University of Zurich[UZH] | ETH zürich

Institute of Informatics – Institute of Neuroinformatics

ROBOTICS & PERCEPTION GROUP

Lecture 06
Point Feature Detection and Matching
Part 2

Davide Scaramuzza

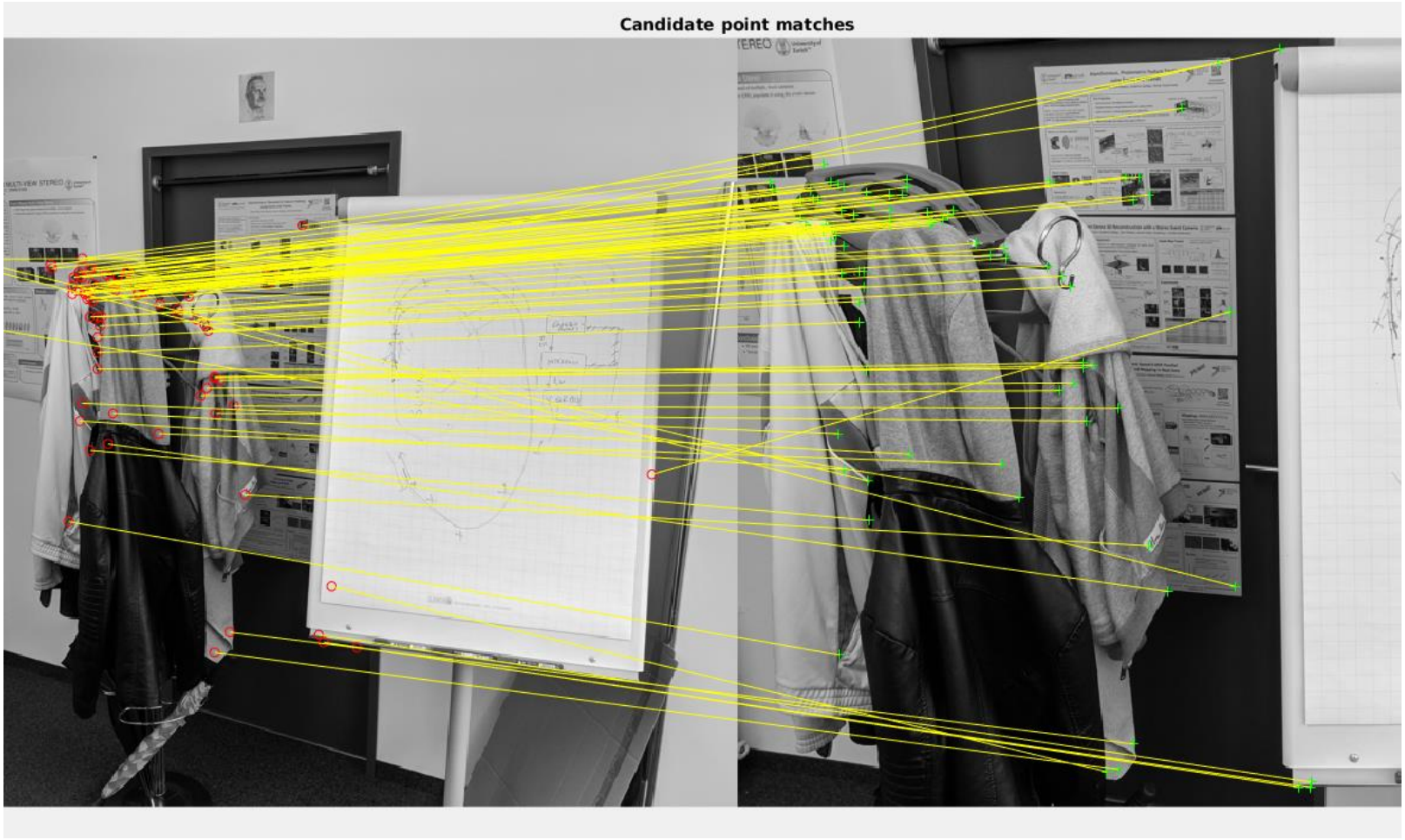
http://rpg.ifi.uzh.ch

Course Schedule updated: Exercise this afternoon

Date	Time	Description of the lecture/exercise	Lecturer
20.09.2018	10:15 - 12:00	01 – Introduction	Davide Scaramuzza
27.09.2018	10:15 - 12:00	02 - Image Formation 1: perspective projection and camera models	Davide Scaramuzza
	13:15 – 15:00	Exercise 1: Augmented reality wireframe cube	Titus Cieslewski & Mathias Gehrig
04.10.2018	10:15 - 12:00	03 - Image Formation 2: camera calibration algorithms	Guillermo Gallego
	13:15 – 15:00	Exercise 2: PnP problem	Antonio Loquercio & Mathias Gehrig
11.10.2018	10:15 - 12:00	04 - Filtering & Edge detection	Davide Scaramuzza
18.10.2018	10:15 - 12:00	05 - Point Feature Detectors 1: Harris detector	Guillermo Gallego
	13:15 – 15:00	Exercise 3: Harris detector + descriptor + matching	Antonio Loquercio & Mathias Gehrig
25.10.2018	10:15 - 12:00	06 - Point Feature Detectors 2: SIFT, BRIEF, BRISK	Davide Scaramuzza
	13:15 – 15:00	Exercise 4: SIFT detector + descriptor + matching	Antonio Loquercio & Mathias Gehrig
01.11.2018	10:15 - 12:00	07 - Multiple-view geometry 1	Guillermo Gallego
	13:15 – 15:00	Exercise 5: Stereo vision: rectification, epipolar matching, disparity, triangulation	Antonio Loquercio & Mathias Gehrig
08.11.2018	10:15 - 12:00	08 - Multiple-view geometry 2	Davide Scaramuzza
	13:15 – 15:00	Exercise 6: Eight-Point algorithm	Antonio Loquercio & Mathias Gehrig
15.11.2018	10:15 - 12:00	09 - Multiple-view geometry 3	Davide Scaramuzza
	13:15 – 15:00	Exercise 7: P3P algorithm and RANSAC	Antonio Loquercio & Mathias Gehrig
22.11.2018	10:15 - 12:00	10 - Dense 3D Reconstruction (Multi-view Stereo)	Davide Scaramuzza
	13:15 – 15:00	Exercise session: Intermediate VO Integration	Antonio Loquercio & Mathias Gehrig
29.11.2018	10:15 - 12:00	11 - Optical Flow and Tracking (Lucas-Kanade)	Davide Scaramuzza
	13:15 – 15:00	Exercise 8: Lucas-Kanade tracker	Antonio Loquercio & Mathias Gehrig
06.12.2018	10:15 - 12:00	12 – Place recognition	Davide Scaramuzza
	13:15 – 15:00	Exercise session: Deep Learning Tutorial	Antonio Loquercio
	10:15 - 12:00	13 – Visual inertial fusion	Davide Scaramuzza
13.12.2018	13:15 – 15:00	Exercise 9: Bundle adjustment	Antonio Loquercio & Mathias Gehrig
20.12.2018	10:15 - 12:00	14 - Event based vision	Davide Scaramuzza
	12:30 – 13:30	**Scaramuzza's lab visit and live demonstrations**: Andreasstrasse 15, 2.11, 8050	Davide Scaramuzza & his lab
	14:00 – 16:00	Exercise session: final VO integration (it will take place close to Scaramuzza's lab)	Antonio Loquercio & Mathias Gehrig

Lab Exercise 4 - Today afternoon

- Room ETH HG E 1.1 from 13:15 to 15:00
- Work description: implement the SIFT blob detector and tracker

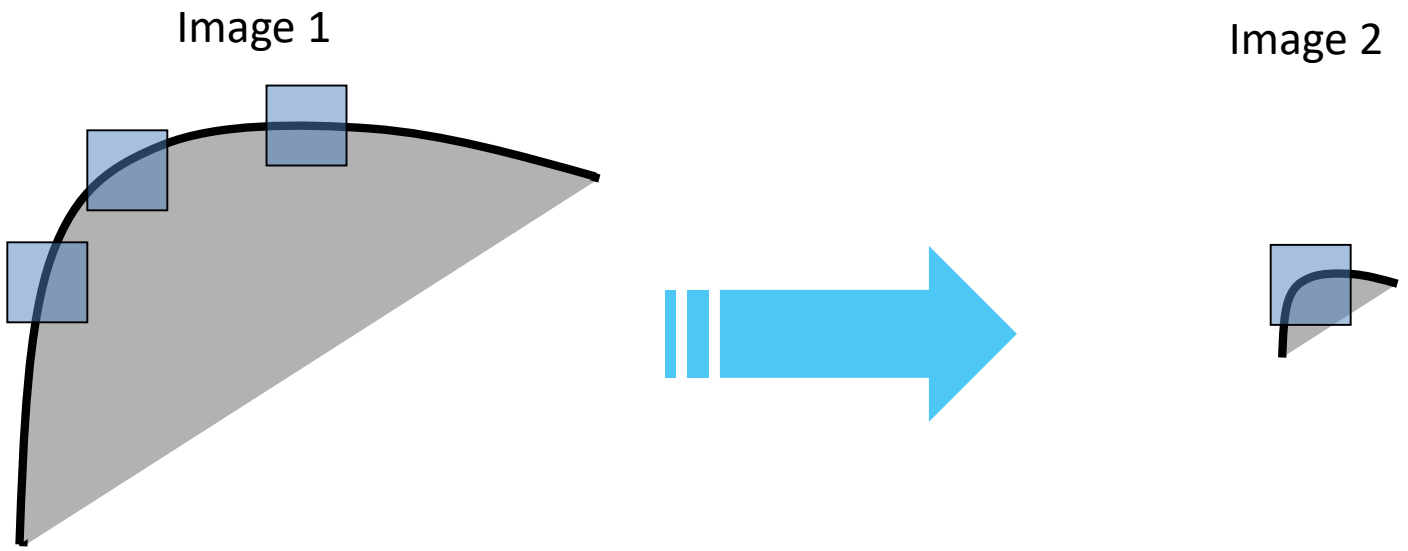


Outline

- Automatic Scale Selection
- The SIFT blob detector and descriptor
- Other corner and blob detectors and descriptors

Scale changes

- How can we match image patches corresponding to the same feature but belonging to images taken at different scales?

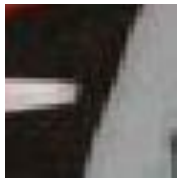


Scale changes

- How can we match image patches corresponding to the same feature but belonging to images taken at different scales?
  - Possible solution: rescale the patch

Image 1

Image 2

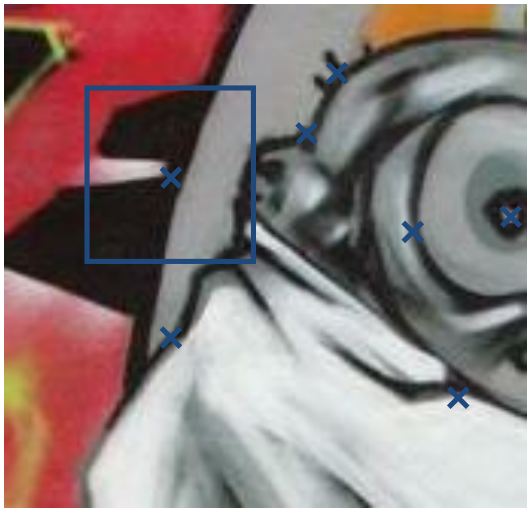


Scale changes

- How can we match image patches corresponding to the same feature but belonging to images taken at different scales?
  - Possible solution: rescale the patch

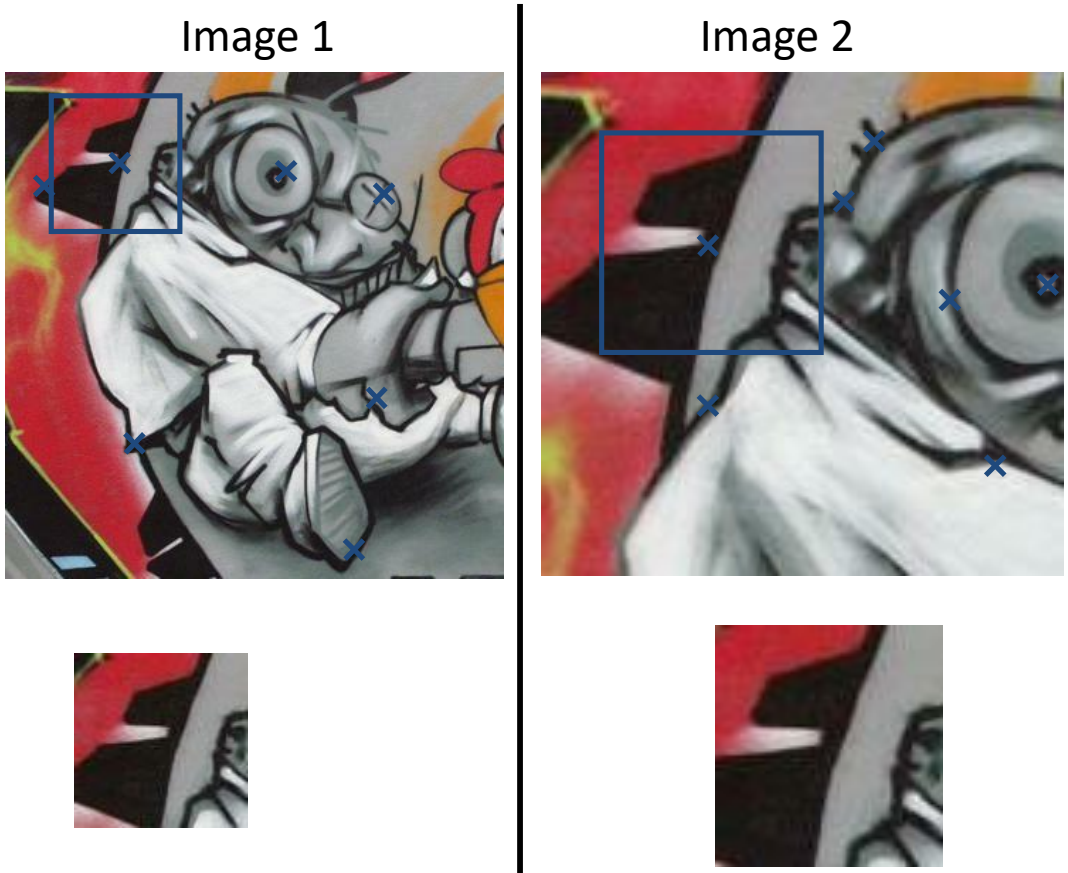
Image 1

Image 2



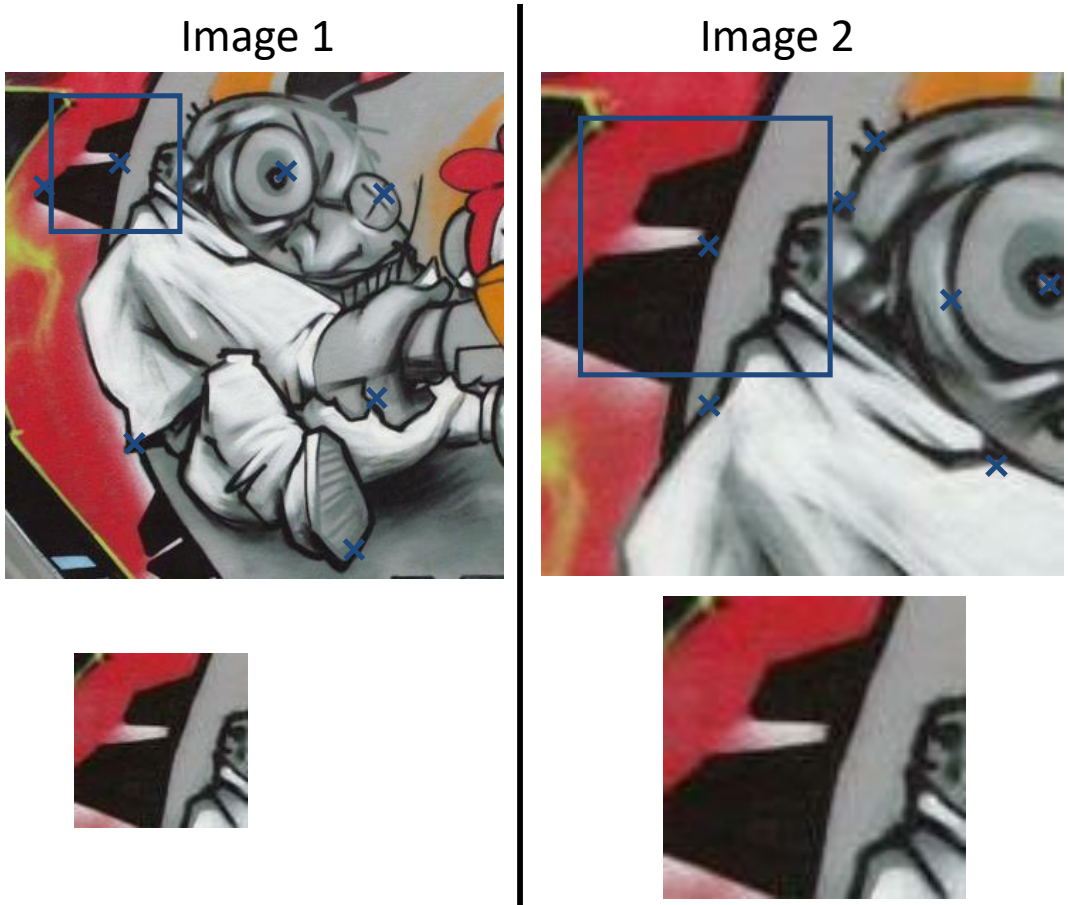
Scale changes

- How can we match image patches corresponding to the same feature but belonging to images taken at different scales?
  - Possible solution: rescale the patch



