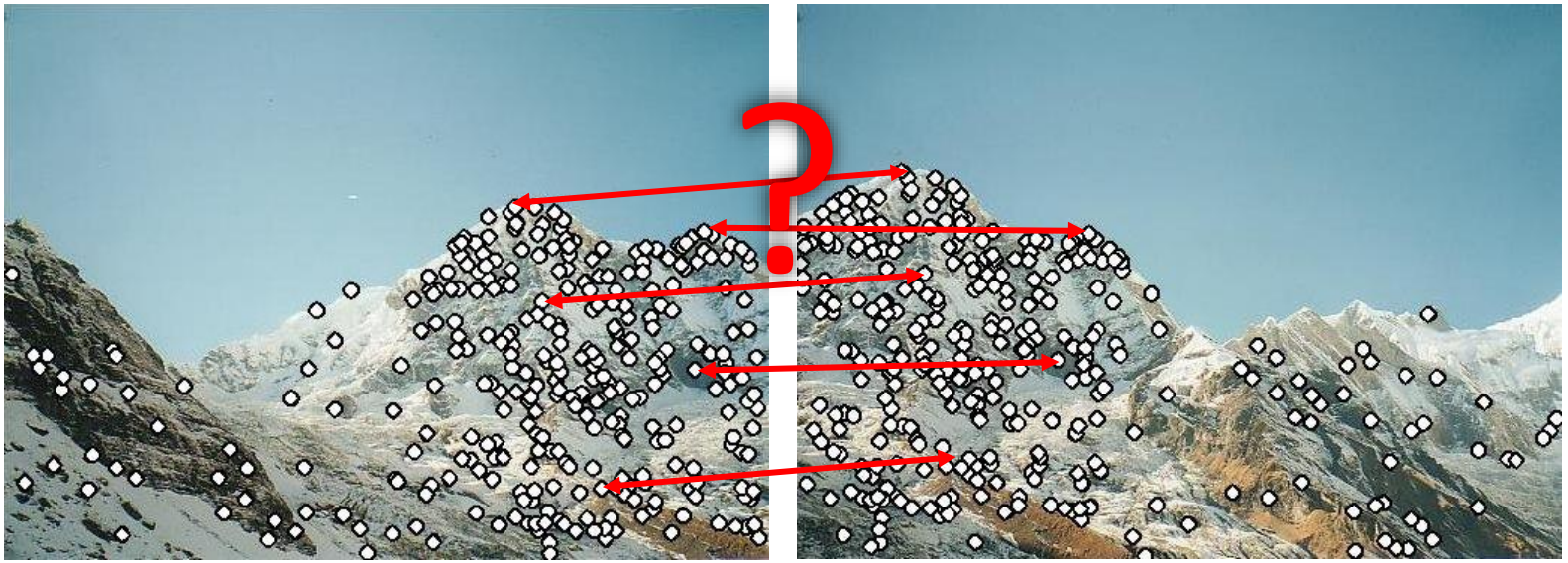


Answer: Come up with a *descriptor* for each point, find similar descriptors between the two images

Feature descriptors

We know how to detect good points

Next question: **How to match them?**



Lots of possibilities

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

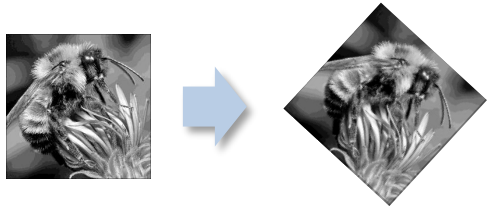
Invariance vs. discriminability

- Invariance:
  - Descriptor shouldn't change even if image is transformed
- Discriminability:
  - Descriptor should be highly unique for each point

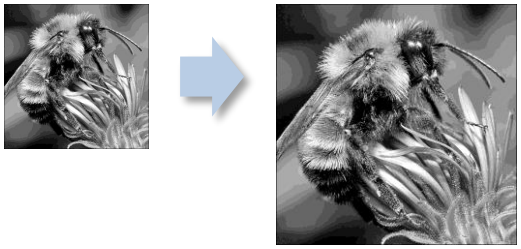
Image transformations revisited

- Geometric

Rotation

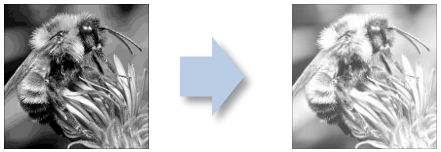


Scale



- Photometric

Intensity change



How to achieve invariance

Need both of the following:

1. Make sure your detector is invariant
2. Design an invariant feature descriptor
   - Simplest descriptor: a single 0
     - What's this invariant to?
   - Next simplest descriptor: a square, axis-aligned 5x5 window of pixels
     - What's this invariant to?
   - Let's look at some better approaches…

Rotation invariance for feature descriptors

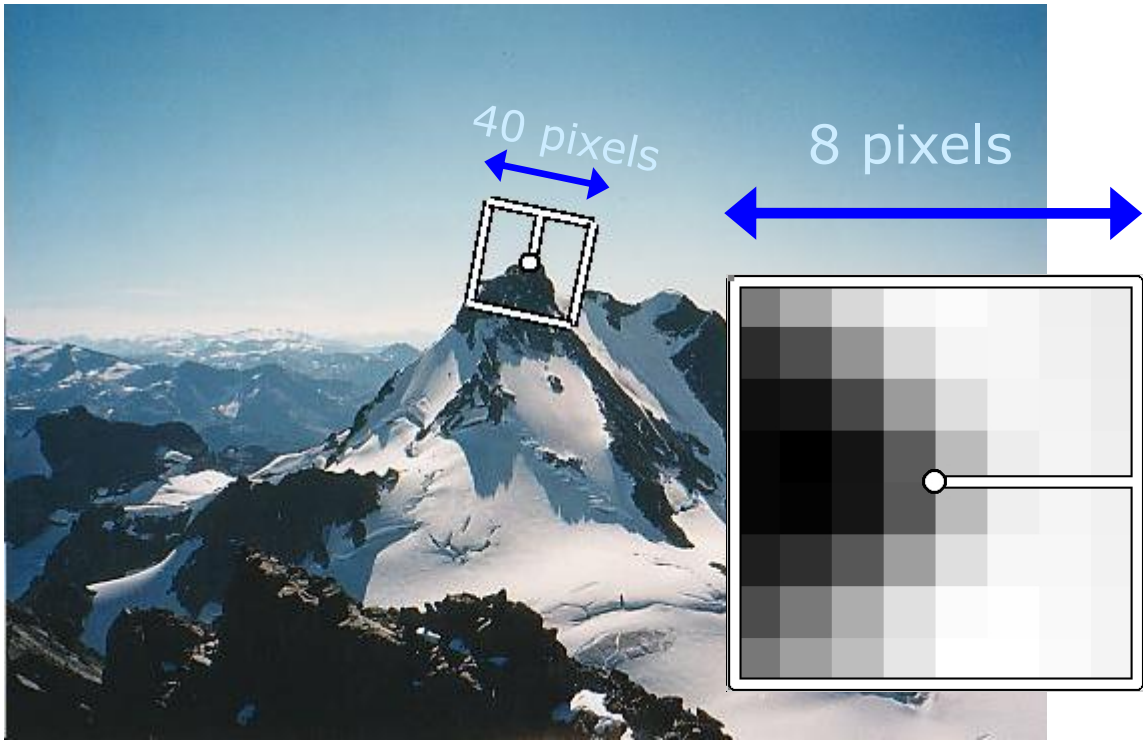
- Find dominant orientation of the image patch
  - E.g., given by $\mathbf{x}_{max}$, the eigenvector of **H** corresponding to λ_{max} (the *larger* eigenvalue)
  - Or simply the orientation of the (smoothed) gradient
  - Rotate the patch according to this angle

Figure by Matthew Brown

Multiscale Oriented PatcheS descriptor

Take 40x40 square window around detected feature

- Scale to 1/5 size (using prefiltering)
- Rotate to horizontal
- Sample 8x8 square window centered at feature
- Intensity normalize the window by subtracting the mean, dividing by the standard deviation in the window (why?)



Adapted from slide by Matthew Brown

Detections at multiple scales

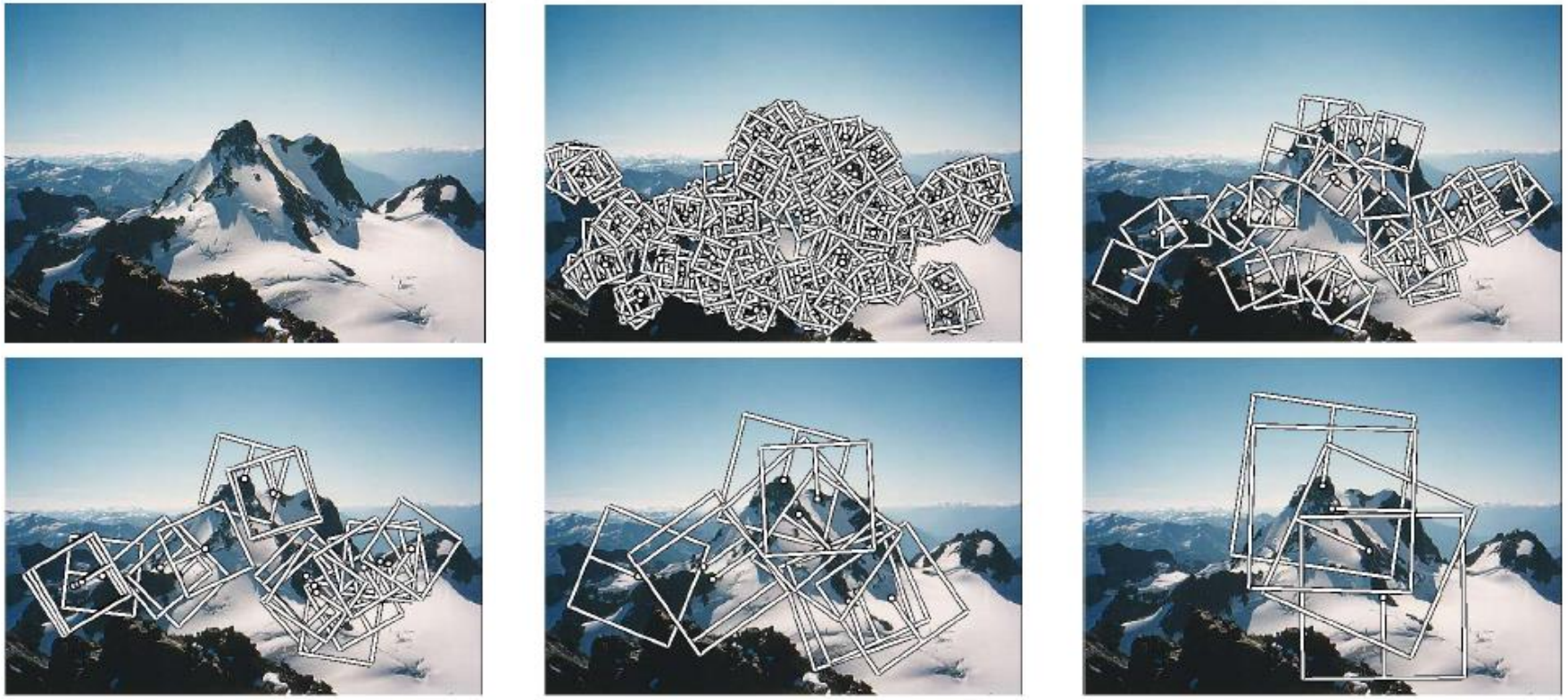
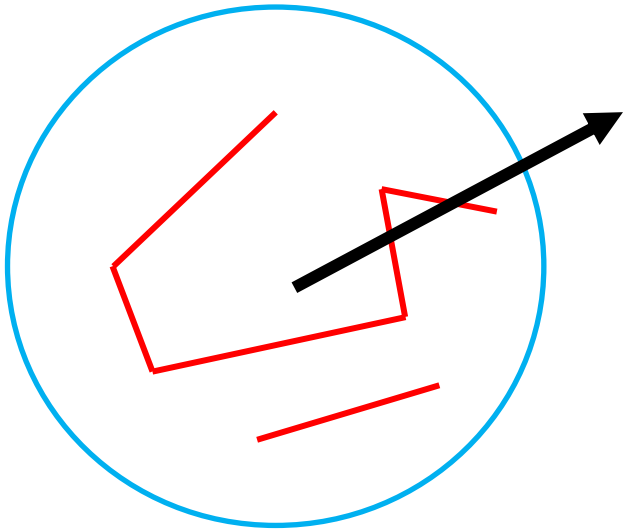


Figure 1. Multi-scale Oriented Patches (MOPS) extracted at five pyramid levels from one of the Matier images. The boxes show the feature orientation and the region from which the descriptor vector is sampled.

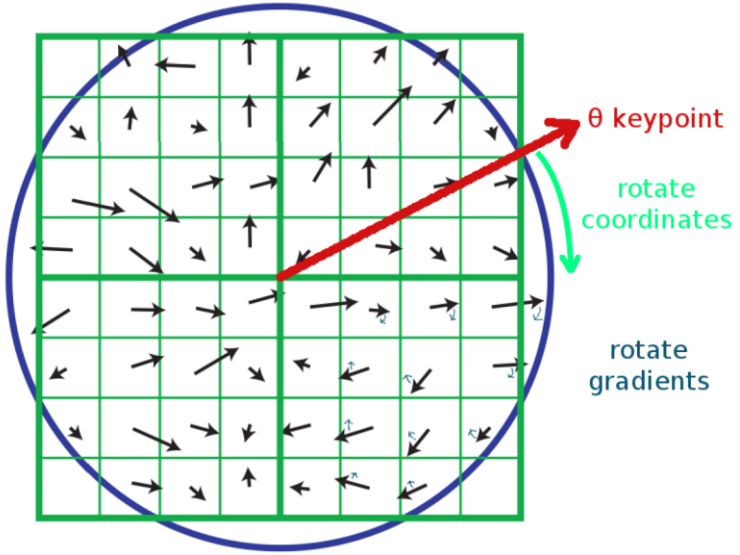
SIFT: an image region descriptor

- Becoming rotation invariant
  - We are given a keypoint and its scale from DoG
  - We will select a characteristic orientation for the keypoint (based on the most prominent gradient there; discussed next slide)
  - Causes features to be rotation invariant!
    - If the keypoint appears rotated in another image, the features will be the same, because they're relative to the characteristic orientation

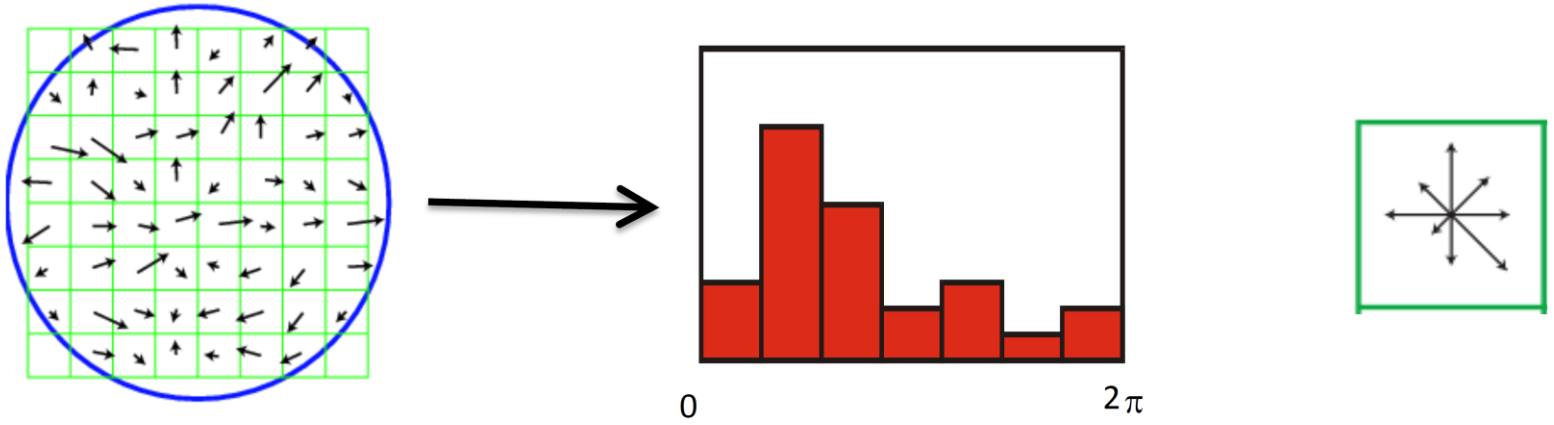


SIFT: an image region descriptor

- SIFT descriptor formation
  - Use the blurred image associated with the keypoint's scale
  - Take image gradients over the keypoint neighborhood.
  - To become rotation invariant, rotate the gradient directions AND locations by (-keypoint orientation)
    - Now we've cancelled out rotation and have gradients expressed at locations relative to keypoint orientation θ
    - We could also have just rotated the whole image by -θ, but that would be slower.



SIFT: an image region descriptor

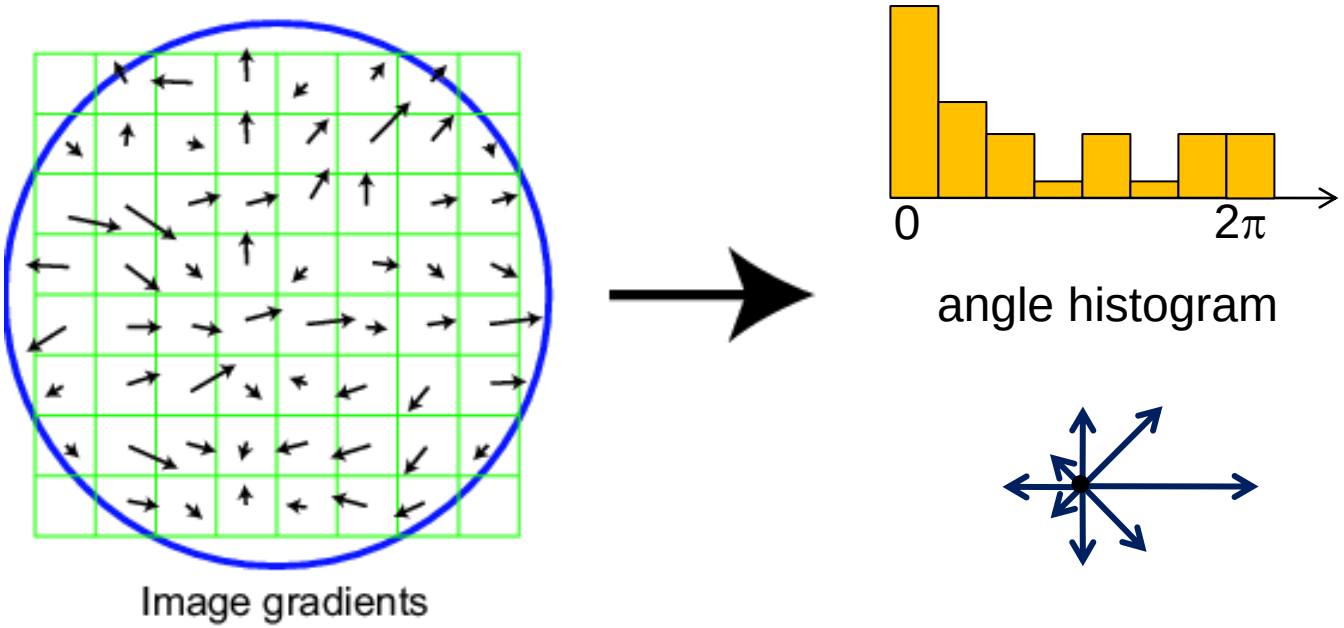


- Using precise gradient locations is fragile. We'd like to allow some "slop" in the image, and still produce a very similar descriptor
- Create array of orientation histograms (a 4x4 array is shown)
- Put the rotated gradients into their local orientation histograms
  - A gradients's contribution is divided among the nearby histograms based on distance. If it's halfway between two histogram locations, it gives a half contribution to both.
  - Also, scale down gradient contributions for gradients far from the center
- The SIFT authors found that best results were with 8 orientation bins per histogram.

SIFT: an image region descriptor

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

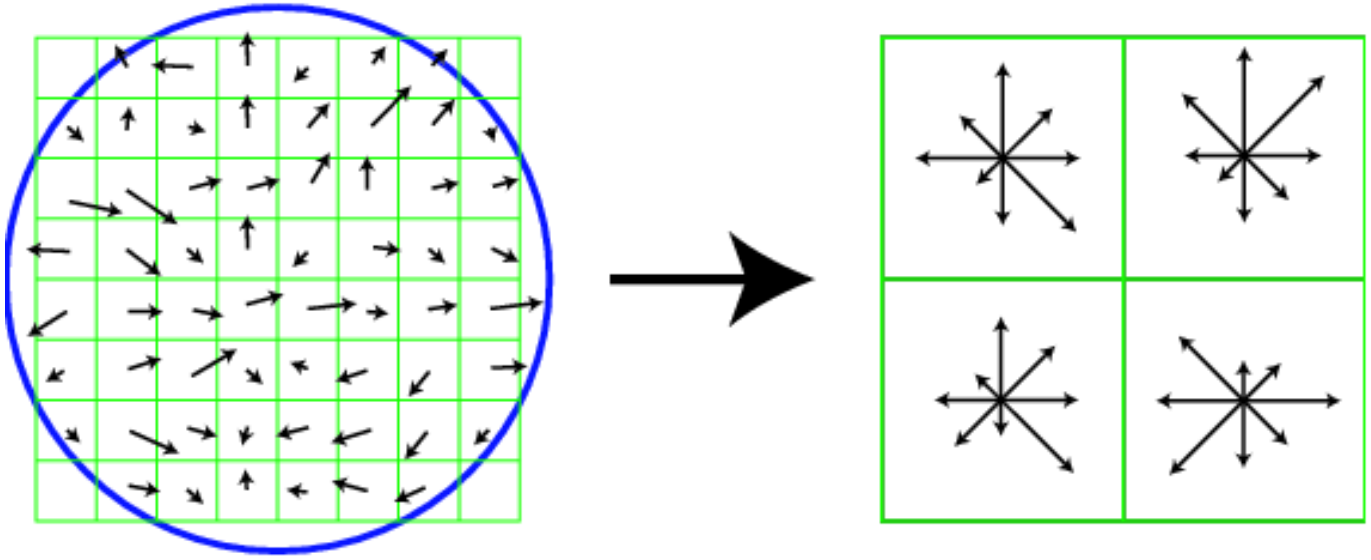


Adapted from slide by David Lowe

SIFT: an image region descriptor

Full version

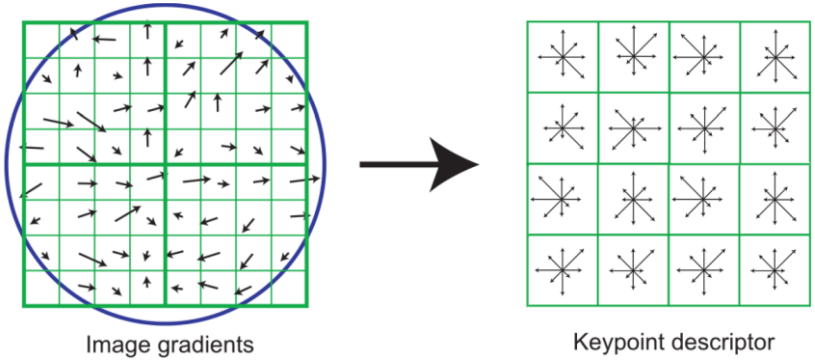
- Divide the 16x16 window into a 4x4 grid of cells (2x2 case shown below)
- Compute an orientation histogram for each cell
- 16 cells * 8 orientations = 128 dimensional descriptor



Adapted from slide by David Lowe

SIFT: an image region descriptor

- SIFT descriptor formation
  - 8 orientation bins per histogram, and a 4x4 histogram array, yields 8 x 4x4 = 128 numbers.
  - So a SIFT descriptor is a length 128 vector, which is invariant to rotation (because we rotated the descriptor) and scale (because we worked with the scaled image from DoG)
  - We can compare each vector from image A to each vector from image B to find matching keypoints!
    - Euclidean “distance” between descriptor vectors gives a good measure of keypoint similarity



SIFT: an image region descriptor

- SIFT descriptor formation
  - Adding robustness to illumination changes:
  - Remember that the descriptor is made of gradients (differences between pixels), so it's already invariant to changes in brightness (e.g. adding 10 to all image pixels yields the exact same descriptor)
  - A higher-contrast photo will increase the magnitude of gradients linearly. So, to correct for contrast changes, normalize the vector (scale to length 1.0)
  - Very large image gradients are usually from unreliable 3D illumination effects (glare, etc). So, to reduce their effect, clamp all values in the vector to be ≤ 0.2 (an experimentally tuned value). Then normalize the vector again.
  - Result is a vector which is fairly invariant to illumination changes.