Scale changes

- How can we match image patches corresponding to the same feature but belonging to images taken at different scales?
  - Possible solution: rescale the patch

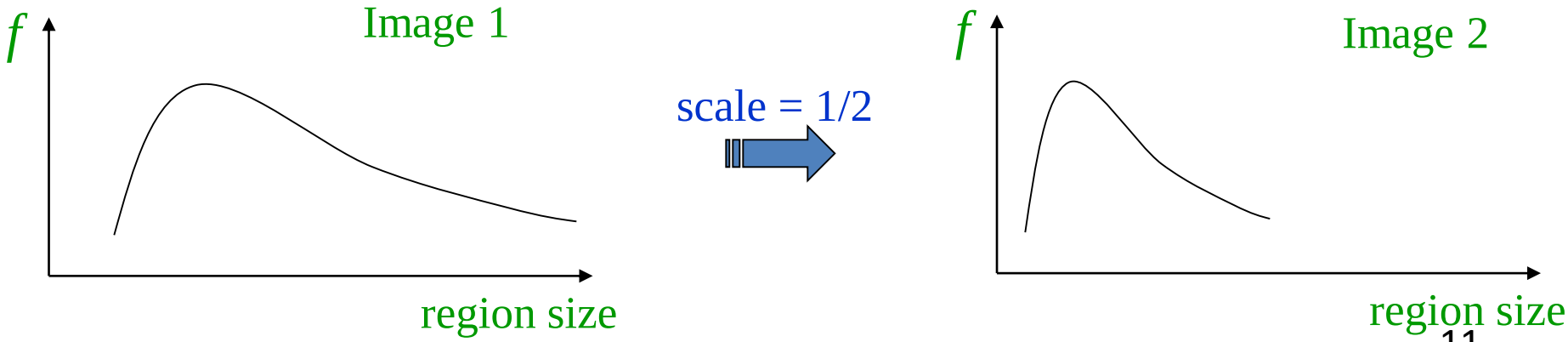


Scale changes

- Scale search is time consuming (needs to be done individually for all patches in one image)
  - **Complexity** would be $(\boldsymbol{NS})^{\mathbf{2}}$ (assuming that we have N features per image and S scale levels for each image)
- Possible **solution**: **assign each feature its own "scale"** (i.e., size).
  - What's the optimal scale (i.e., size) of the patch?

Automatic Scale Selection

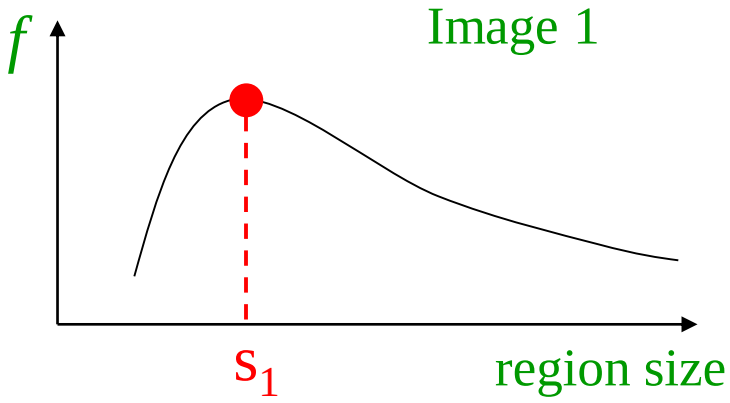
- Solution:
  - Design a function on the image patch, which is "scale invariant" (i.e., which has the same value for corresponding regions, even if they are at different scales)

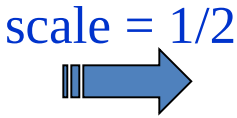


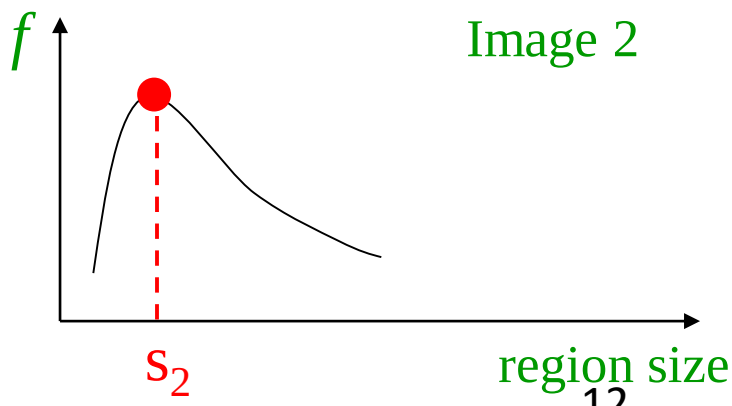
Automatic Scale Selection

- Approach:
  - Take a local maximum or minimum of this function
  - Region size for which the maximum or minimum is achieved should be *invariant* to image scale.

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



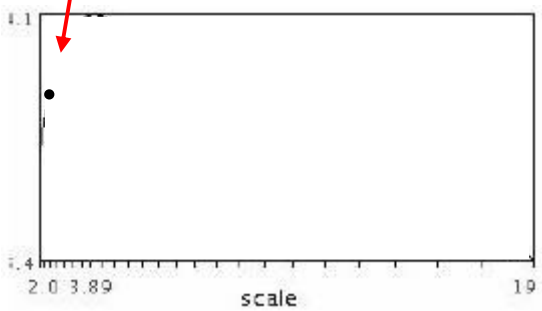




Automatic Scale Selection

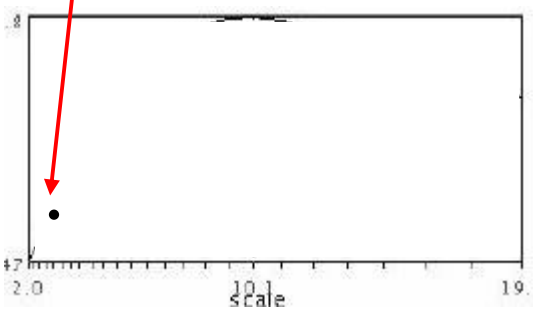
- Function responses for increasing scale (scale signature)

Image 1



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

Image 2

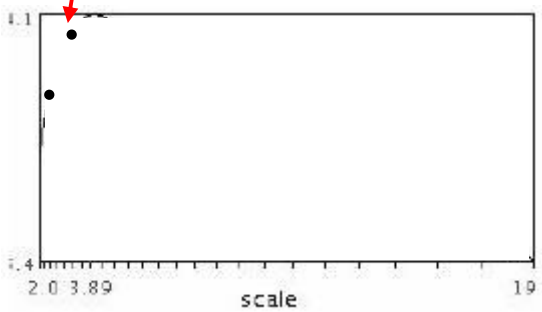


$f(I_{i_1 \ldots i_m}(x',\sigma))$

Automatic Scale Selection

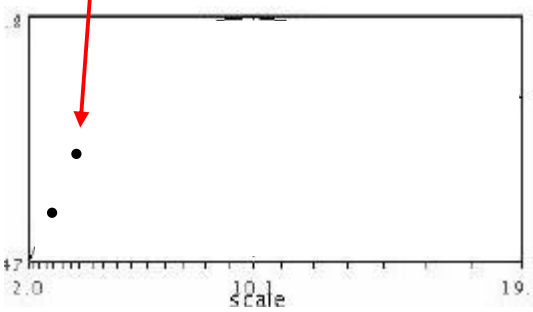
- Function responses for increasing scale (scale signature)

Image 1



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

Image 2

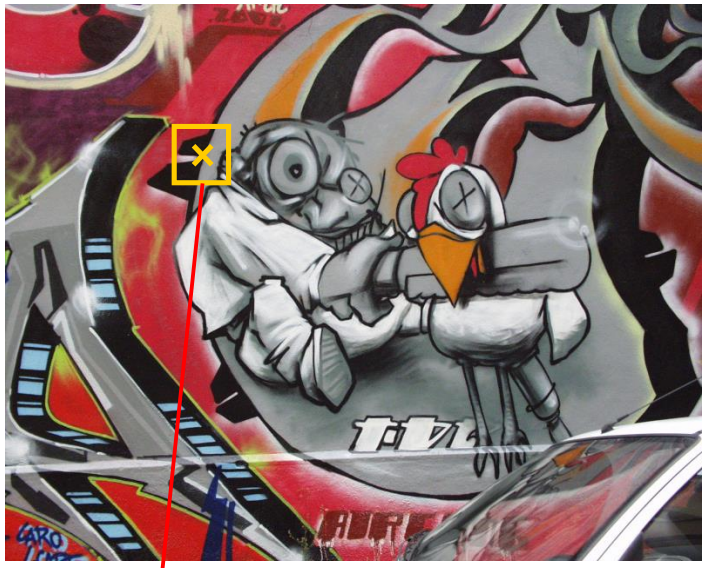


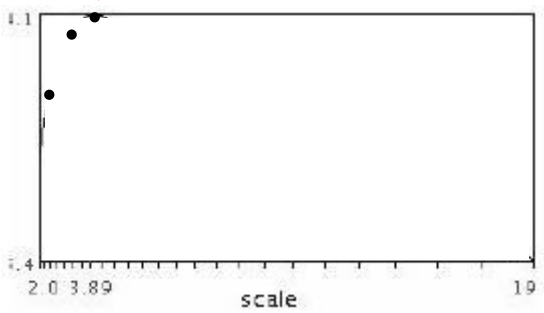
$f(I_{i_1 \ldots i_m}(x',\sigma))$

Automatic Scale Selection

- Function responses for increasing scale (scale signature)

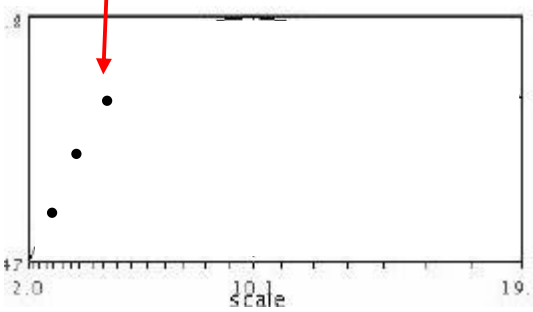
Image 1





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

Image 2

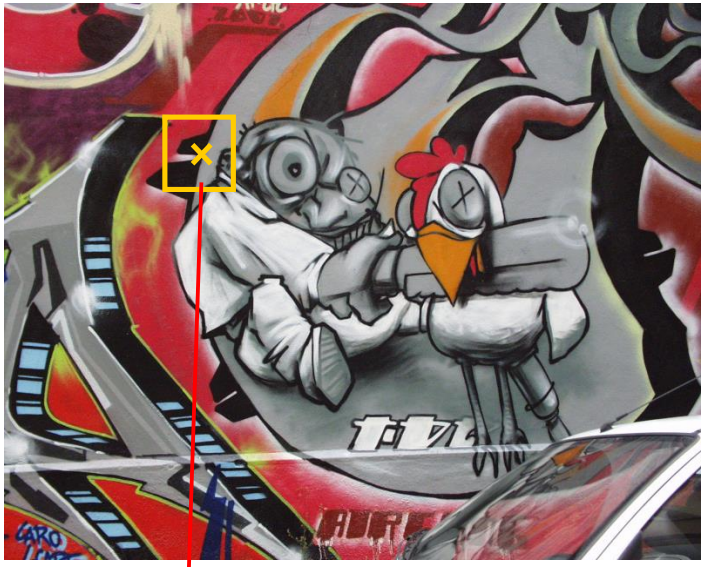


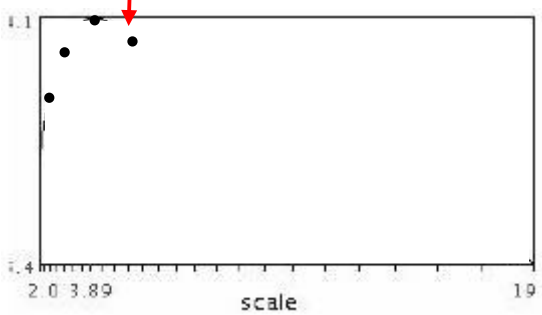
$f(I_{i_1 \ldots i_m}(x',\sigma))$

Automatic Scale Selection

- Function responses for increasing scale (scale signature)

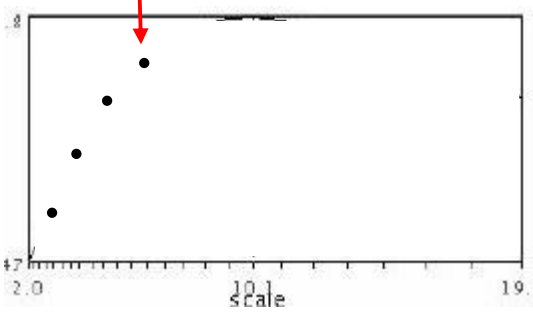
Image 1





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

Image 2

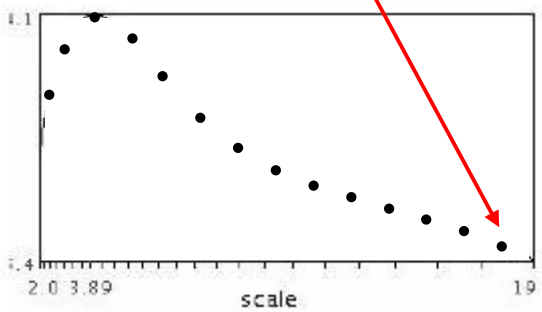


$f(I_{i_1 \ldots i_m}(x',\sigma))$

Automatic Scale Selection

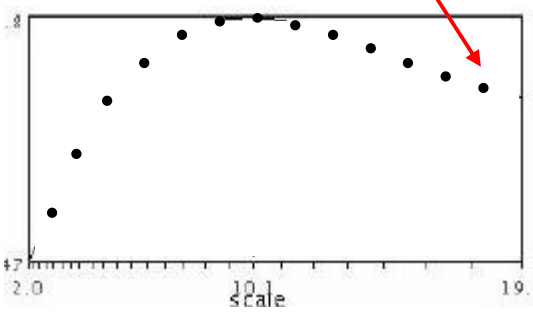
- Function responses for increasing scale (scale signature)

Image 1



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

Image 2

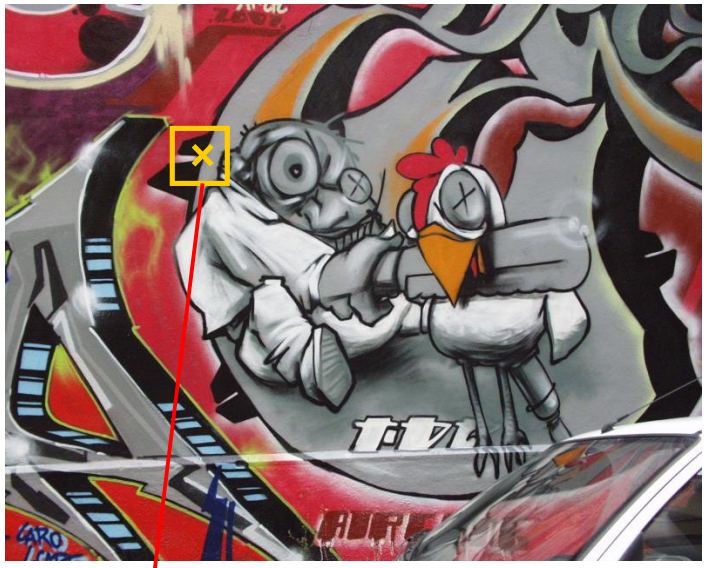


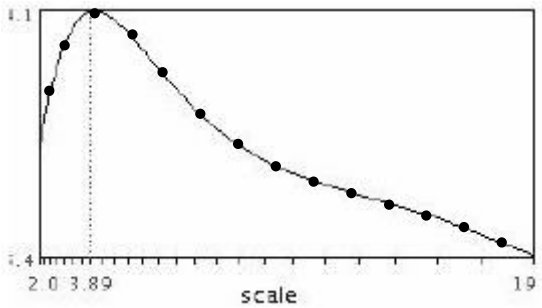
$f(I_{i_1 \ldots i_m}(x',\sigma))$

Automatic Scale Selection

- Function responses for increasing scale (scale signature)