SIFT: an image region descriptor

- Sensitivity to number of histogram orientations

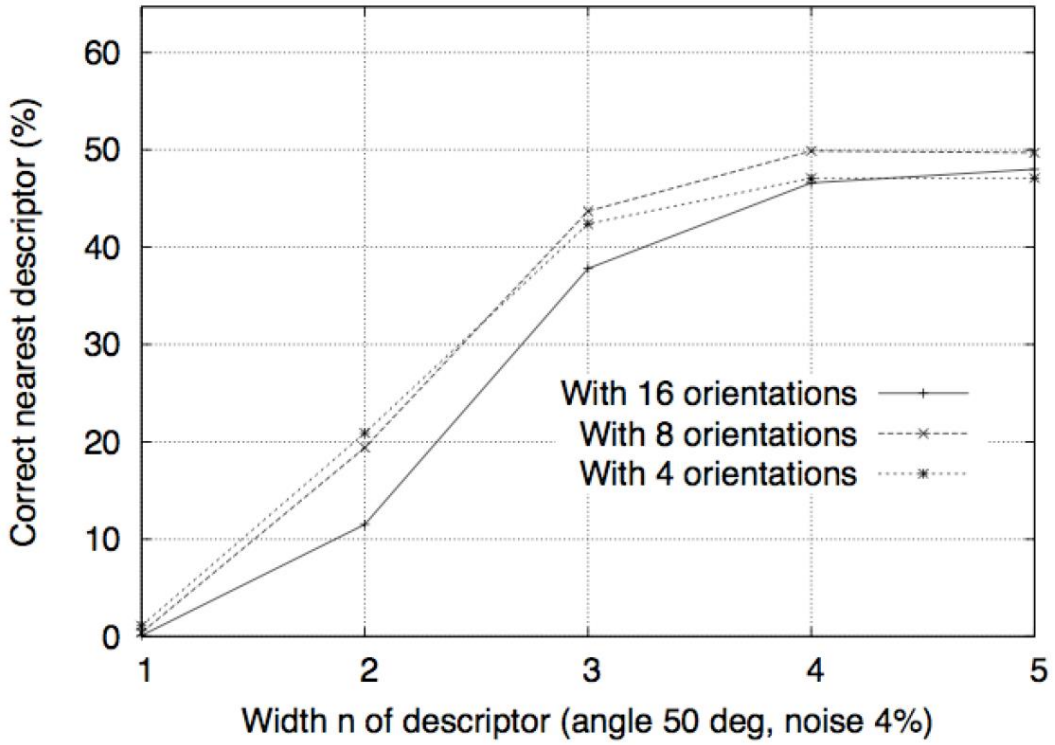
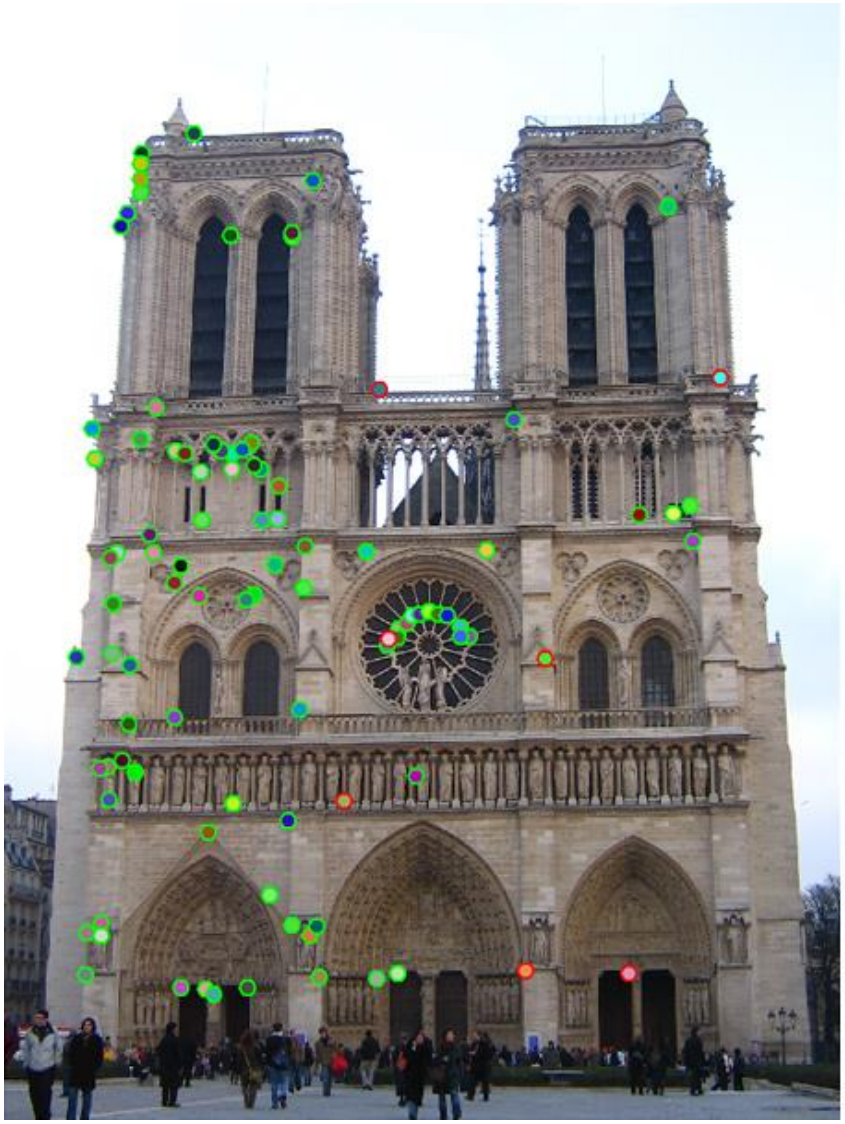
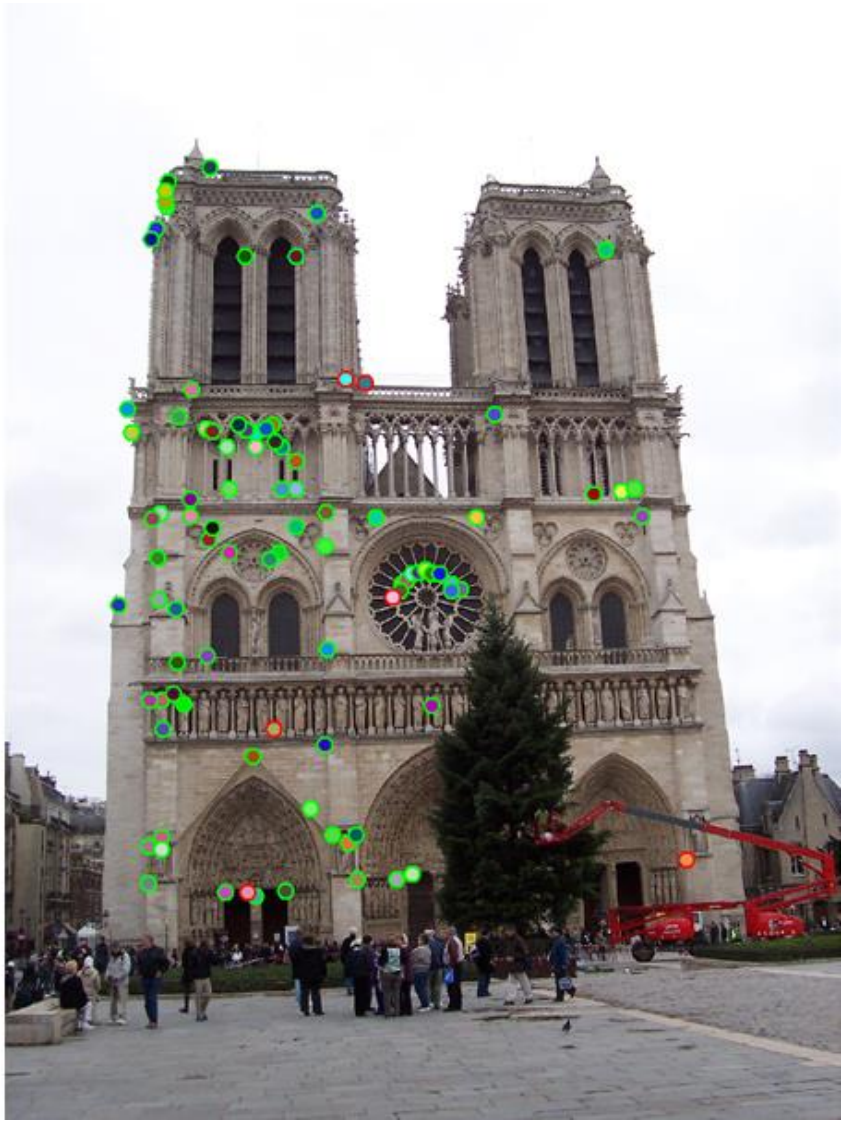


Figure 8: This graph shows the percent of keypoints giving the correct match to a database of 40,000 keypoints as a function of width of the $n \times n$ keypoint descriptor and the number of orientations in each histogram. The graph is computed for images with affine viewpoint change of 50 degrees and addition of 4% noise.

Which features match?





Feature matching

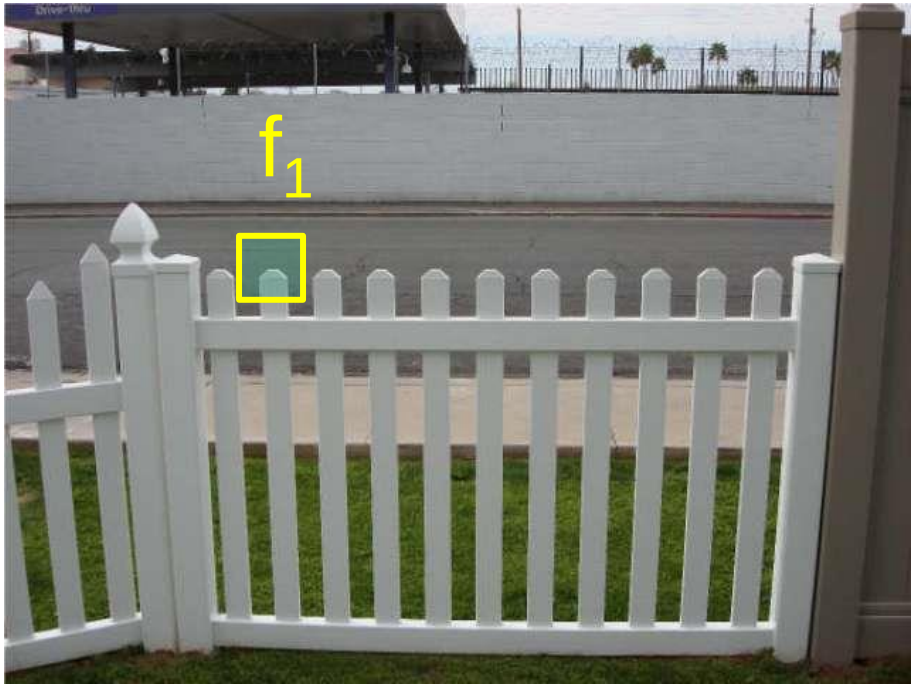
Given a feature in I_1, how to find the best match in I_2?

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

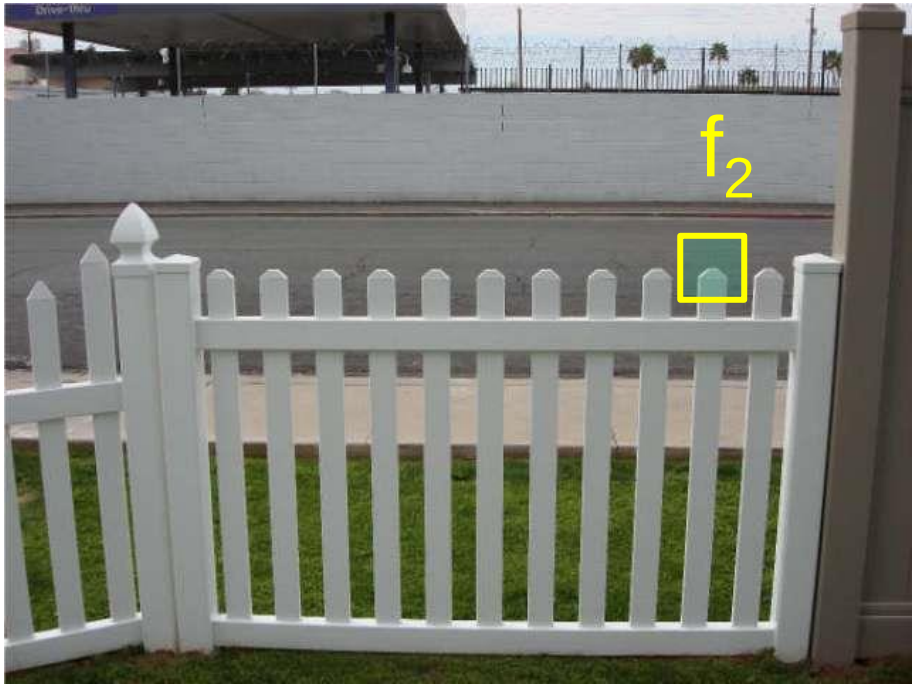
Feature distance

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

- Simple approach: L_2 distance, $||f_1 - f_2||$
- can give small distances for ambiguous (incorrect) matches



I_1

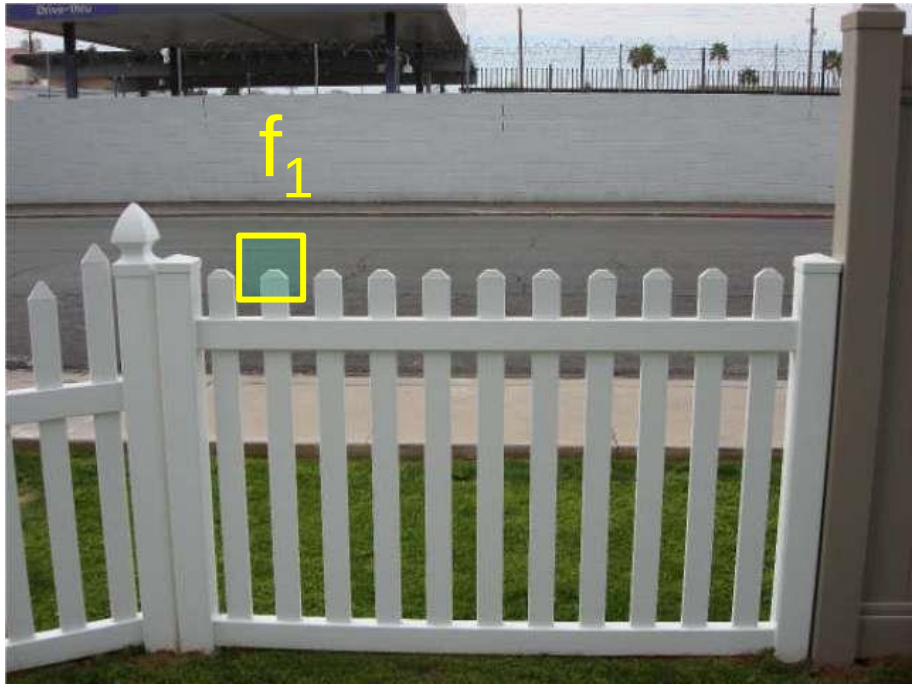


I_2

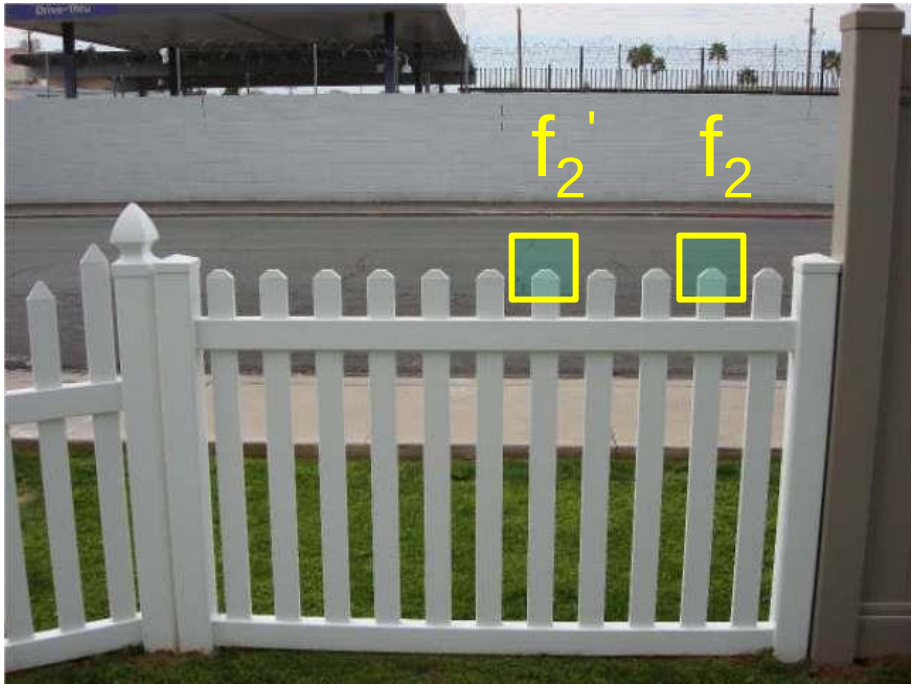
Feature distance

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

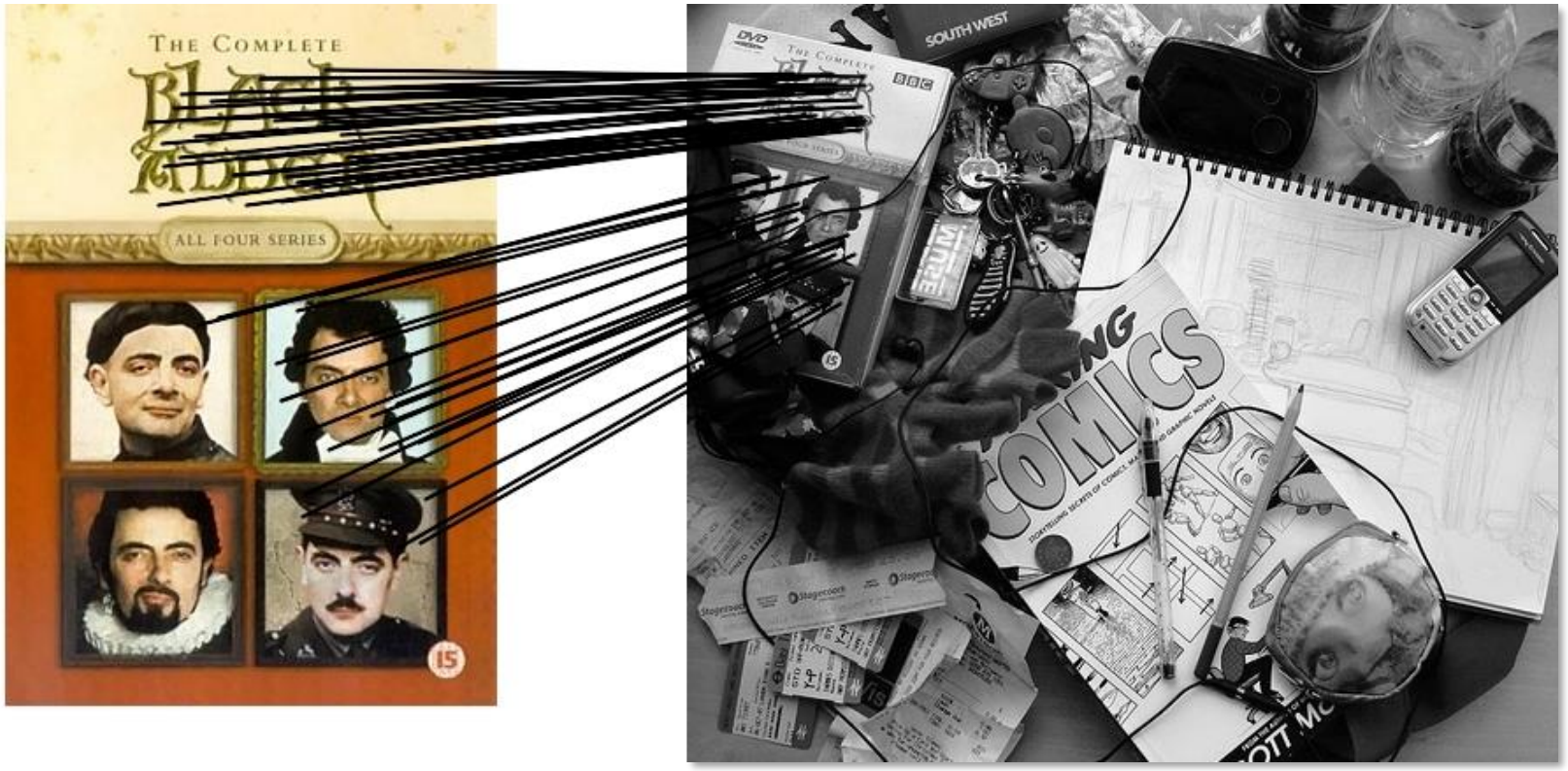


I_1



I_2

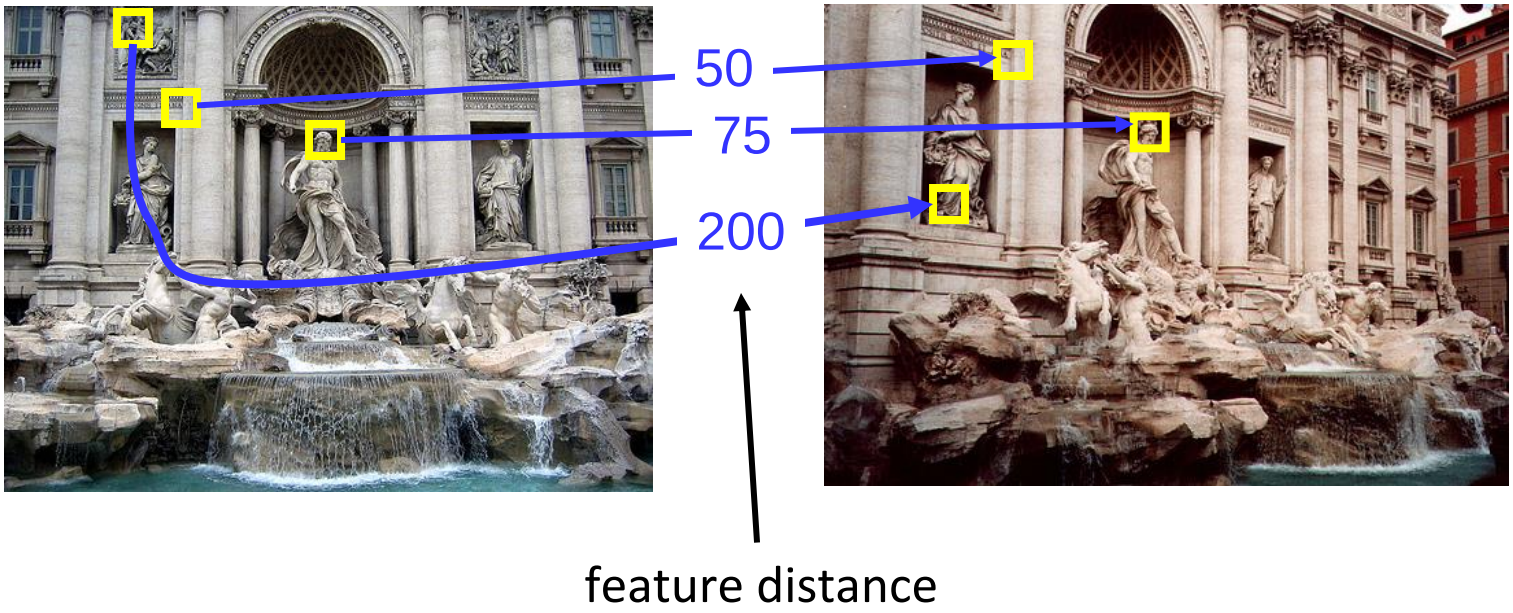
Feature matching example



58 matches (thresholded by ratio score)

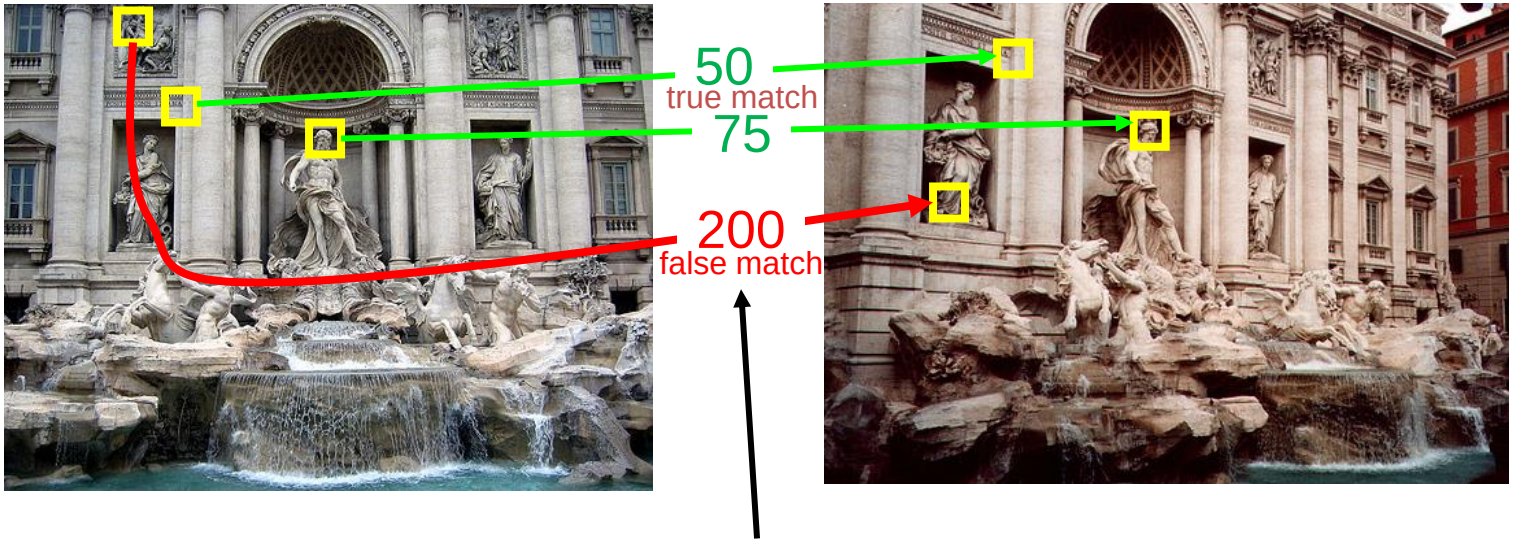
Evaluating the results

How can we measure the performance of a feature matcher?



True/false positives

How can we measure the performance of a feature matcher?