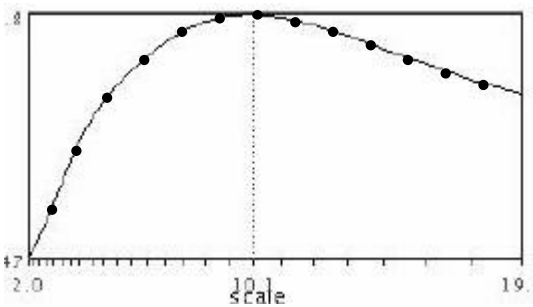
Image 1





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

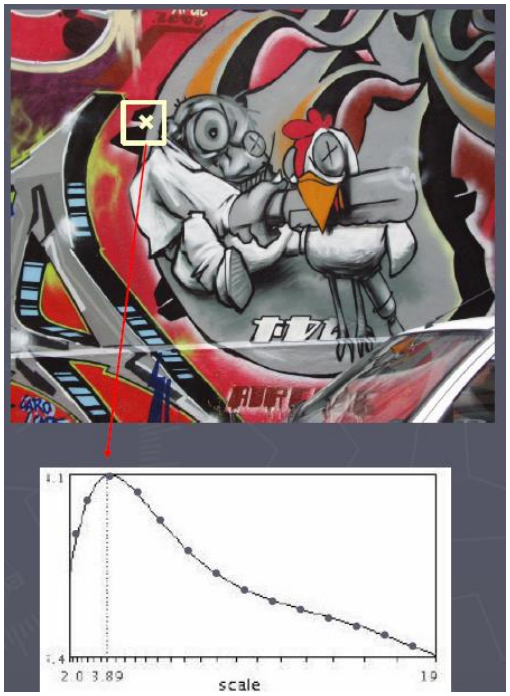
Image 2

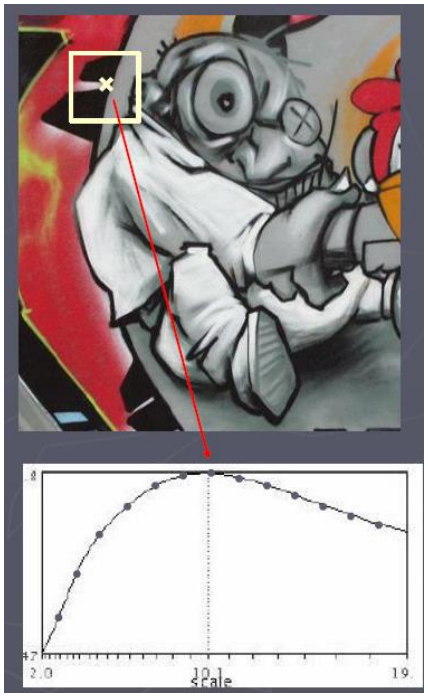


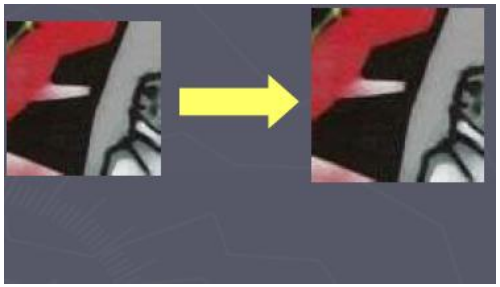
$f(I_{i_1 \ldots i_m}(x',\sigma'))$

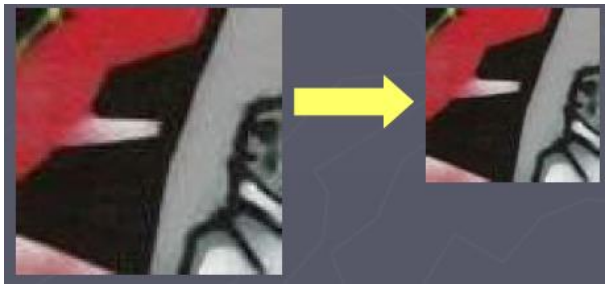
Automatic Scale Selection

- When the right scale is found, the patch must be normalized









Automatic Scale Selection

- A “good” function for scale detection should have a single & sharp peak

- What if there are multiple peaks? Is it really a problem?

- **Sharp, local intensity changes** are good regions to monitor in order to identify the scale

  ⇨ **Blobs and corners** are the **ideal locations**!

Automatic Scale Selection

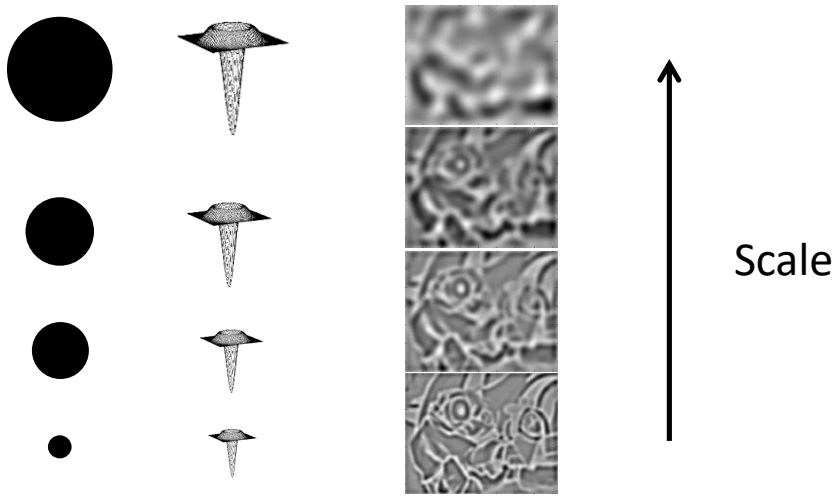
- The **ideal function** for determining the scale **is one that highlights sharp discontinuities**
- **Solution**: convolve image with a **kernel that highlights edges**

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

- It has been shown that the **Laplacian of Gaussian kernel** is optimal under certain assumptions [Lindeberg'94]:

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

- Correct scale is found as local maxima or minima across consecutive smoothed images



Lindeberg, Scale-space theory: A basic tool for analysing structures at different scales, Journal of Applied Statistics, 1994

Automatic Scale Selection

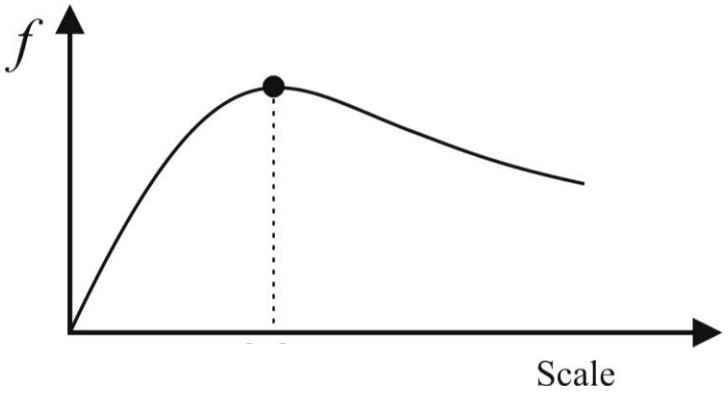
- The **ideal function** for determining the scale **is one that highlights sharp discontinuities**
- **Solution**: convolve image with a **kernel that highlights edges**

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

- It has been shown that the **Laplacian of Gaussian kernel** is optimal under certain assumptions [Lindeberg'94]:

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

- Correct scale is found as local maxima or minima across consecutive smoothed images



Lindeberg, Scale-space theory: A basic tool for analysing structures at different scales, Journal of Applied Statistics, 1994

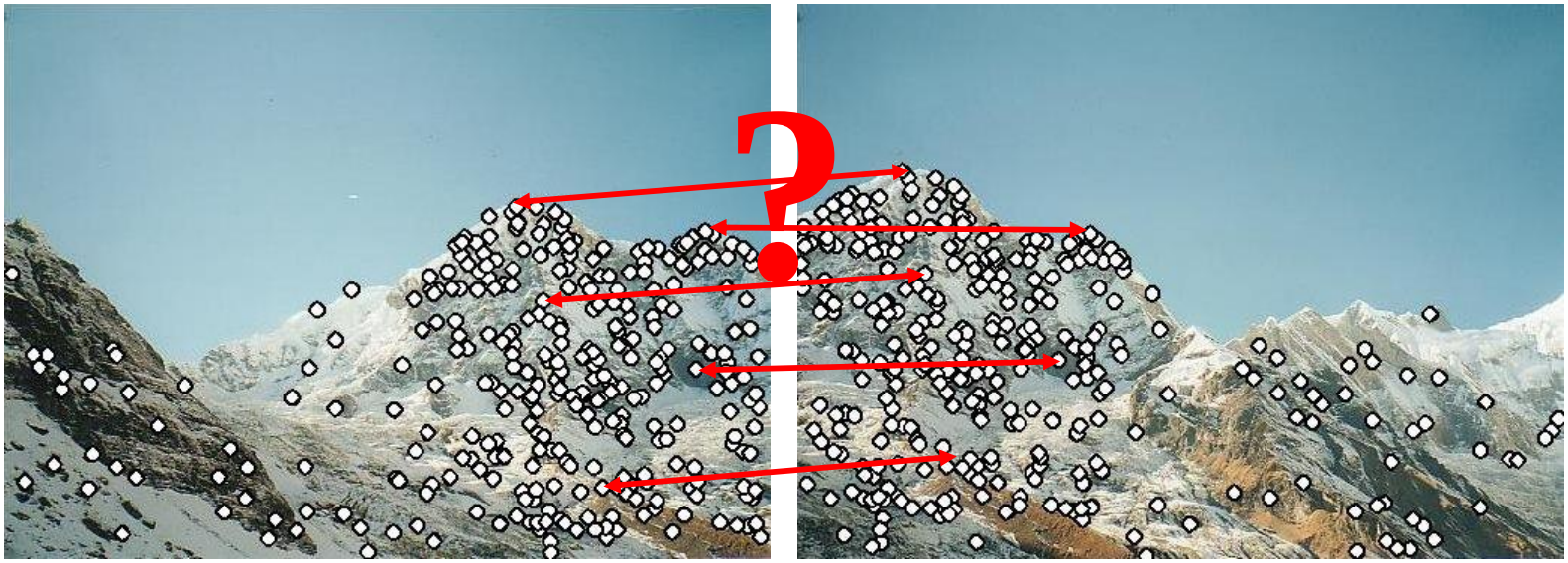
Main questions

- What points are distinctive (i.e., *features*, *keypoints, salient* points), such that they are *repeatable? (i.e.,* can be re-*detected* from other views)
- How to *describe* a local region?
- How to establish *correspondences*, i.e., compute matches?

Feature descriptors

- We know how to detect points
- Next question:

How to *describe* them for matching?



- **Simplest descriptor**: **patch intensity values** (also called *patch descriptor*)
- Alternative: **Census Transform** or **Histograms of Oriented Gradients** (like in SIFT, see later)
- Then, descriptor matching can be done using **Hamming Distance (Census)** or **(Z)SSD, (Z)SAD, or (Z)NCC**

Feature descriptors

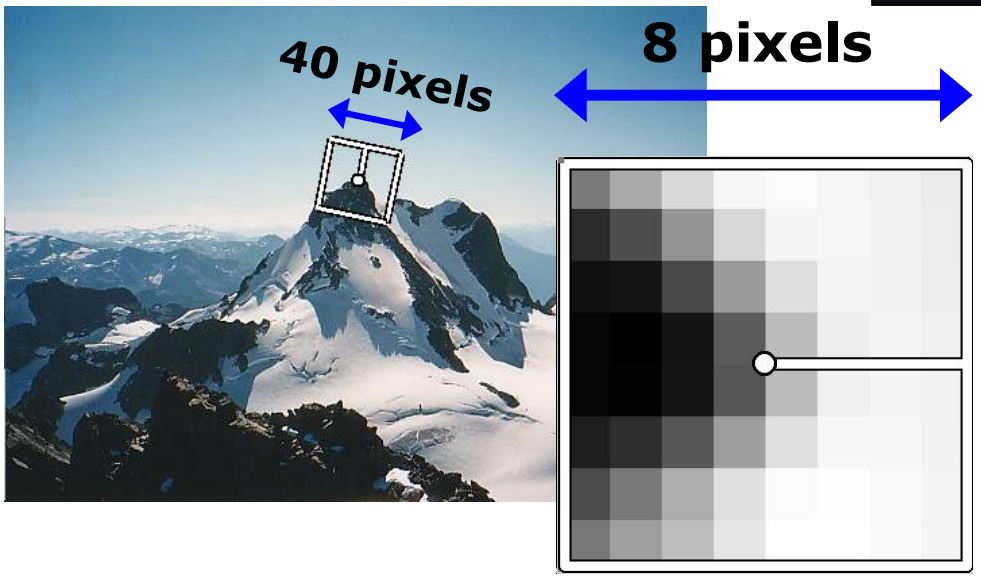
- We'd like to find a feature descriptor that is *invariant* to:
  - **geometric changes:** *rotation, scale, view point*
  - **photometric changes:** *illumination*

- **Most feature methods** are designed to be invariant to
    - 2D translation,
    - 2D rotation,
    - Scale
  - Some of them can also handle
    - **Small view-point invariance** (SIFT, SURF, ORB, BRISK; SIFT works with up to 50 degrees of viewpoint changes!)
    - **Affine illumination changes** (SIFT, SURF, ORB, BRISK)

How to achieve invariance with Patch descriptors

Step 1: Re-scaling and De-rotation

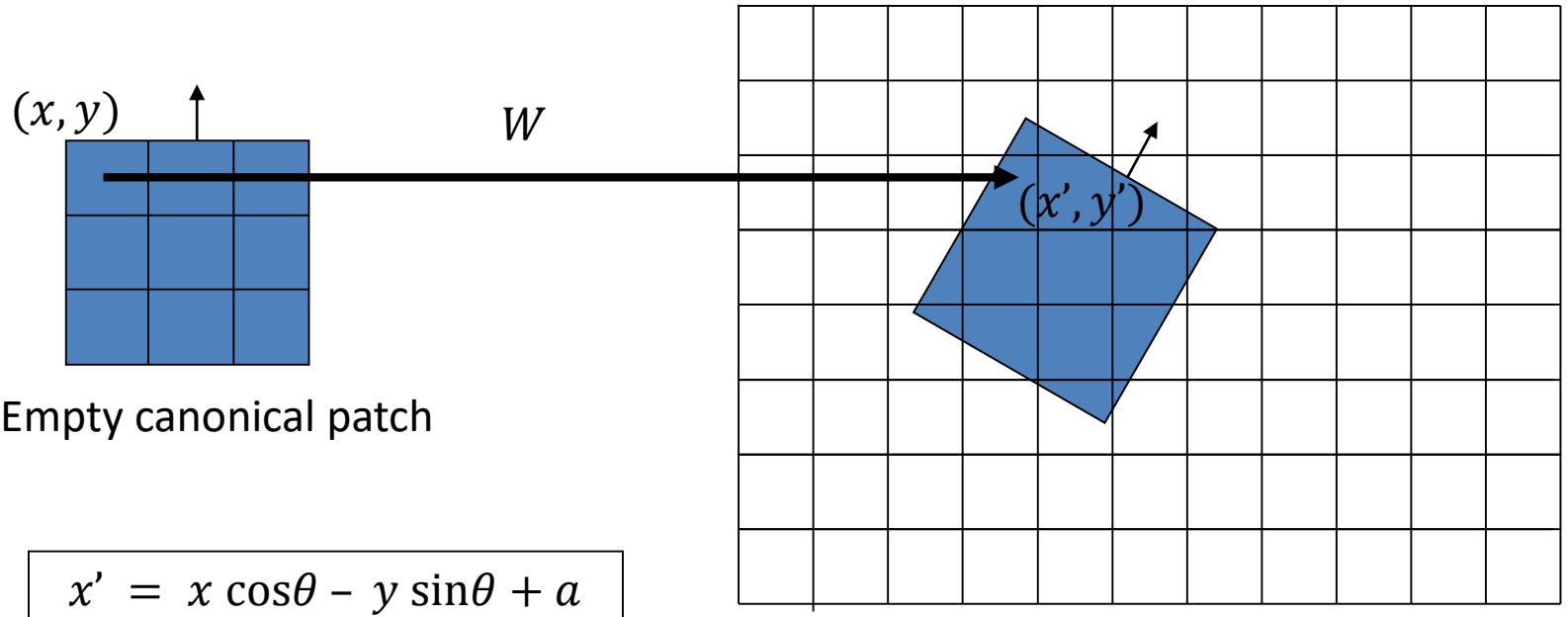
- **Find correct scale** using LoG operator
- **Rescale the patch** to a default size (e.g., 8x8 pixels)
- **Find local orientation**
  - Dominant direction of gradient for the image patch (e.g., Harris eigenvectors)
- **De-rotate patch through "patch warping"**
  - This puts the patches into a **canonical orientation**



How to warp a patch?