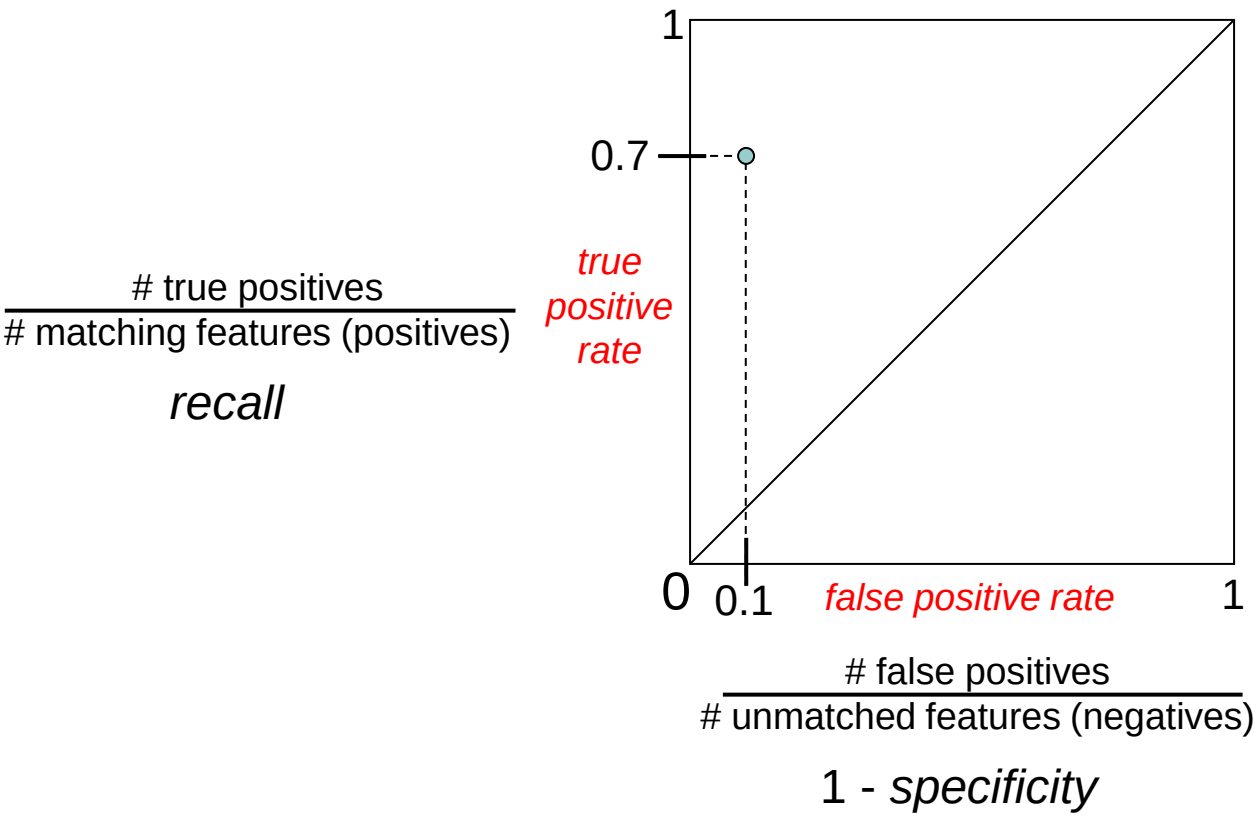


The distance threshold affects performance

- 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

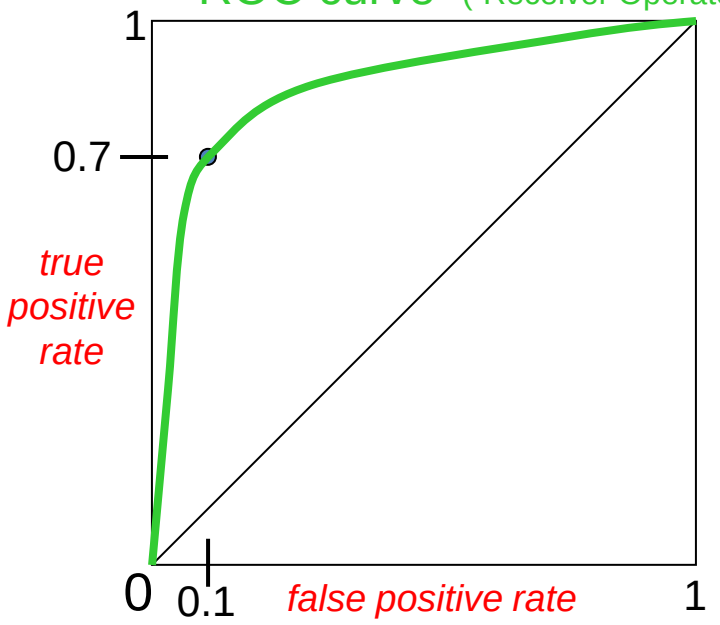
How can we measure the performance of a feature matcher?



Evaluating the results

How can we measure the performance of a feature matcher?

ROC curve ("Receiver Operator Characteristic")



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

recall

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

1 - *specificity*

Single number: Area Under the Curve (AUC)

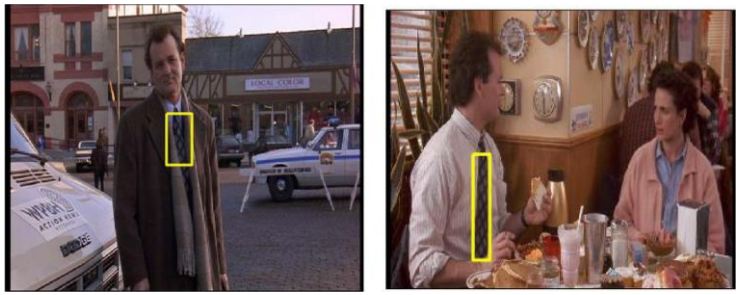
E.g. AUC = 0.87

1 is the best

SIFT: an image region descriptor

- Recognition of specific objects, scenes

Schmid and Mohr 1997



Sivic and Zisserman, 2003

Rothganger et al. 2003

Lowe 2002

SIFT: an image region descriptor

- Applications of local invariant features
  - Wide baseline stereo
  - Motion tracking
  - Panoramas
  - Mobile robot navigation
  - 3D reconstruction
  - Recognition
  - …

What we will learn today?

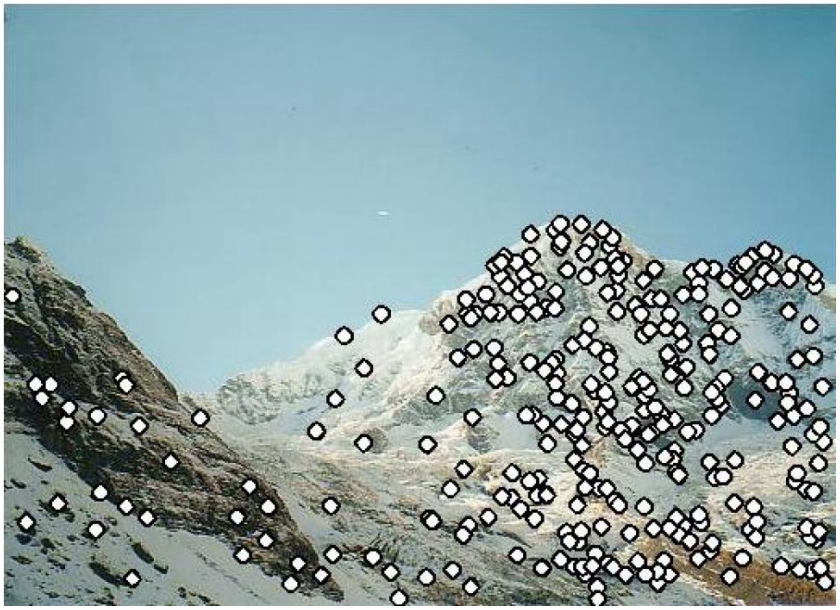
- A quick review
- Scale invariant region detection
  - Automatic scale selection
  - Difference-of-Gaussian (DoG) detector
- SIFT: an image region descriptor
- Application: Panorama

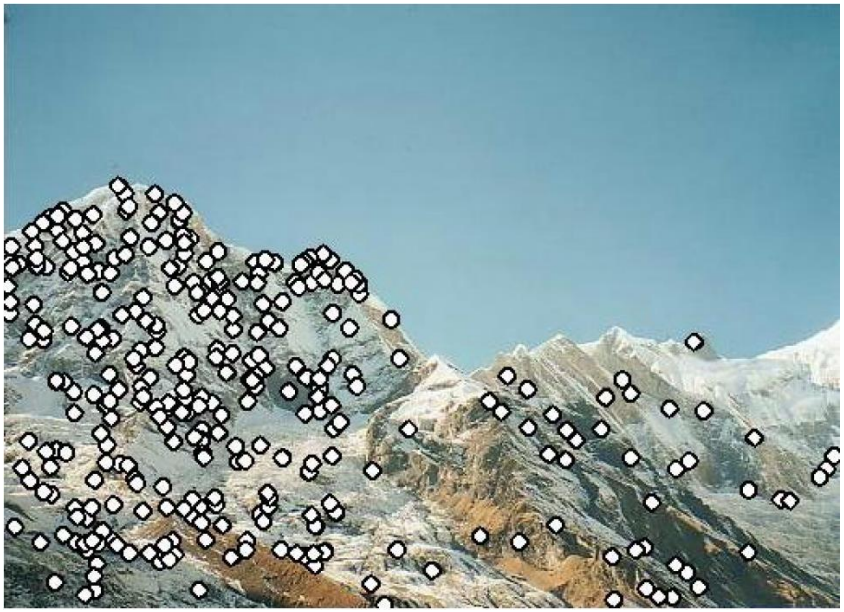
Application: Panorama

- Application: Image Stitching

Application: Panorama

- Application: Image Stitching



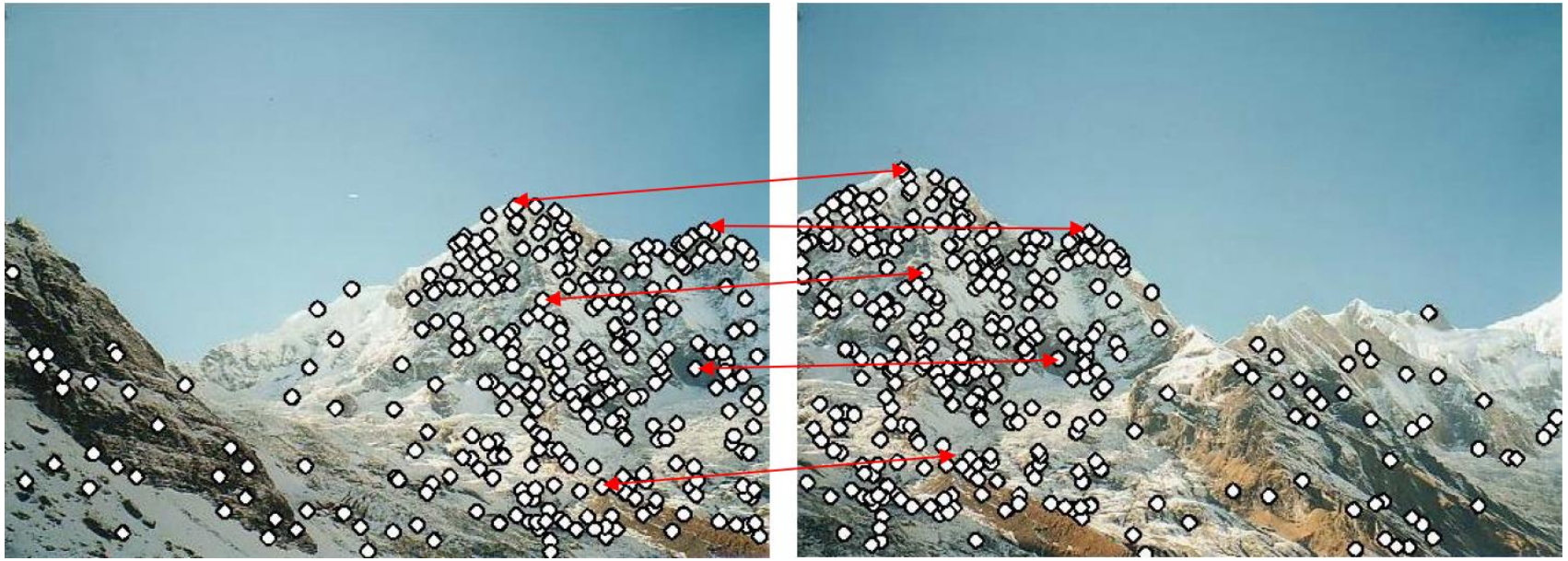


Procedure:

- Detect feature points in both images

Application: Panorama

- Application: Image Stitching



Procedure:

- Detect feature points in both images
- Find corresponding pairs

Application: Panorama

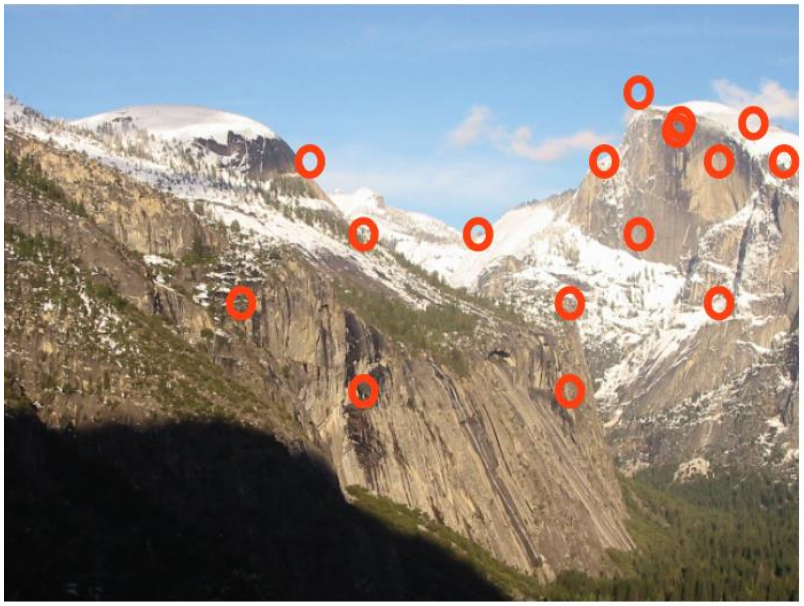
- Application: Image Stitching

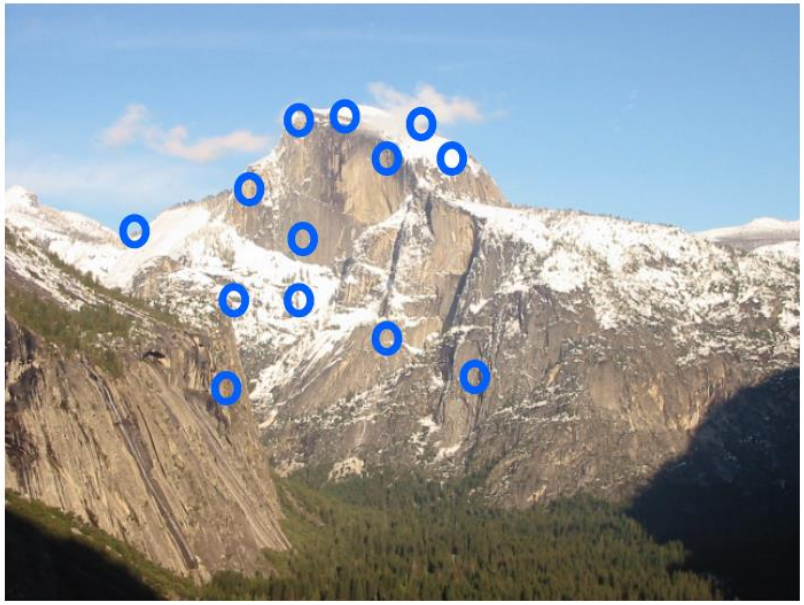
Procedure:

- Detect feature points in both images
- Find corresponding pairs
- Use these pairs to align the images

Application: Panorama

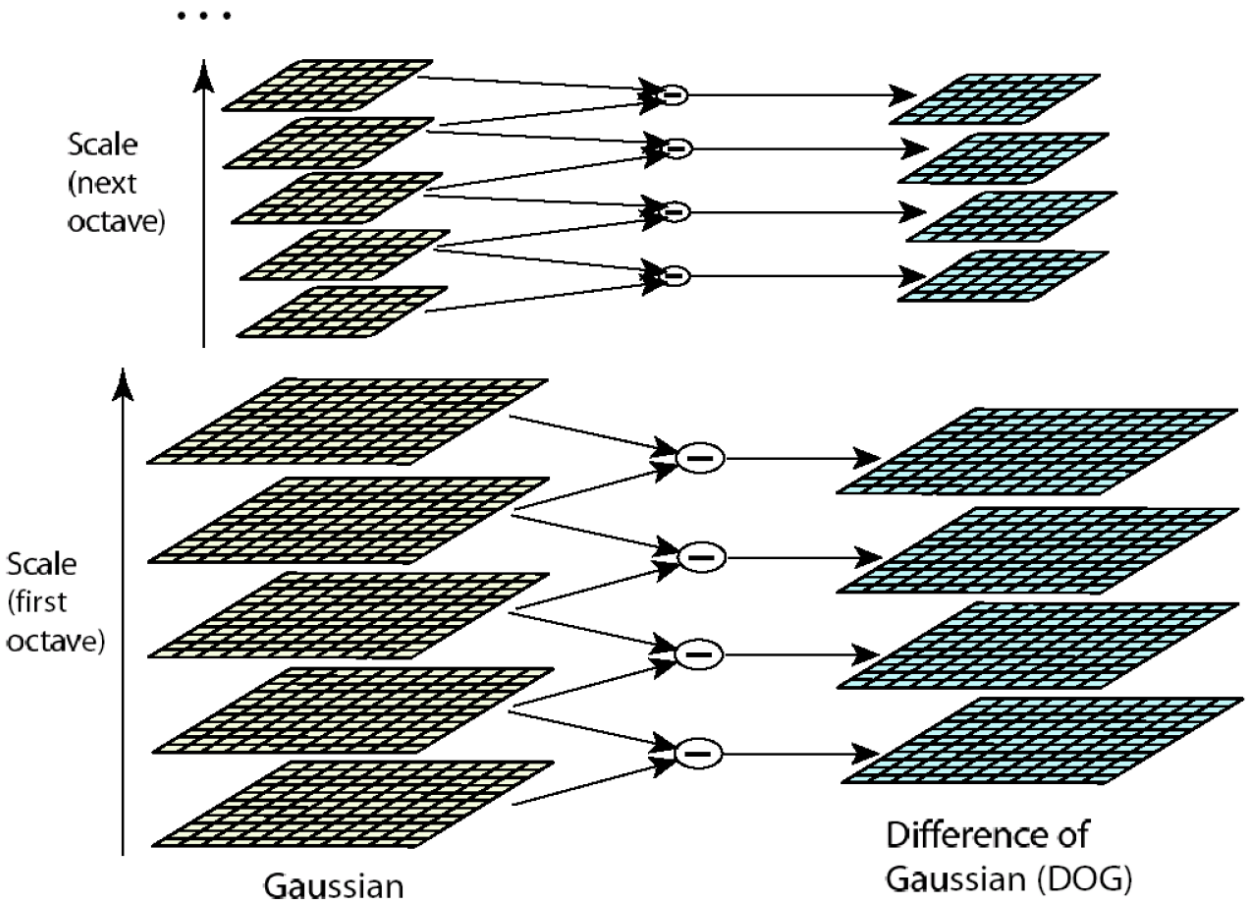
- Main Flow
  - Detect key points





Application: Panorama

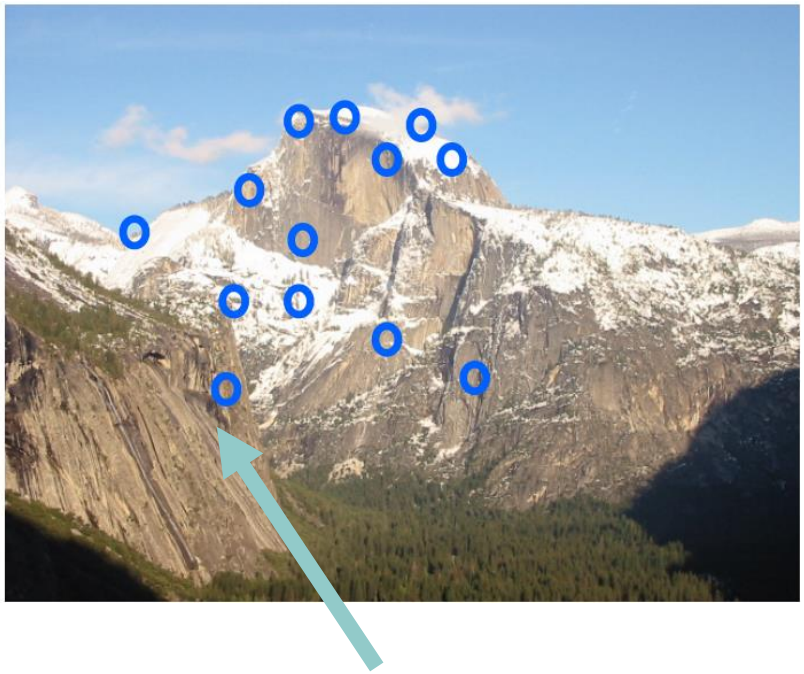
- Detect Key Points



Application: Panorama

- Main Flow
  - Detect key points
  - Build the SIFT descriptors

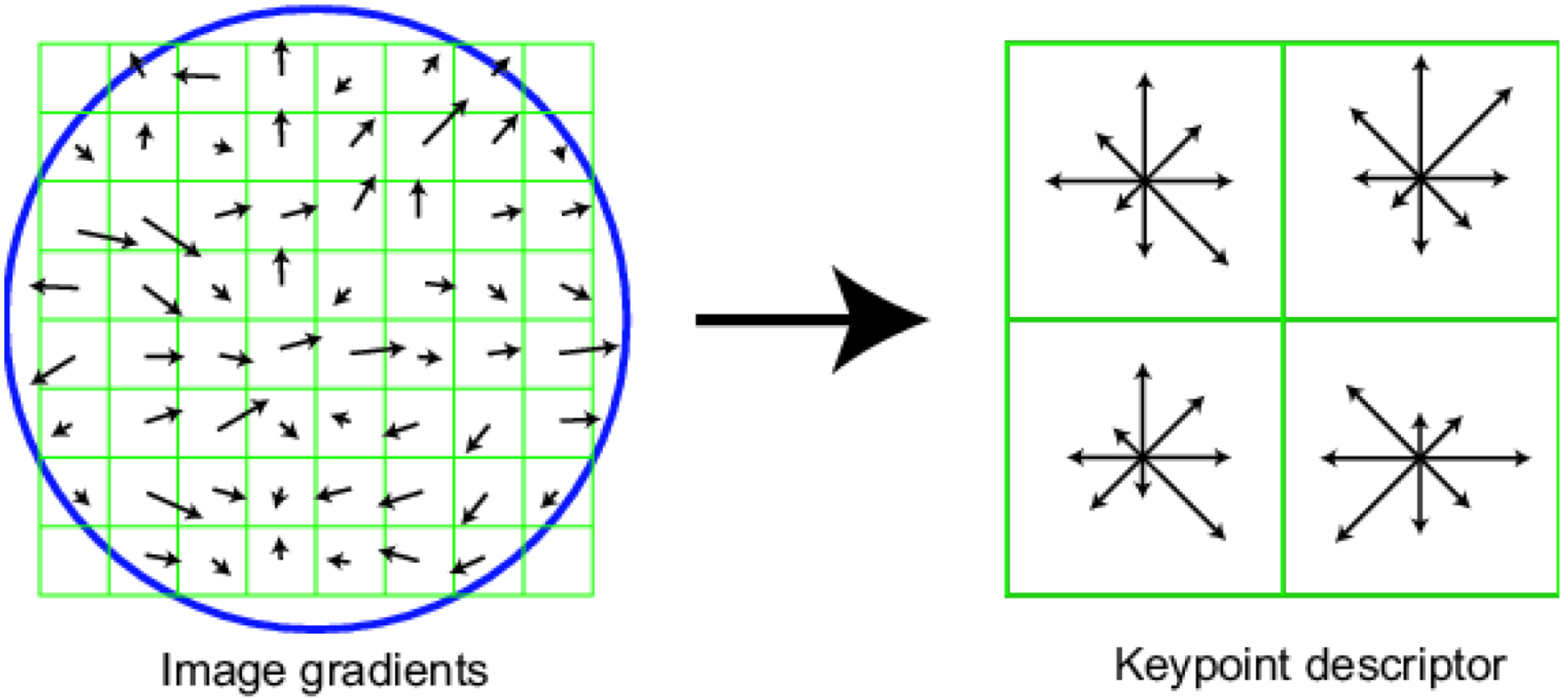
$(u_1, u_2, \ldots, u_{128})$



$(v_1, v_2, \ldots, v_{128})$

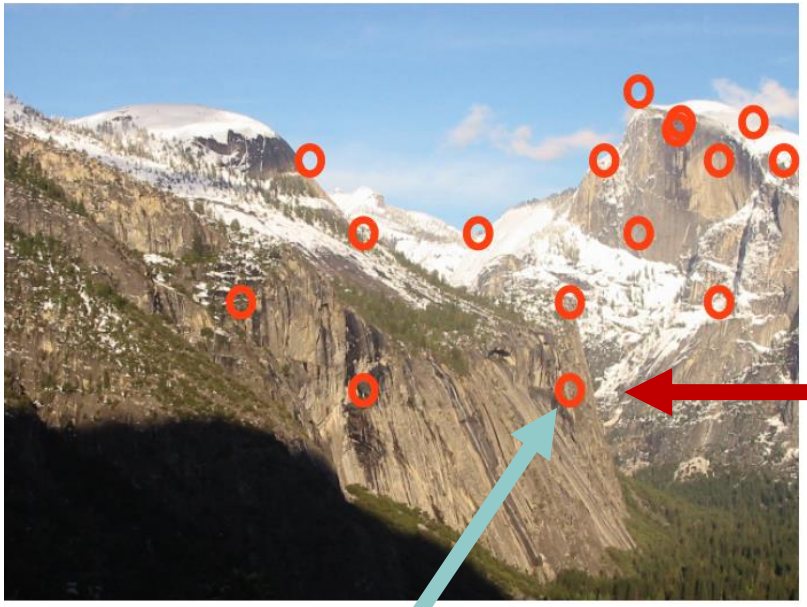
Application: Panorama

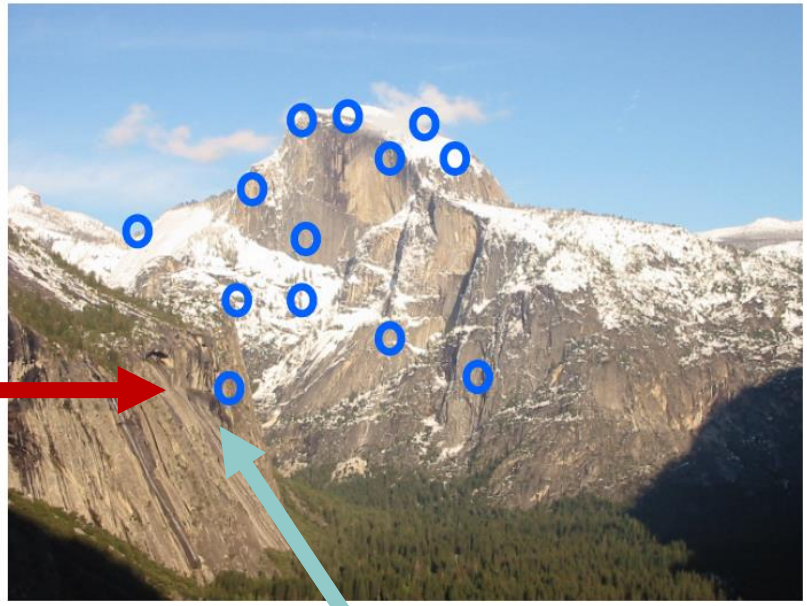
- Build the SIFT Descriptors



Application: Panorama

- Main Flow
  - Detect key points
  - Build the SIFT descriptors
  - Match SIFT descriptors



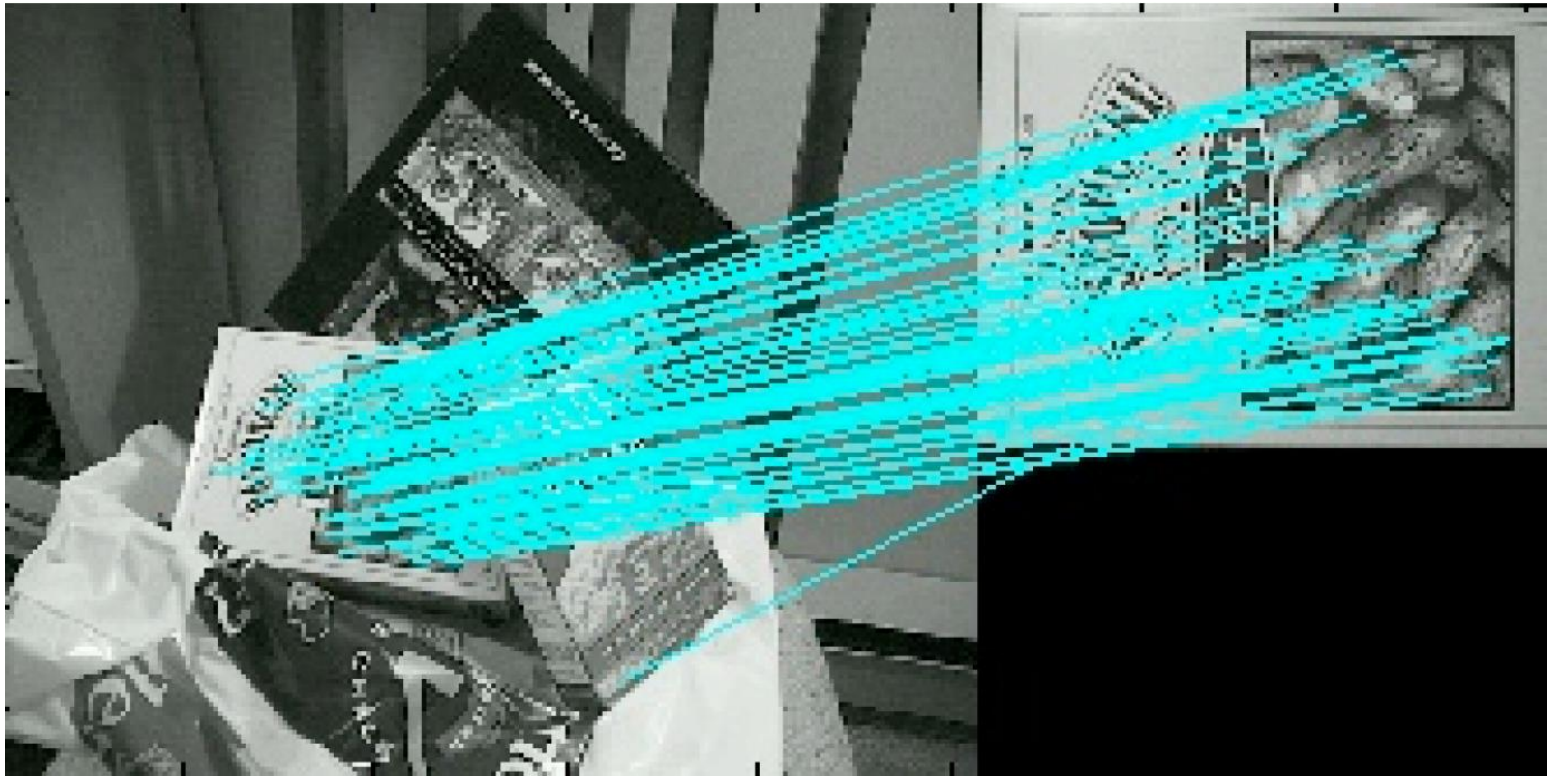


$(u_1, u_2, \ldots, u_{128})$

$(v_1, v_2, \ldots, v_{128})$

Application: Panorama

- Match SIFT Descriptors
  - Euclidean distance between descriptors



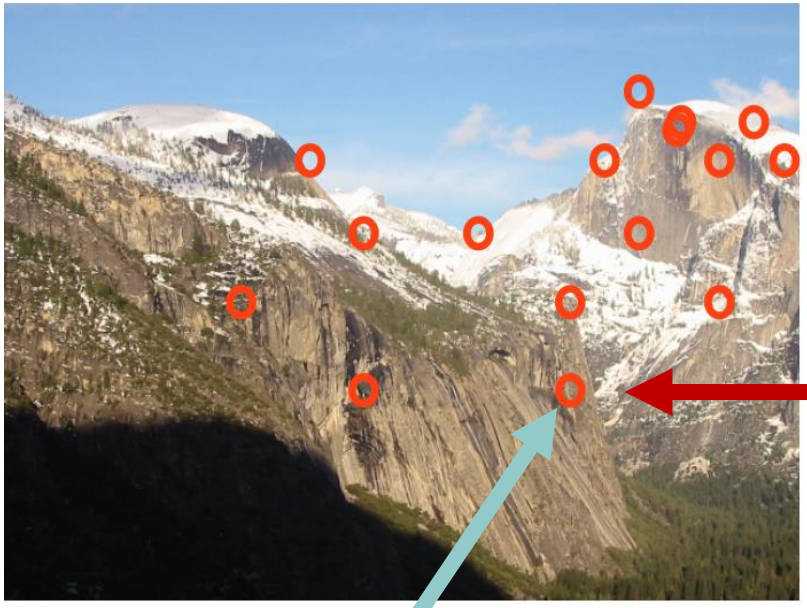
Application: Panorama

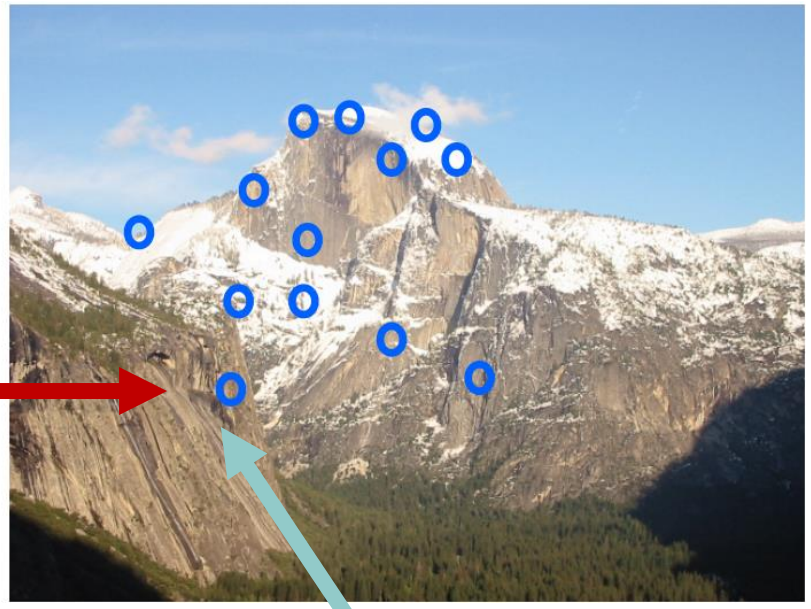
- Skeleton Code
  - Match SIFT descriptors (6 lines of code)
    - Input: D1, D2, thresh (default 0.7)
    - Output: match [D1's index, D2's index]
    - Try to use one for loop

Application: Panorama

- Main Flow
  - Detect key points
  - Build the SIFT descriptors
  - Match SIFT descriptors
  - Fitting the transformation

$$T = \begin{bmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ 0 & 0 & 1 \end{bmatrix}$$



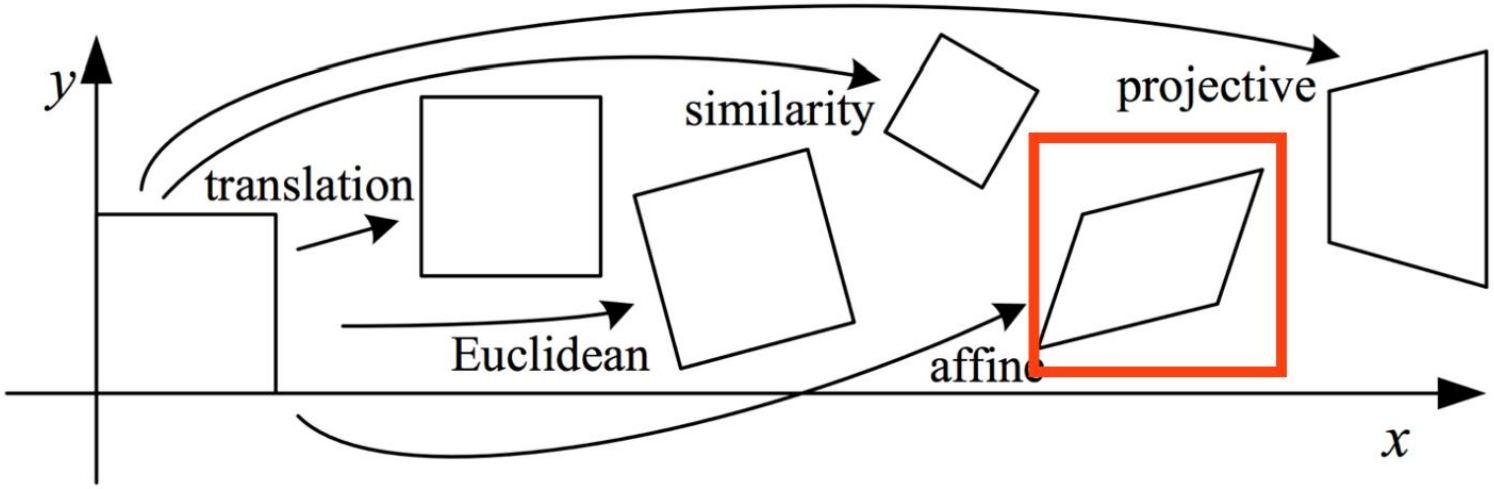


$(u_1, u_2, \ldots, u_{128})$

$(v_1, v_2, \ldots, v_{128})$

Application: Panorama

- Fitting the transformation
  - 2D transformations



Application: Panorama

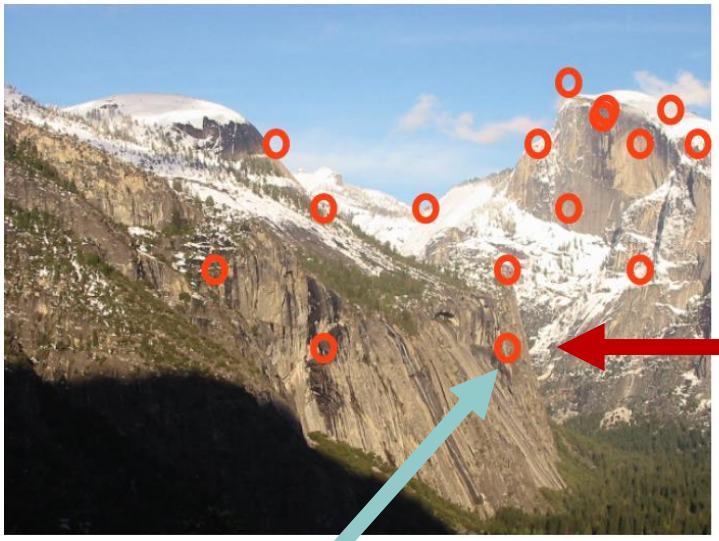
- Skeleton Code
  - Fit the transformation matrix