- Start with an "**empty**" canonical patch (all pixels set to 0)
- For each pixel (x, y) in the empty patch, apply the **warping function** $\boldsymbol{W(x, y)}$ to compute the corresponding position in the source image. It will be in floating point and will fall between the image pixels.
- **Interpolate** the intensity values of the 4 closest pixels in the detected image. You can use:
  - *Nearest neighbor interpolation*
  - *Bilinear interpolation*
  - *Bicubic interpolation*

Example 1: Roto-Translational warping



Empty canonical patch

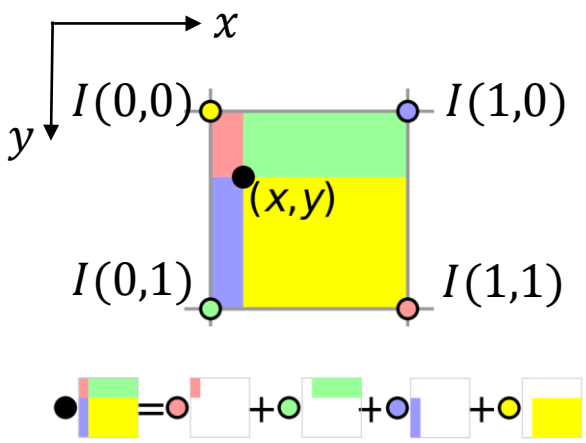
Patch detected in the image

$$W \quad \boxed{\begin{aligned} x' &= x\cos\theta - y\sin\theta + a \\ y' &= x\sin\theta + y\cos\theta + b \end{aligned}}$$

counterclockwise rotation plus translation

Bilinear Interpolation

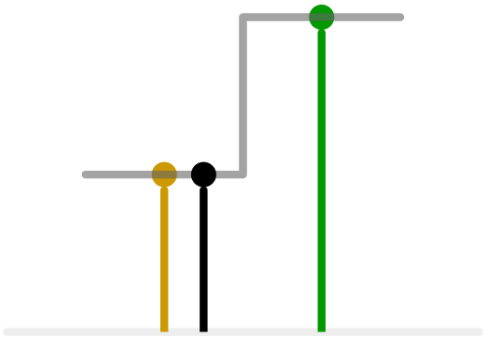
- It is an ***extension* of *linear interpolation*** for interpolating functions of two variables (e.g., x and y) on a *rectilinear 2D grid*.
- The key idea is to perform linear interpolation first in one direction, and then again in the other direction. Although each step is linear in the sampled values and in the position, the interpolation as a whole is not linear but rather quadratic in the sample location.



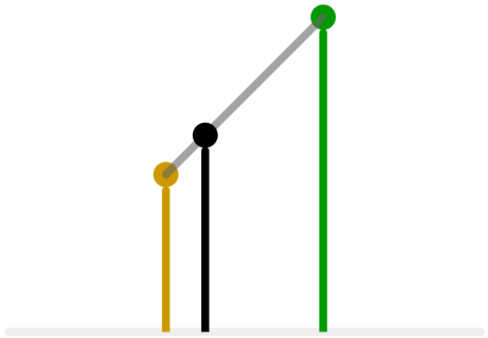
$$
\begin{aligned}
&I(x,y) = \\
&I(0,0)(1-x)(1-y) + \\
&I(0,1)(1-x)(y) + \\
&I(1,0)(x)(1-y) + \\
&I(1,1)(x)(y)
\end{aligned}
$$

In this geometric visualization, the value at the black spot is the sum of the value at each colored spot multiplied by the area of the rectangle of the same color.

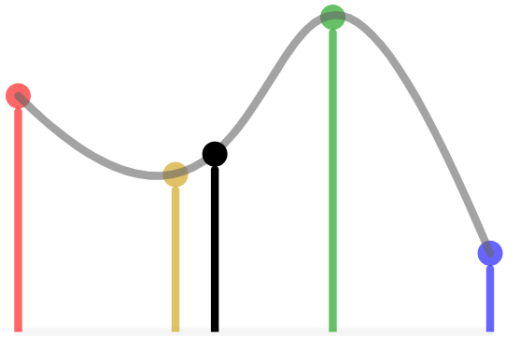
Nearest Neighbor vs Bilinear vs Bicubic Interpolation



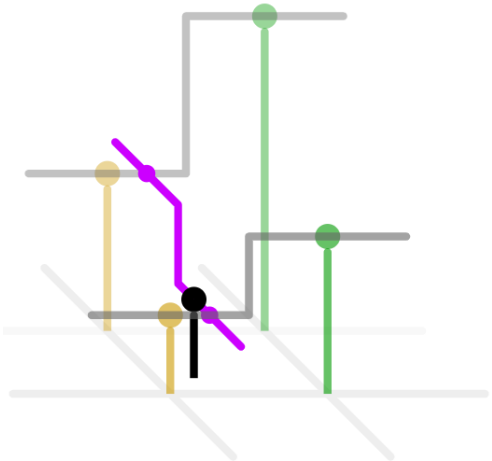
1D nearest-neighbour



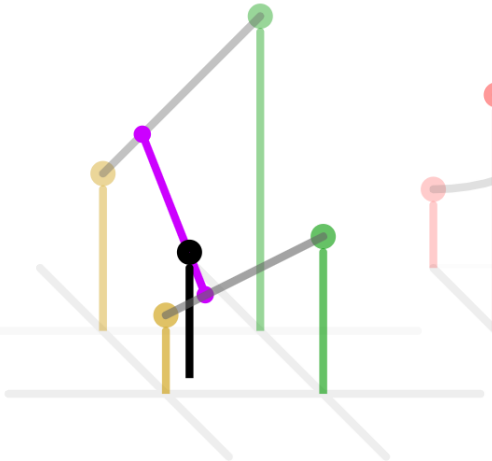
Linear



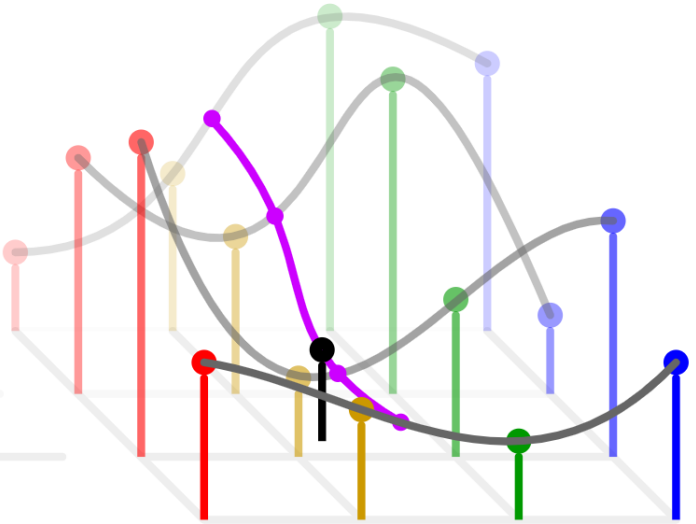
Cubic



2D nearest-neighbour



Bilinear



Bicubic

Nearest Neighbor vs Bilinear vs Bicubic Interpolation

Original image:  x 10

Nearest-neighbor interpolation

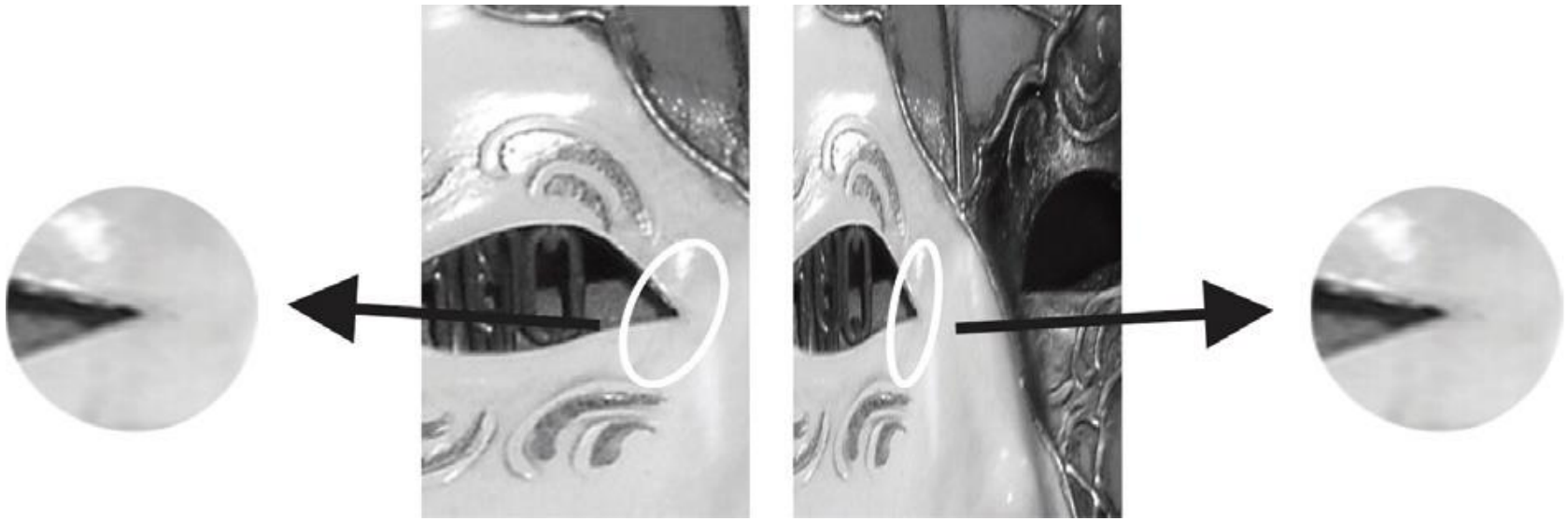
Bilinear interpolation

Bicubic interpolation

Example 2: Affine Warping

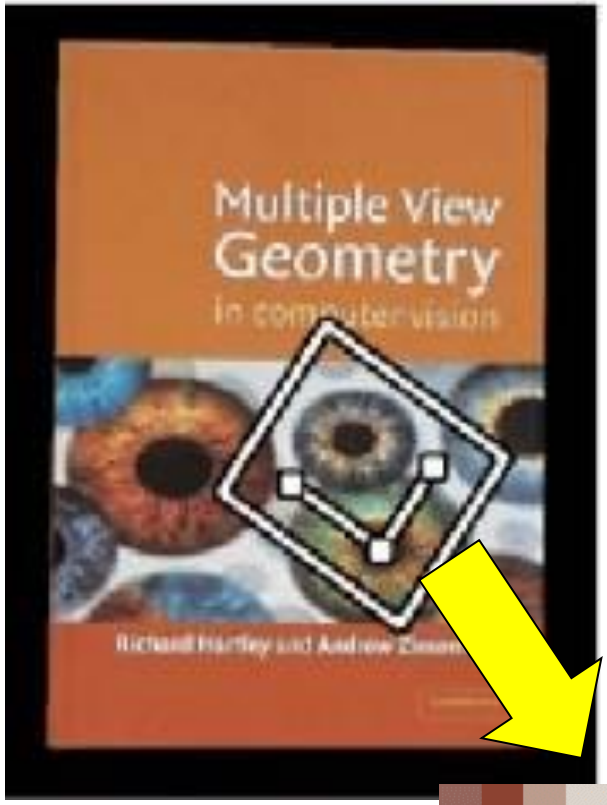
Affine warping (to achieve slight view-point invariance)

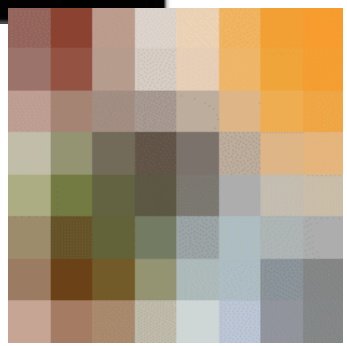
- The second moment matrix *M* can be used to identify the two directions of fastest and slowest change of intensity around the feature.
- Out of these two directions, an elliptic patch is extracted at the scale computed by with the LoG operator.
- The region inside the ellipse is normalized to a circular one

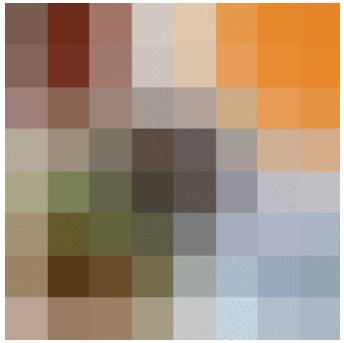


How to achieve invariance

Example: de-rotation, re-scaling, and affine un-warping







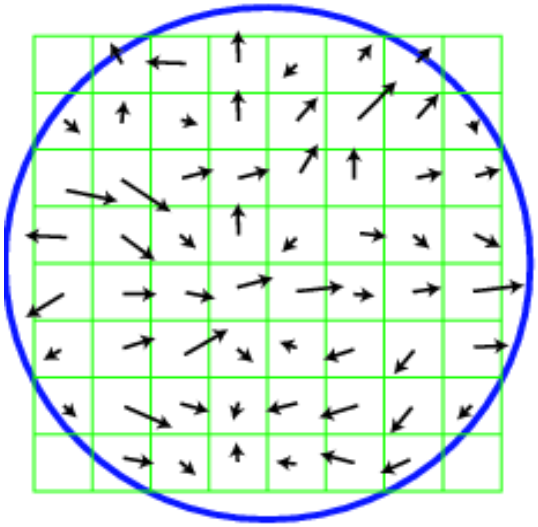
Feature descriptors

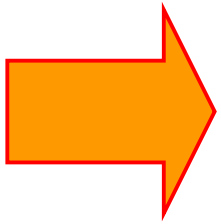
- **Disadvantage of patches descriptors**:
  - If warping not accurately estimated, very small errors in rotation, scale, and view-point will affect matching score significantly
  - Computationally expensive (need to unwarp every patch)

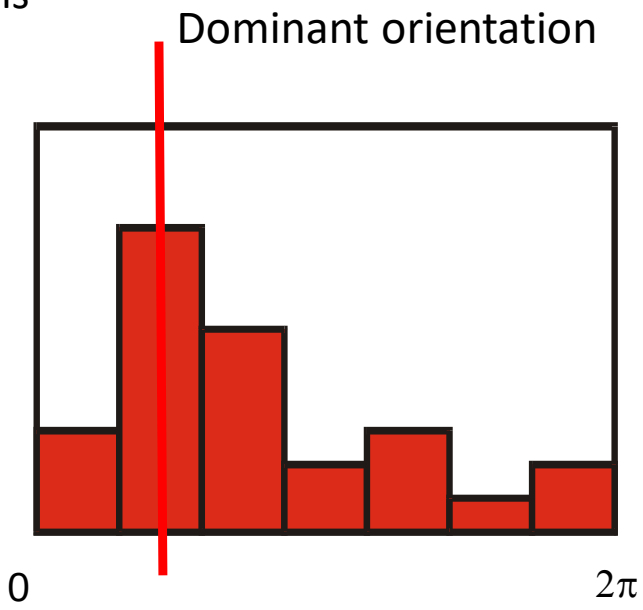
- Better solution **nowadays**: build descriptors from **Histograms of Oriented Gradients (HOGs)**.
  - No need to warp the patch (based on the observation that HOGs are little affected by small viewpoint changes)

HOG descriptor (Histogram of Oriented Gradients)

- First, **multiply the patch by a Gaussian kernel** to make the shape circular rather than square
- Then, **compute gradients vectors** at each pixel
- Build a **histogram of gradient orientations**, weighted by the gradient magnitudes. The histogram represents the HOG descriptor
- Extract all local maxima of HOG.
  - All local maxima above a threshold are all candidate dominant orientations. In this case, **construct a different keypoint descriptor (with different dominant orientation) for each of these peaks**
- To **make the descriptor rotation invariant**, apply circular shift to the descriptor elements such that the dominant orientation coincides with 0 radians





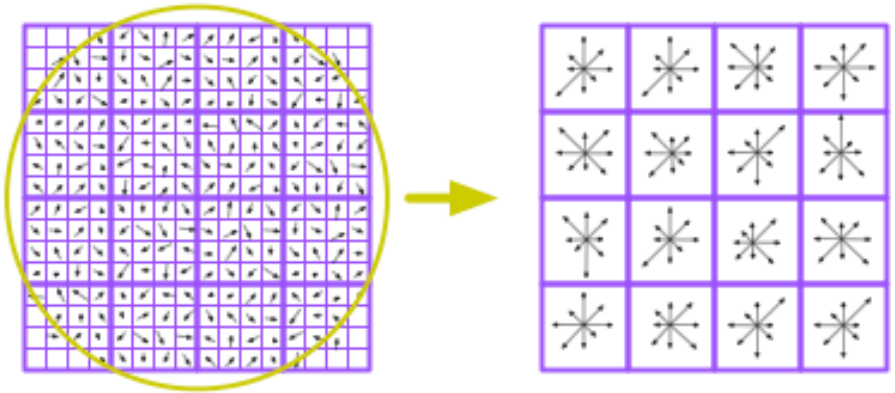


Outline

- Automatic Scale Selection
- The SIFT blob detector and descriptor
- Other corner and blob detectors and descriptors

SIFT descriptor

- **S**cale **I**nvariant **F**eature **T**ransform
- Invented by David Lowe [IJCV, 2004] (now at Google)
- Descriptor computation:
  - Multiply the patch by a Gaussian filter
  - Divide patch into 4×4 sub-patches = 16 cells
  - Compute HOG (8 bins, i.e., 8 directions) for all pixels inside each sub-patch
  - Concatenate all HOGs into a single 1D vector:
    - Resulting SIFT descriptor: 4×4×8 = 128 values
  - Descriptor Matching: SSD (i.e., Euclidean-distance)



Intensity Normalization

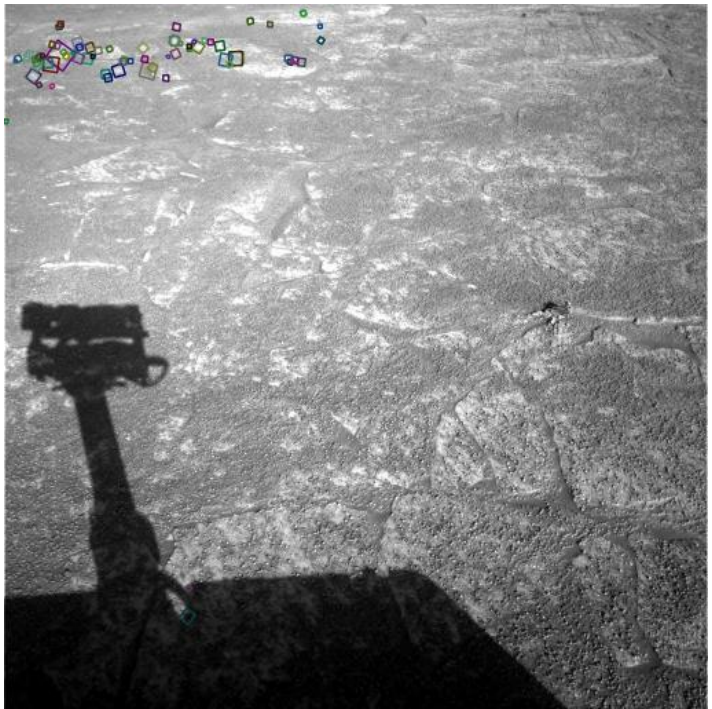
- The descriptor vector $\boldsymbol{v}$ is then normalized such that its l_2 norm is 1:

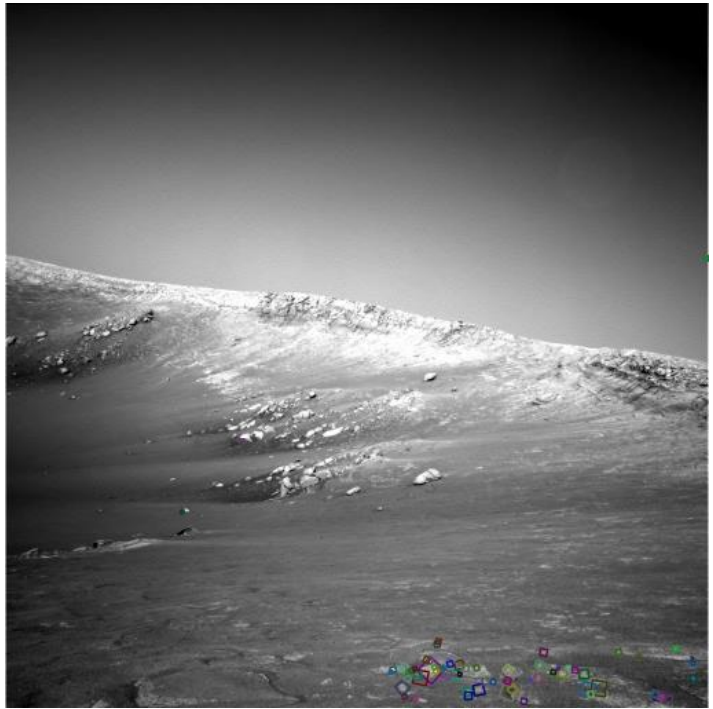
$$\overline{\boldsymbol{v}} = \frac{\boldsymbol{v}}{\sqrt{\sum_i^n v_i^2}}$$

- This guarantees that the descriptor is invariant to linear illumination changes (the descriptor is already invariant to additive illumination because it is based on gradients; so, overall, the SIFT descriptor is invariant to affine illumination changes).

SIFT matching robustness

- Can handle severe viewpoint changes (up to 50 degree out-of-plane rotation)
- Can handle even non affine changes in illumination (low to bright scenes)
- Computationally expensive: 10 frames per second (fps) on an i7 processor
- Original SIFT code (binary files): http://people.cs.ubc.ca/~lowe/keypoints

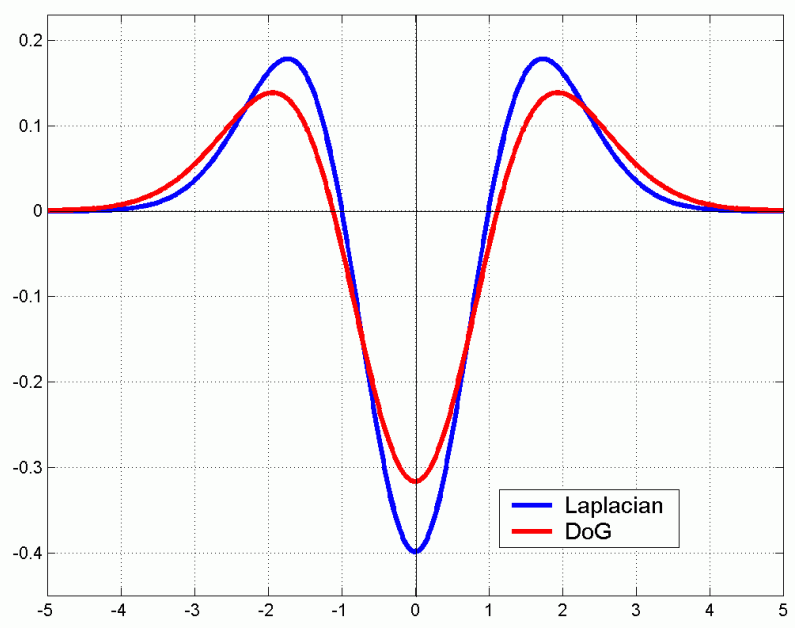




SIFT detector

Difference of Gaussian (DoG) kernel instead of Laplacian of Gaussian (computationally cheaper)

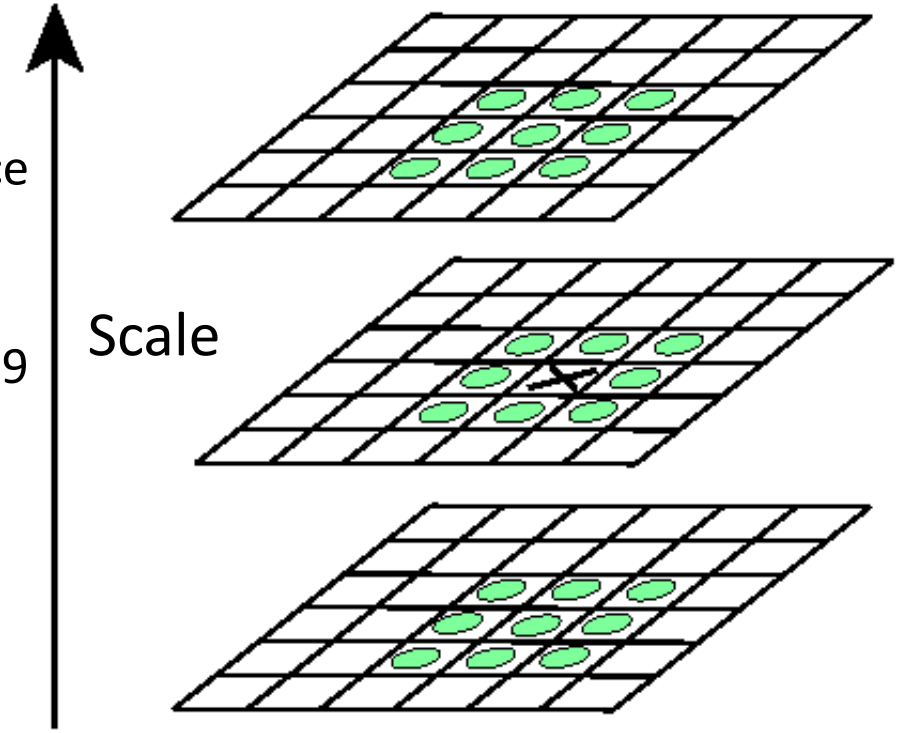
$$LOG \approx DoG = G_{k\sigma}(x, y) - G_{\sigma}(x, y)$$



SIFT detector (location + scale)

SIFT keypoints: **local extrema (i.e., maxima and minima)** in **both space** and **scale** of the DoG images

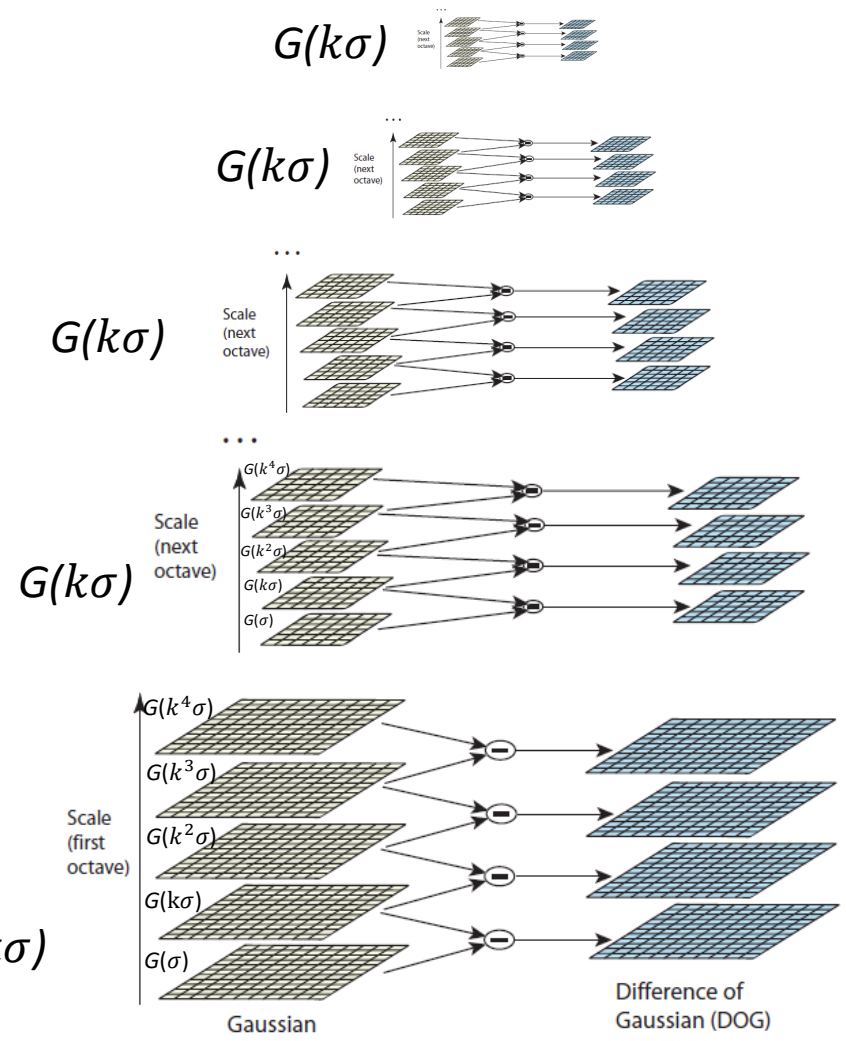
- Detect maxima and minima of difference-of-Gaussian in scale space
- Each point is compared to its 8 neighbors in the current image and 9 neighbours in each of the two adjacent scales (above and below)



For each max or min found, output is the **location** and the **scale**.

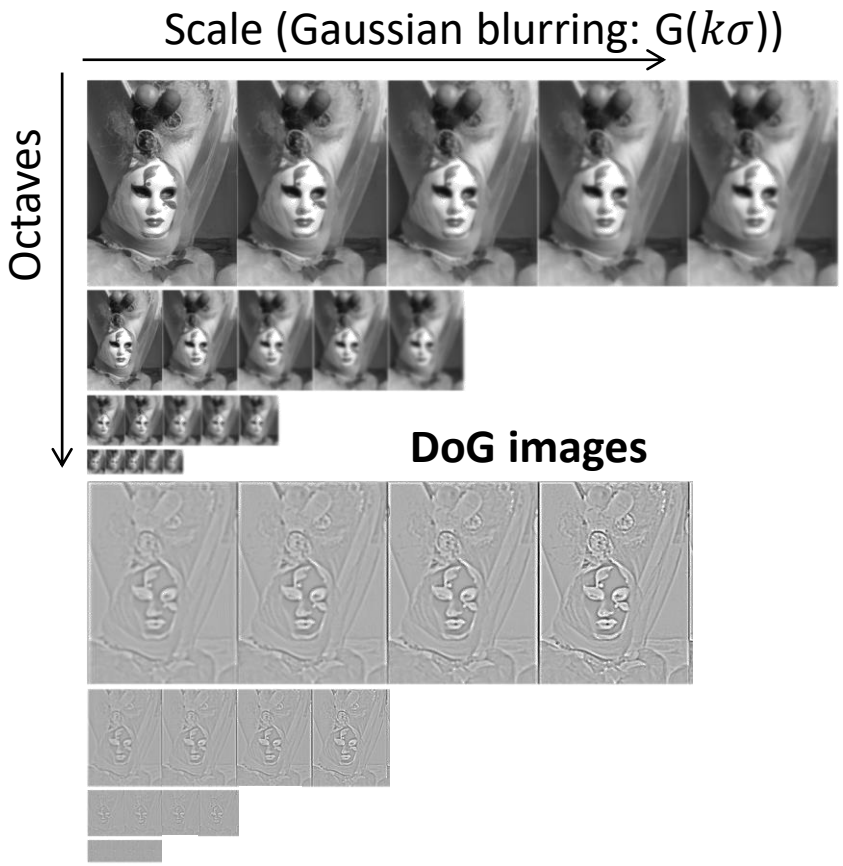
How it is implemented in practice

1. The initial image is **incrementally convolved with Gaussians** $G(k^i\sigma)$ to produce blurred images separated by a constant factor k in scale space (shown stacked in the left column).
   1. The initial Gaussian G(σ) has $\sigma = 1.6$
   2. k is chosen: $k = 2^{1/s}$, where s is the number of intervals into which each octave of scale space is divided
   3. For efficiency reasons, when k^i equals 2, the image is downsampled by a factor of 2 and then the procedure is repeated again up to 5 octaves (pyramid levels)
2. **Adjacent blurred images are then subtracted** to produce the *Difference-of-Gaussian* (DoG) images