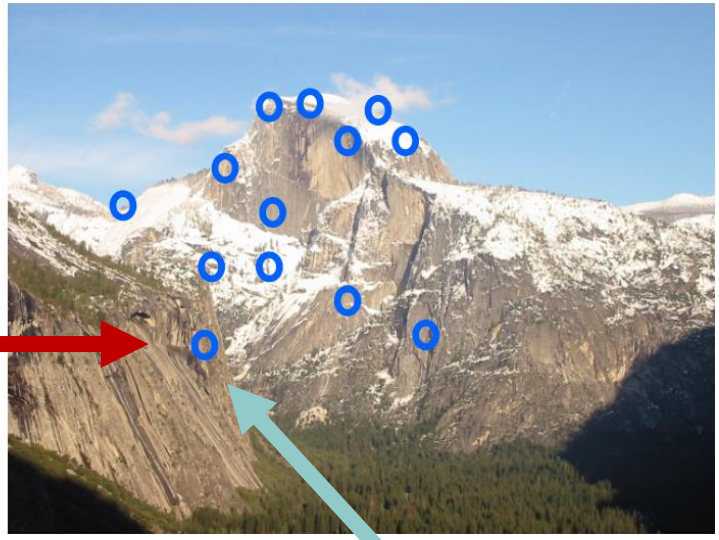
$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ 0 & 0 & 1 \end{bmatrix}$$

  - Six variables
    - each point give two equations
    - at least three points
  - Least squares

Application: Panorama

- Main Flow
  - Detect key points
  - Build the SIFT descriptors
  - Match SIFT descriptors
  - Fitting the transformation
  - RANSAC



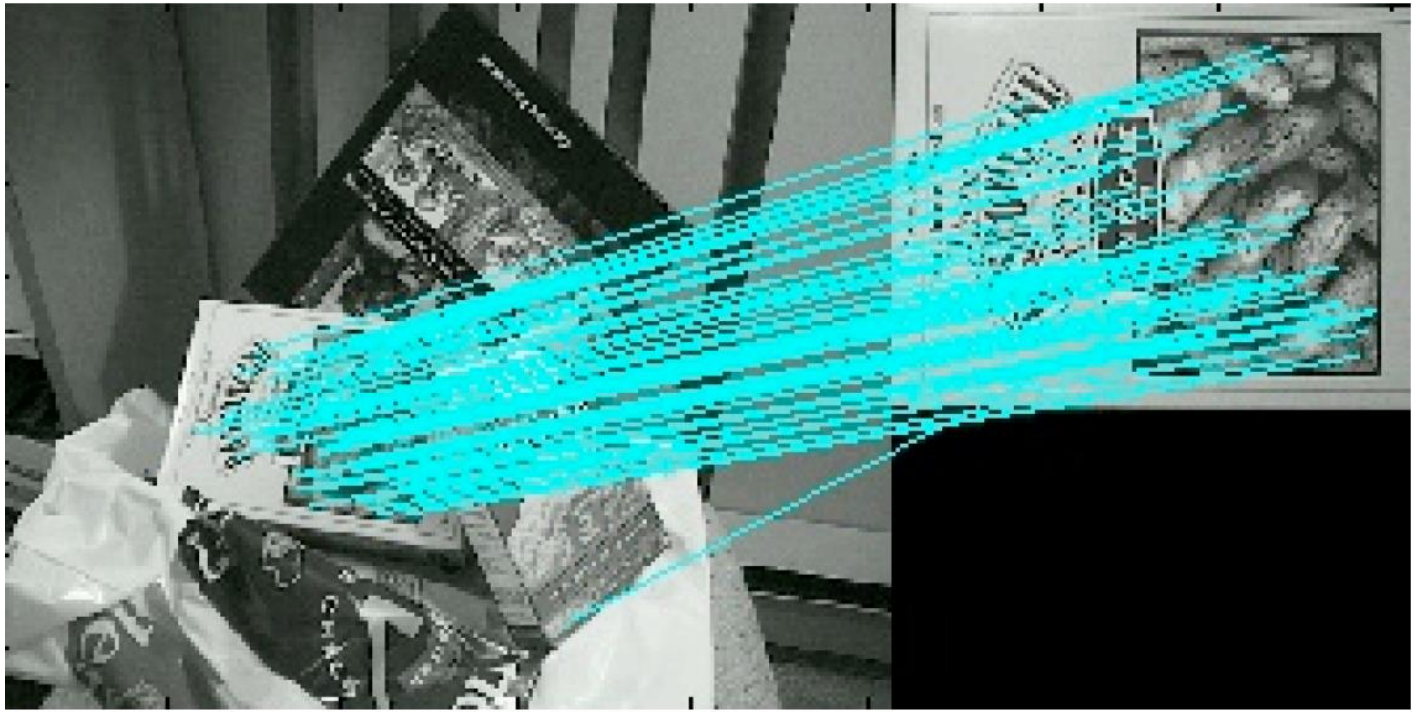


$(u_1, u_2, \ldots, u_{128})$

$(v_1, v_2, \ldots, v_{128})$

Application: Panorama

- RANSAC
  - A further refinement of matches



RANSAC: Algorithm

Algorithm 15.4: RANSAC: fitting lines using random sample consensus

Determine:
    n — the smallest number of points required
    k — the number of iterations required
    t — the threshold used to identify a point that fits well
    d — the number of nearby points required
      to assert a model fits well
Until k iterations have occurred
    Draw a sample of n points from the data
      uniformly and at random
    Fit to that set of n points
    For each data point outside the sample
      Test the distance from the point to the line
        against t; if the distance from the point to the line
        is less than t, the point is close
    end
    If there are d or more points close to the line
      then there is a good fit. Refit the line using all
      these points.
end
Use the best fit from this collection, using the
  fitting error as a criterion

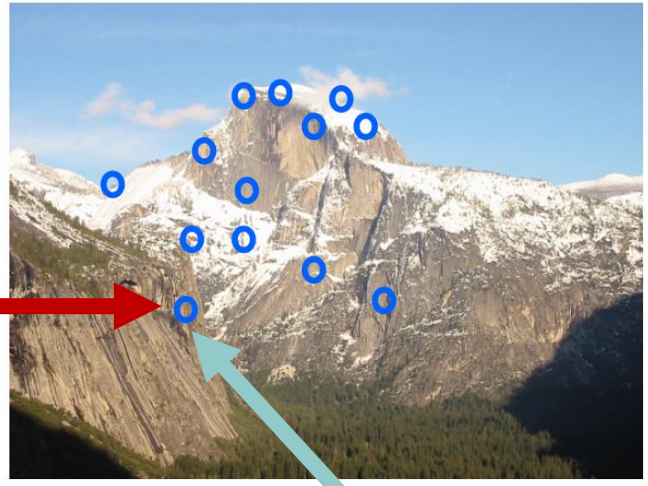
Application: Panorama

- Skeleton Code
  - RANSAC
    - Compute Error

$$\left\| \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} - H \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} \right\|_2$$

Application: Panorama

- Main Flow

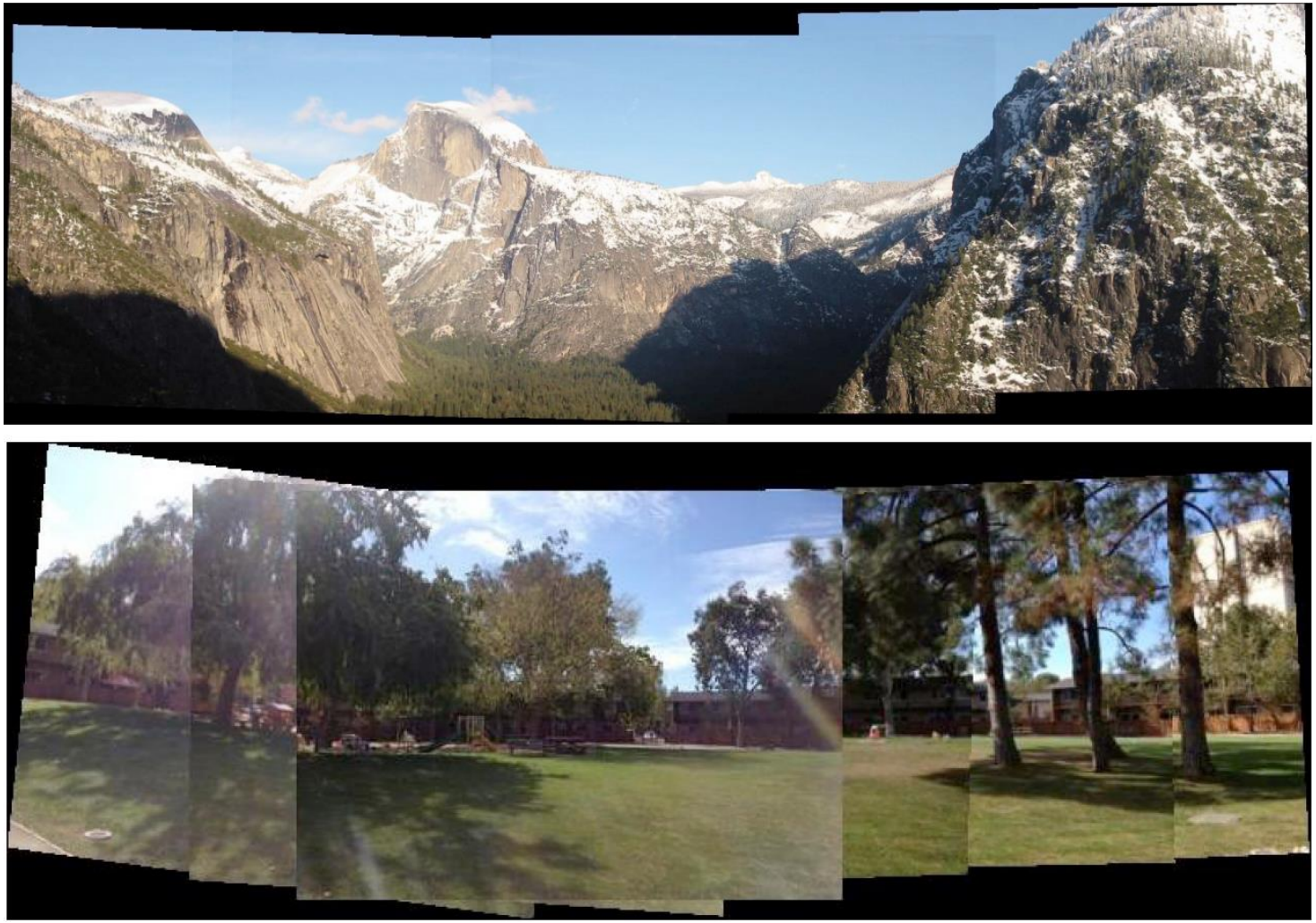


$(u_1, u_2, \ldots, u_{128})$

$(v_1, v_2, \ldots, v_{128})$

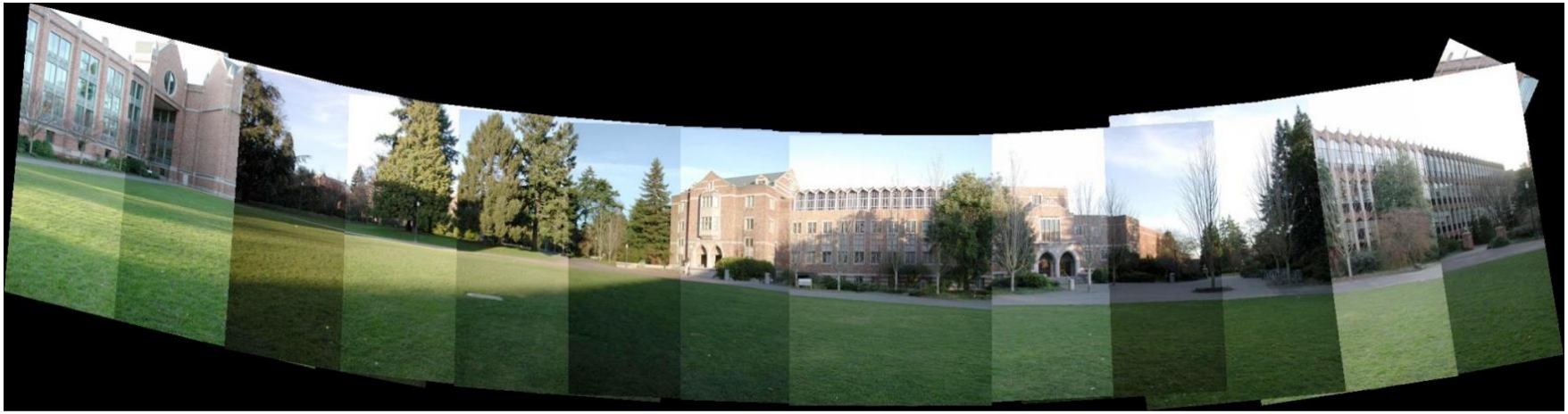
Application: Panorama

- Result



Application: Panorama

- Results



What we have learned today?

- Scale invariant region selection
  - Automatic scale selection
  - Difference-of-Gaussian (DoG) detector
- SIFT: an image region descriptor
- Application: Panorama