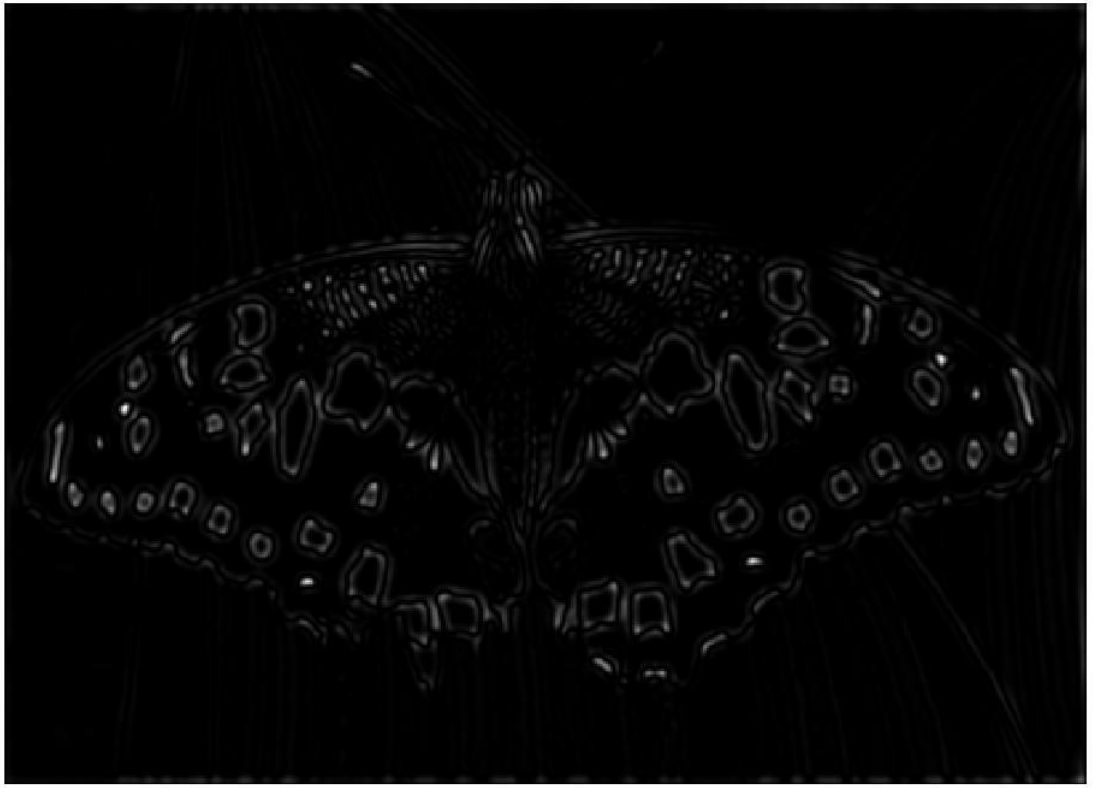
How it is implemented in practice

1. The initial image is **incrementally convolved with Gaussians** $G(k^i\sigma)$ to produce blurred images separated by a constant factor k in scale space (shown stacked in the left column).
   1. The initial Gaussian $G(\sigma)$ has $\sigma = 1.6$
   2. k is chosen: $k = 2^{1/s}$, where s is the number of intervals into which each octave of scale space is divided
   3. For efficiency reasons, when k^i equals 2, the image is downsampled by a factor of 2 and then the procedure is repeated again up to 5 octaves (pyramid levels)
2. **Adjacent blurred images are then subtracted** to produce the *Difference-of-Gaussian* (DoG) images



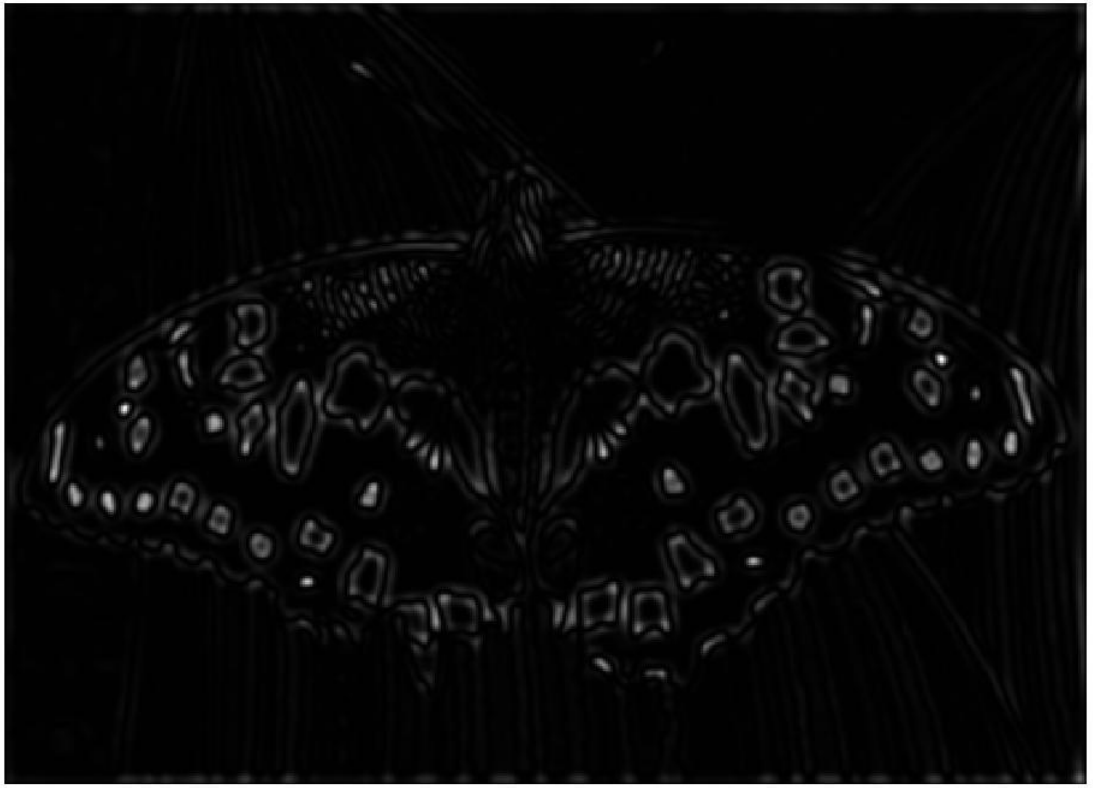
Scale-space detection: Example

DoG Images example



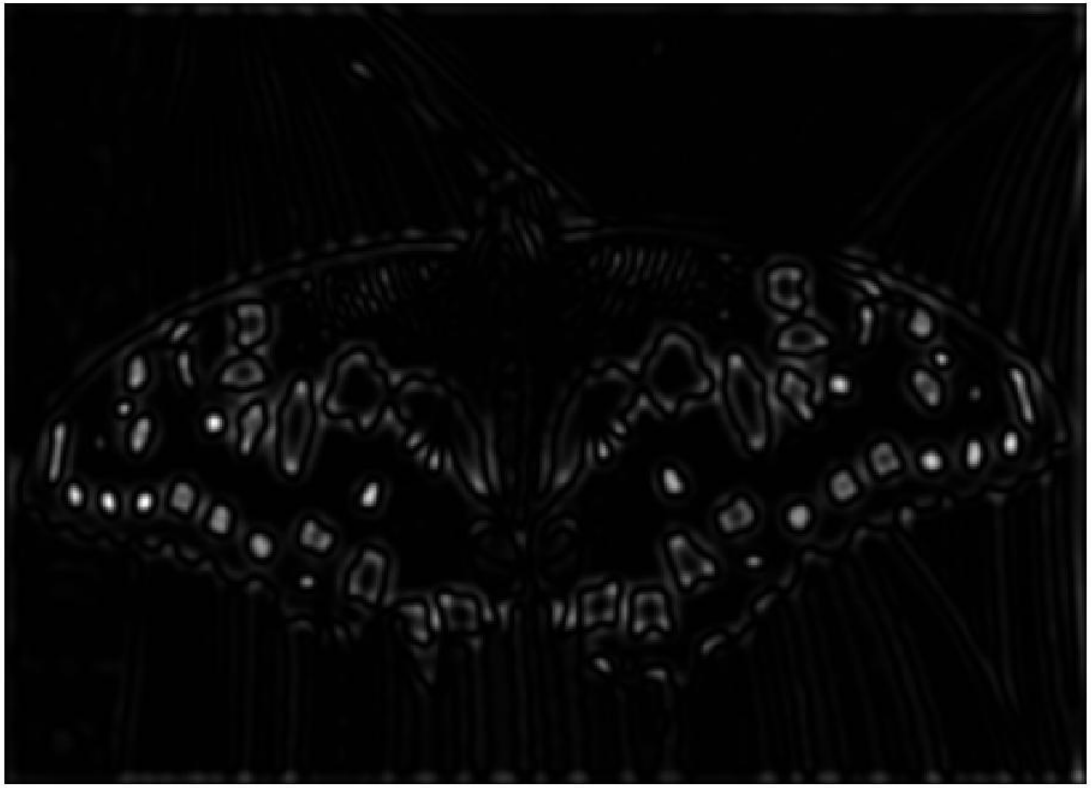
$G(k\sigma) - G(\sigma)$ | $s = 4; \sigma = 1.6$ |

DoG Images example



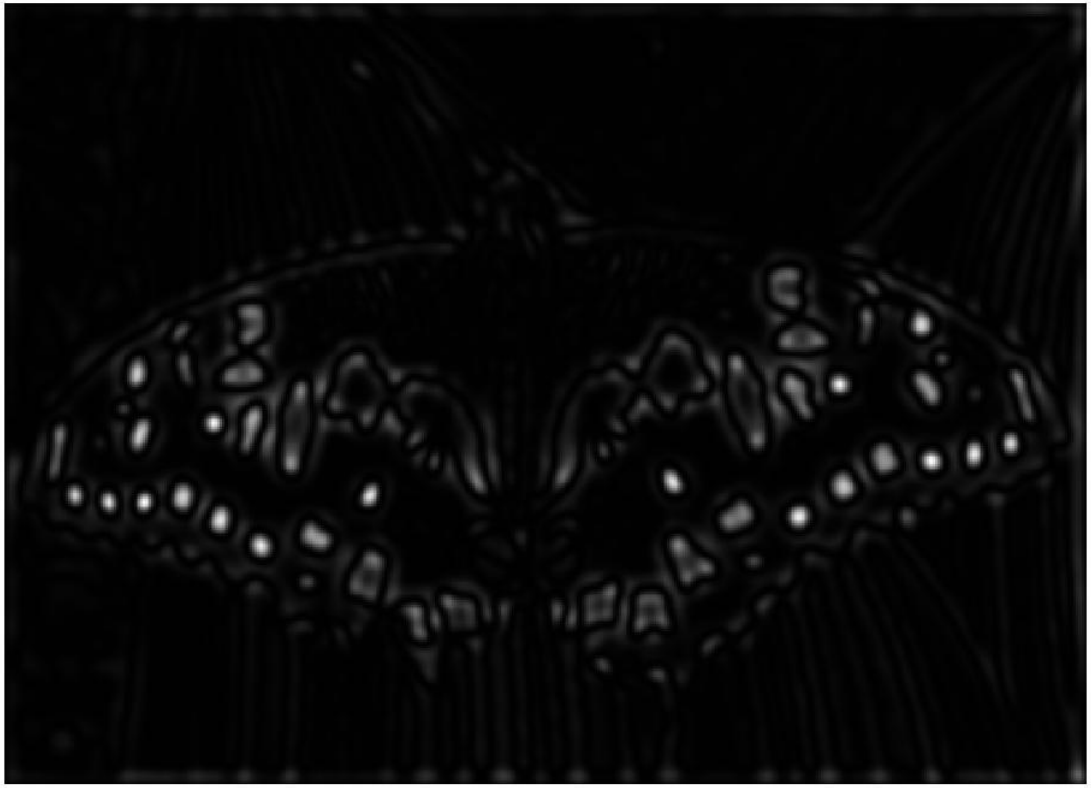
$G(k^2\sigma) - G(k\sigma) \mid s = 4; \sigma = 1.6 \mid$

DoG Images example



$G(k^3\sigma) - G(k^2\sigma) \mid s = 4; \sigma = 1.6 \mid$

DoG Images example

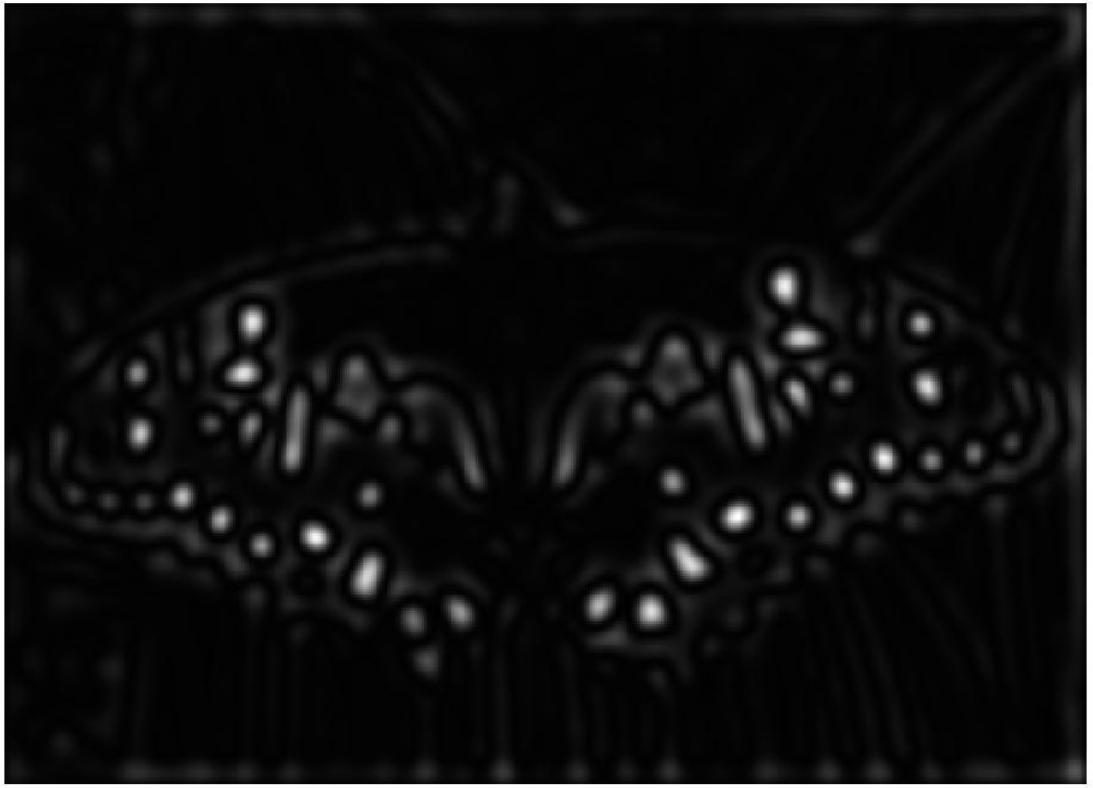


$G(k^4\sigma) - G(k^3\sigma)$ | $s = 4; \sigma = 1.6$ |

DoG Images example

$G(k^5\sigma) - G(k^4\sigma)$ | $s = 4; \sigma = 1.6$ |
(second octave shown at the input resolution for convenience)

DoG Images example



$G(k^6\sigma) - G(k^5\sigma)$ | $s = 4; \sigma = 1.6$ |
(second octave shown at the input resolution for convenience)

DoG Images example

$G(k^7\sigma) - G(k^6\sigma)$ | $s = 4; \sigma = 1.6$ |
(second octave shown at the input resolution for convenience)

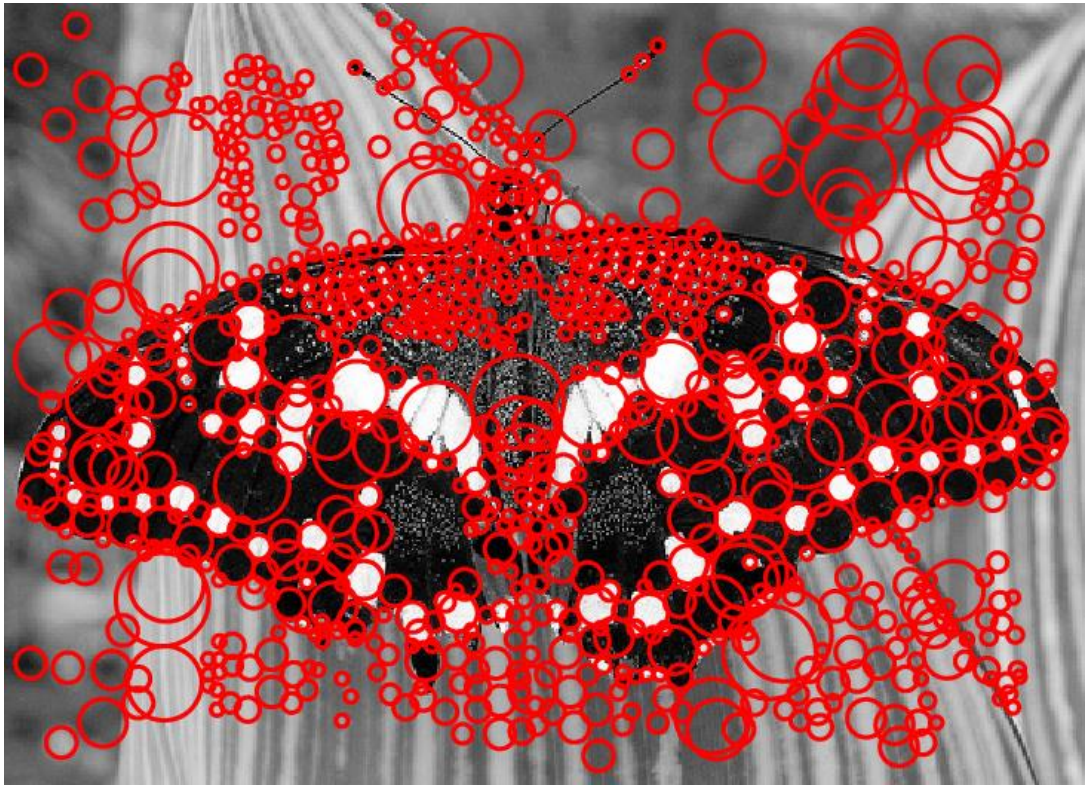
DoG Images example

$G(k^8\sigma) - G(k^7\sigma)$ | $s = 4; \sigma = 1.6$ |
(second octave shown at the input resolution for convenience)

DoG Images example

$G(k^9\sigma) - G(k^8\sigma)$ | $s = 4; \sigma = 1.6$ |
(third octave shown at the input resolution for convenience)

DoG local maxima and minima



SIFT Features: Summary

- **SIFT: Scale Invariant Feature Transform** [Lowe, IJCV 2004]
- An approach to **detect and describe** regions of interest in an image.
  - NB: SIFT detector = DoG detector
- SIFT features are **reasonably invariant** to changes in **rotation, scaling**, and **changes in viewpoint** (up to 50 degrees) **and illumination**
- **Real-time but still slow (10 Hz on an i7 laptop)**
  - Expensive steps are the scale detection and descriptor extraction

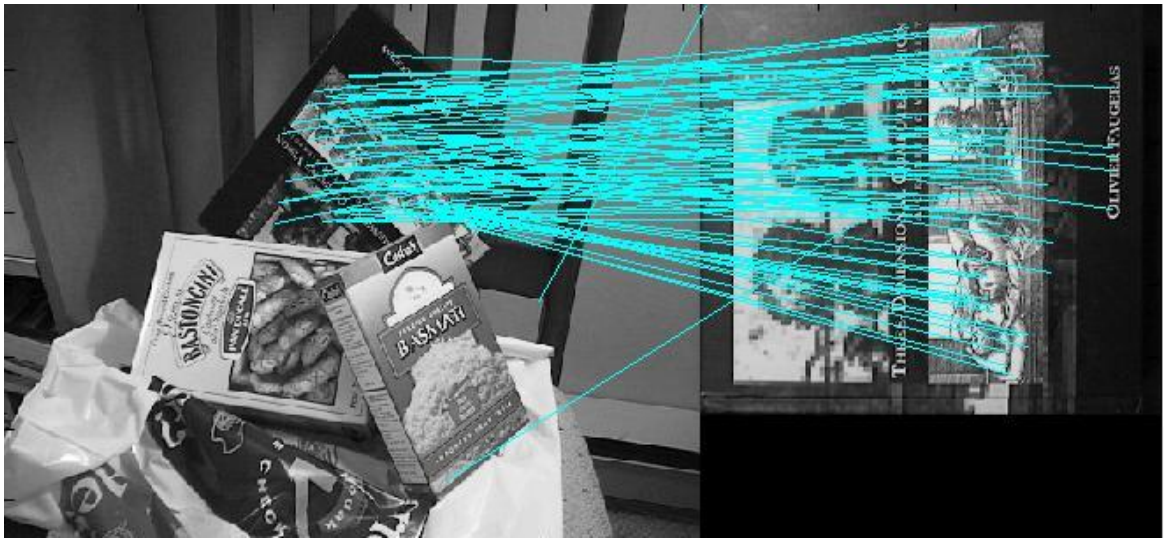
SIFT Demo

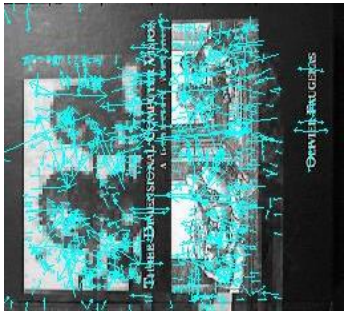
- Download original SIFT binaries and Matlab function from : http://people.cs.ubc.ca/~lowe/keypoints

```
>>[image1, descriptor1s, locs1] = sift('scene.pgm');
>>showkeys(image1, locs1);

>>[image2, descriptors2, locs2] = sift('book.pgm');
>>showkeys(image2, locs2);

>>match('scene.pgm','book.pgm');
```

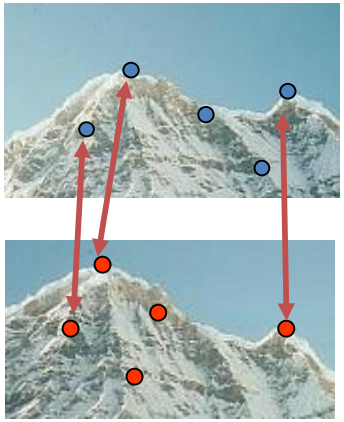


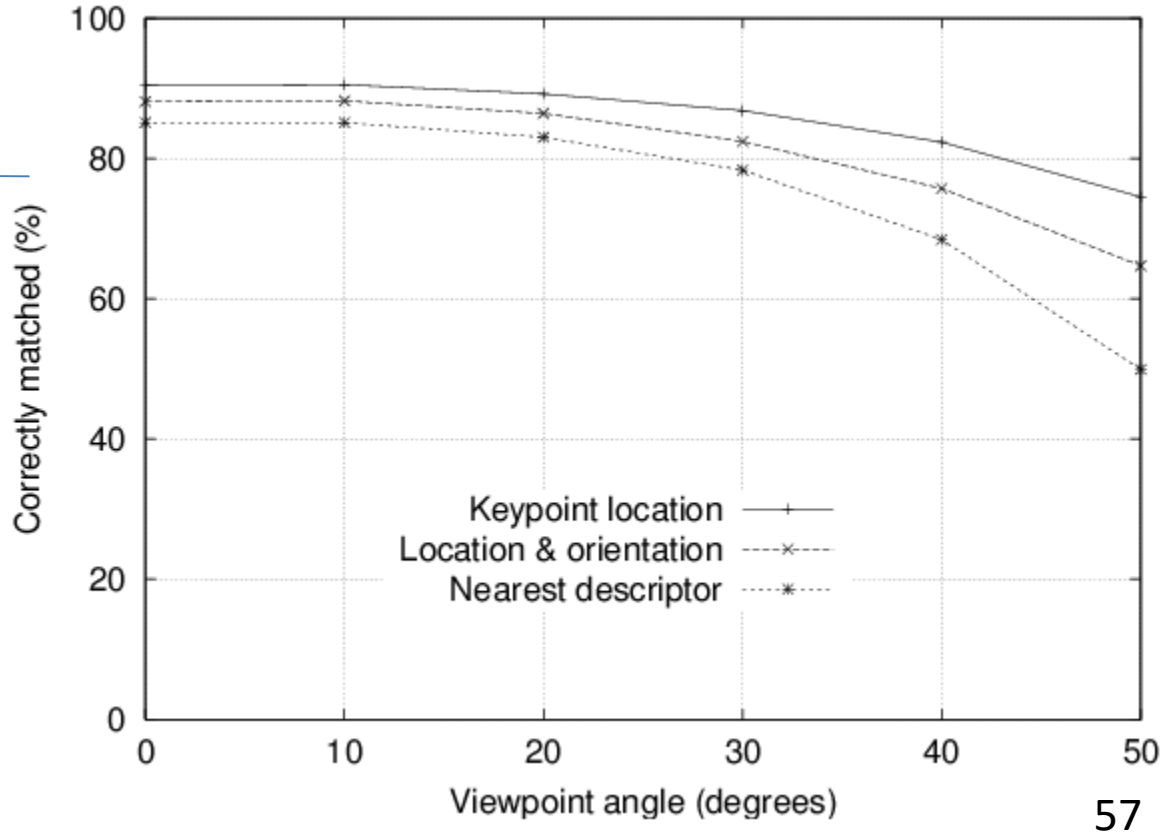


SIFT repeatability vs. viewpoint angle

Repeatability=

$$\frac{\text{\# correspondences detected}}{\text{\# correspondences present}}$$



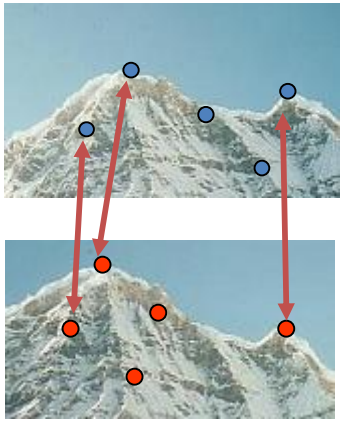


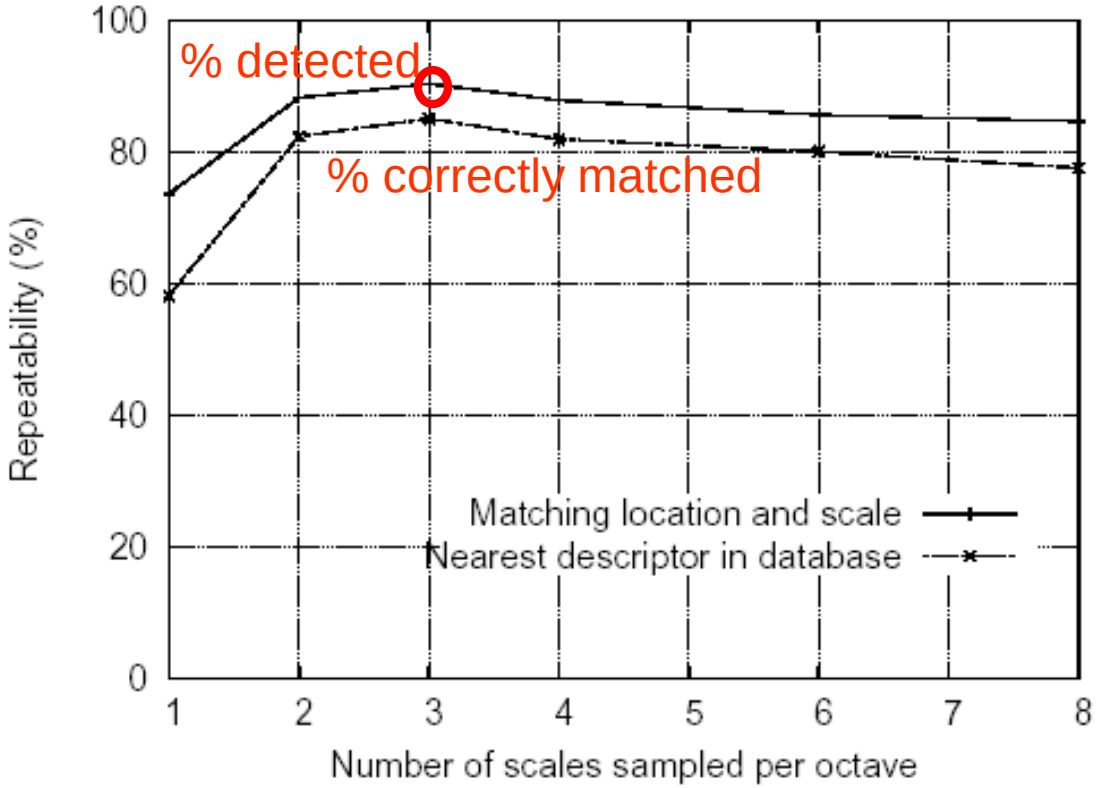
SIFT repeatability vs. Scale

The highest repeatability is obtained when sampling 3 scales per octave!

Repeatability=

$$\frac{\text{\# correspondences detected}}{\text{\# correspondences present}}$$





Influence of Number of Orientations and Number of Sub-patches

The graph shows that a single orientation histogram (n = 1) is very poor at discriminating. The results improve with a 4x4 array of histograms with 8 orientations.

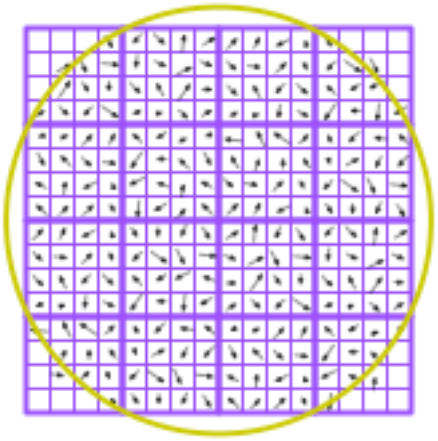


Image gradients

4x4 HOGs

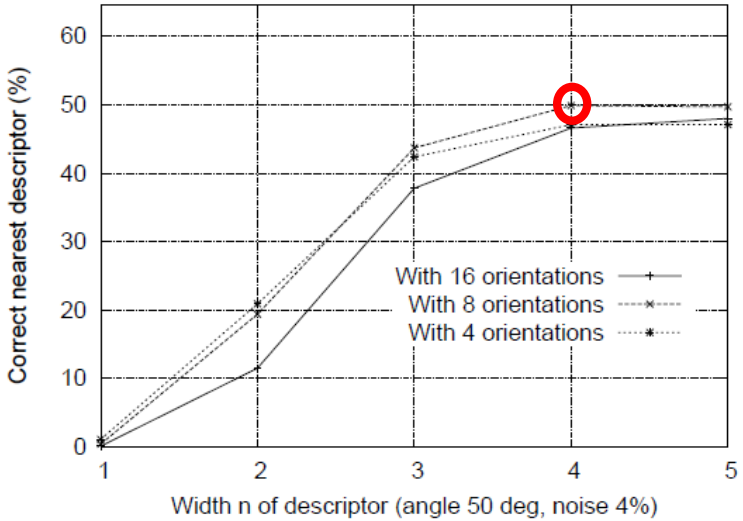
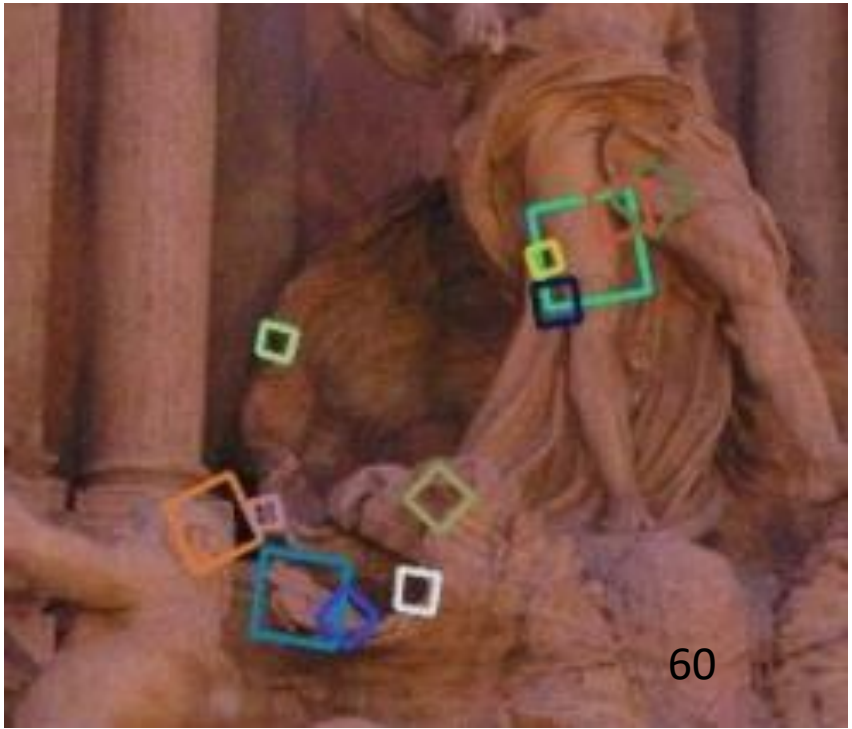


Figure 8: This graph shows the percent of keypoints giving the correct match to a database of 40,000 keypoints as a function of width of the $n \times n$ keypoint descriptor and the number of orientations in each histogram. The graph is computed for images with affine viewpoint change of 50 degrees and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

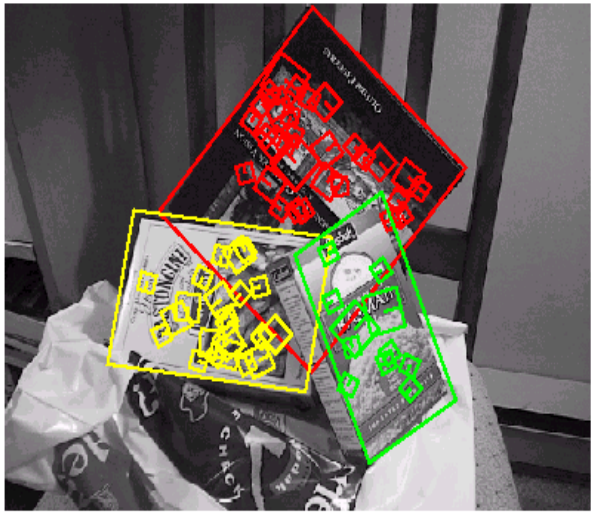
How many parameters are used to define a SIFT feature?

- **Descriptor**: 4x4x8 = 128-element 1D vector
- **Location** (pixel coordinates of the center of the patch): 2D vector
- **Scale** (i.e., size) of the patch: 1 scalar value
- **Orientation** (i.e., angle of the patch): 1 scalar value



SIFT for Object recognition

- Can be implemented easily by returning object with the largest number of correspondences with the template
- For planar objects, 4 point RANSAC can be used to remove outliers (see next lectures).
- For rigid 3D objects, 5 point RANSAC (see next lectures).



SIFT for Panorama Stitching

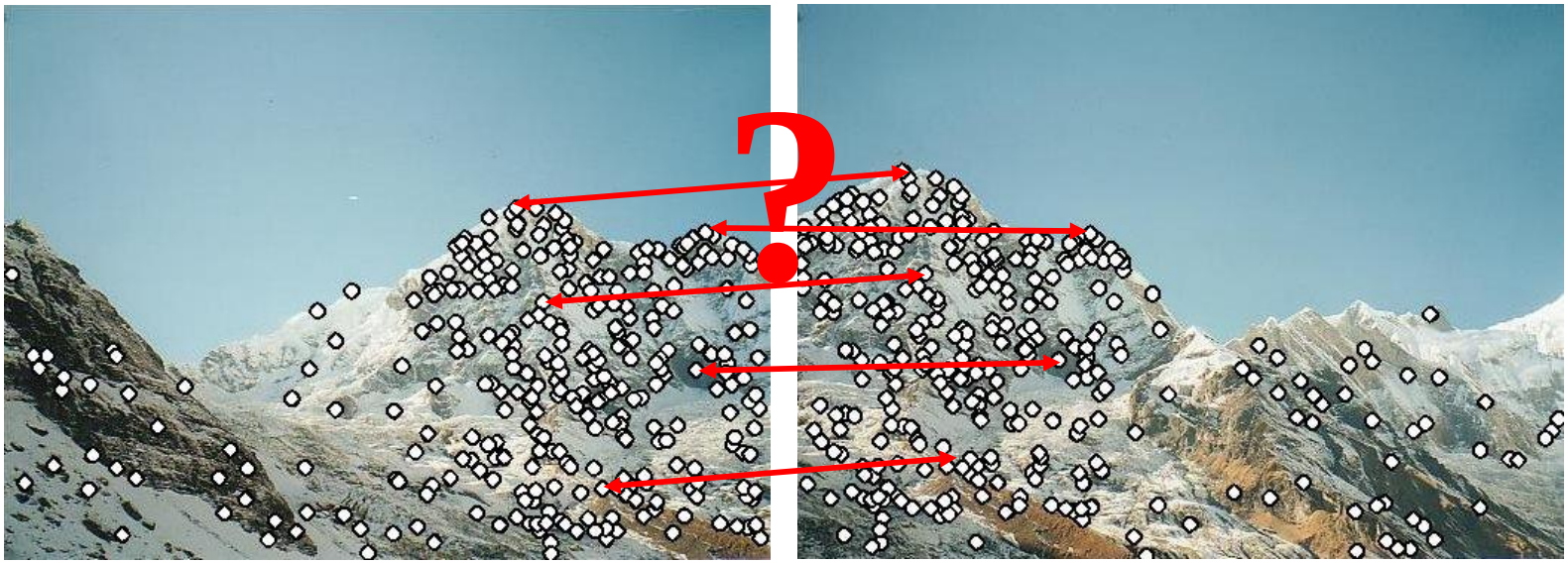
AutoStitch: http://matthewalunbrown.com/autostitch/autostitch.html

[M. Brown and D. G. Lowe. Recognising Panoramas. ICCV 2003]

Main questions

- What points are distinctive (i.e., *features*, *keypoints, salient* points), such that they are *repeatable? (i.e.,* can be re-*detected* from other views)
- How to *describe* a local region?
- How to establish *correspondences*, i.e., compute matches?

Feature matching



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 ((Z)SSD, SAD, NCC or Hamming distance for binary descriptors (e.g., Census, BRIEF, BRISK)
  2. **Brute-force matching:**
     1. Test all the features in I_2
     2. Take the one at min distance

- **Issues with closest descriptor**: can give good scores to very ambiguous (bad) matches (curse of dimensionality)
- **Better approach**: compute ratio of distances to 1st to 2nd closest match

$$d(f_1) \,/\, d(f_2) \;<\; Threshold\ (usually\ 0.8)$$

  - $d(f_1)$ is the distance of the closest neighbor
  - $d(f_2)$ is the distance of the 2nd closest neighbor

Distance ratio: Explanation

- In SIFT, the nearest neighbor is defined as the keypoint with minimum Euclidean distance. However, many features from an Image 1 may not have any correct match in Image 2 because they arise from background clutter or were not detected in the Image 1.
- An effective measure is obtained by comparing the distance of the **closest neighbor** to that of the **second-closest neighbor**. This measure performs well because correct matches need to have the closest neighbor significantly closer than the closest incorrect match to achieve reliable matching.
- For false matches, there will likely be a number of other false matches within similar distances due to the high dimensionality of the feature space (this problem is known as **curse of dimensionality***)*. We can think of the second-closest match as providing an estimate of the density of false matches within this portion of the feature space and at the same time identifying specific instances of feature ambiguity.

SIFT Feature matching: distance ratio

The SIFT paper recommends to use a threshold on 0.8. Where does this come from?

"A threshold of 0.8 eliminates 90% of the false matches while discarding less than 5% of the correct matches."

"This figure was generated by matching images following random scale and orientation change, a depth rotation of 30 degrees, and addition of 2% image noise, against a database of 40,000 keypoints."

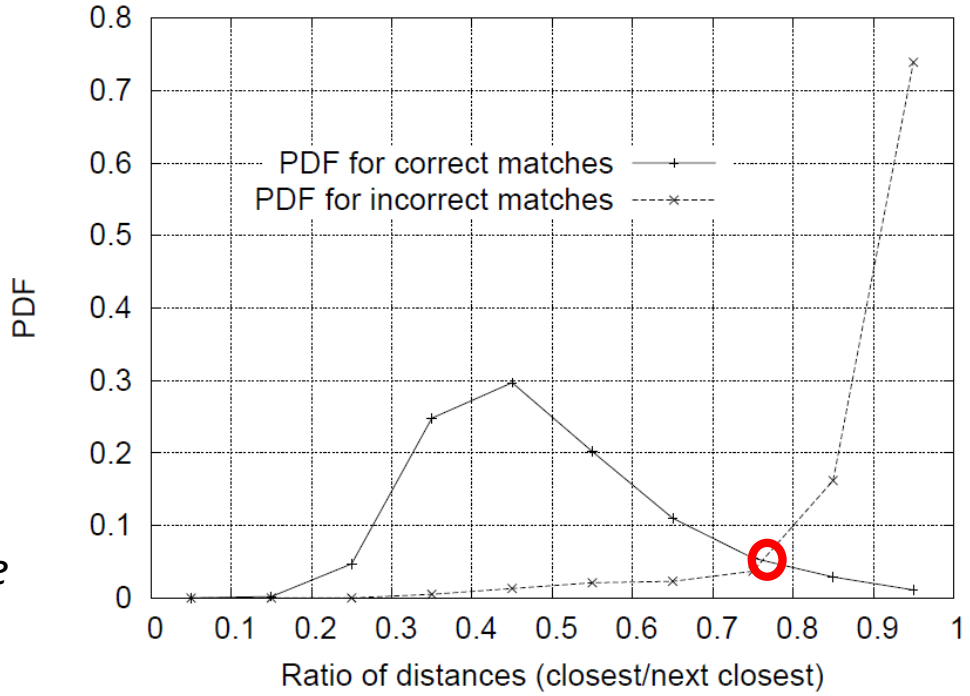


Figure 11: The probability that a match is correct can be determined by taking the ratio of distance from the closest neighbor to the distance of the second closest. Using a database of 40,000 keypoints, the solid line shows the PDF of this ratio for correct matches, while the dotted line is for matches that were incorrect.

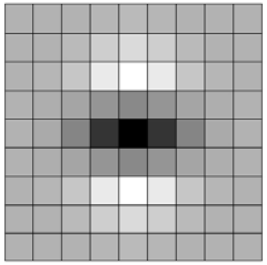
Outline

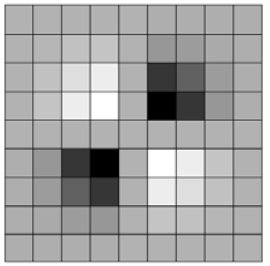
- Automatic Scale Selection
- The SIFT blob detector and descriptor
- Other corner and blob detectors and descriptors

SURF [Bay et al., ECCV 2006]

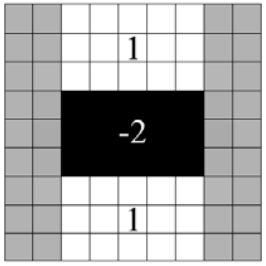
- ***S***peeded ***U***p ***R***obust ***F***eatures
- Based on ideas similar to **SIFT**
- Approximated computation for detection and descriptor
- Results comparable with SIFT, plus:
  - Faster computation
  - Generally shorter descriptors

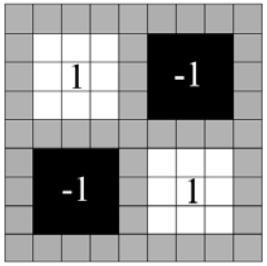
Original second order partial derivatives of a Gaussian





SURF Approximation using **box filter**

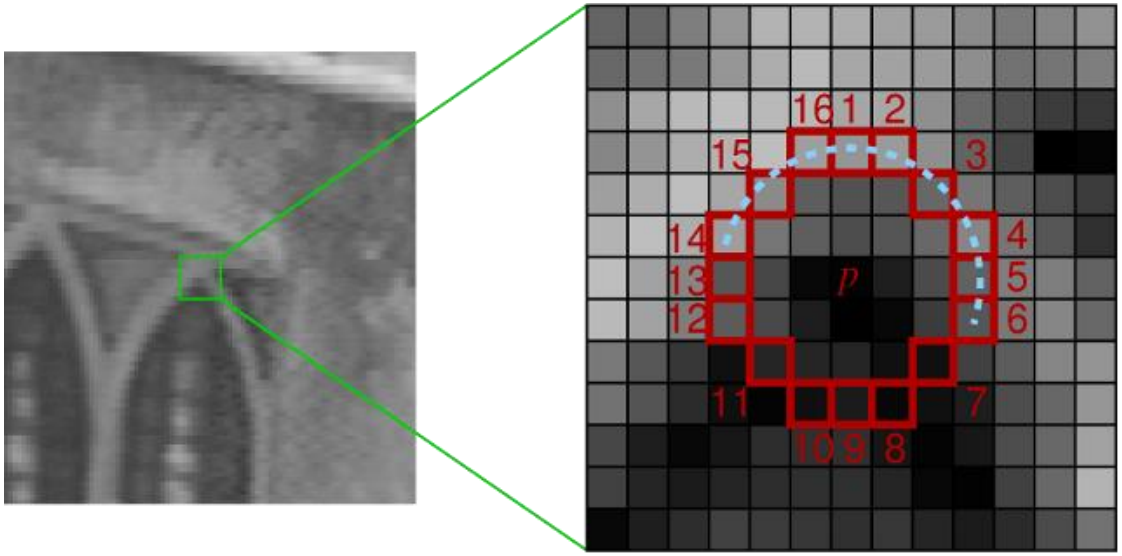




Bay, Tuytelaars, Van Gool, " Speeded Up Robust Features ", European Conference on Computer Vision (ECCV) 2006

FAST detector [Rosten et al., ICCV'05]

- **FAST**: **F**eatures from **A**ccelerated **S**egment **T**est
- Studies intensity of pixels on circle around candidate pixel $\boldsymbol{C}$
- $\boldsymbol{C}$ is a FAST corner **if** a set of $\boldsymbol{N}$ contiguous pixels on circle are:
  - all brighter than $\boldsymbol{intensity\_of(C) + theshold}$, or
  - all darker than $\boldsymbol{intensity\_of(C) + theshold}$



- Typically tests for **9** contiguous pixels in a **16**-pixel circumference
- **Very fast detector** - in the order of 100 Mega-pixel/second

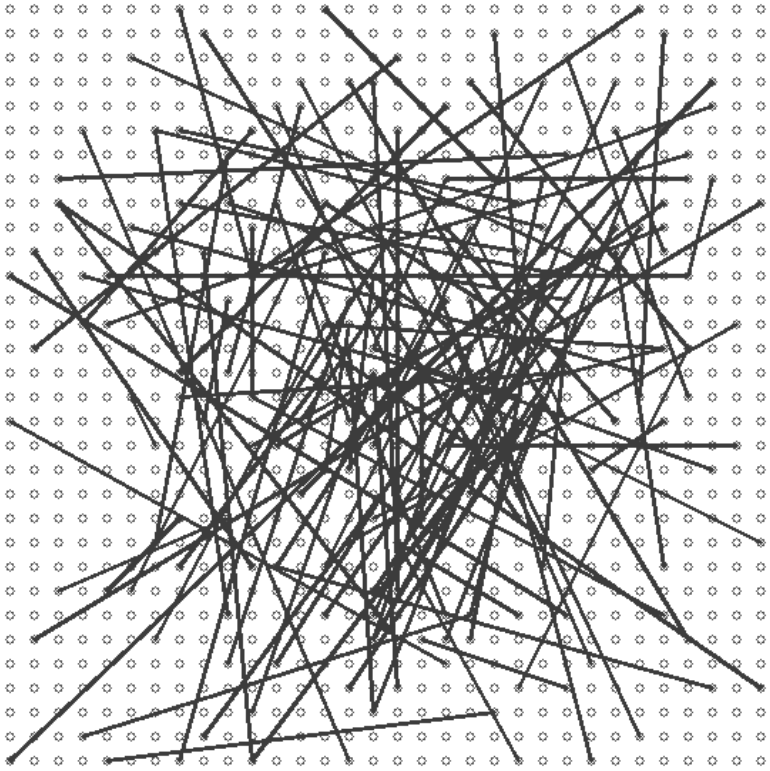
Rosten, Drummond, Fusing points and lines for high performance tracking, International Conference on Computer Vision (ICCV), 2005

BRIEF descriptor [Calonder et. al, ECCV 2010]

- **B**inary **R**obust **I**ndependent **E**lementary **F**eatures
- Goal: high speed (in description and matching)

- **Binary** descriptor formation:
  - Smooth image
  - **for each** detected keypoint (e.g. FAST),
  - ***sample*** 256 intensity pairs $({p_1}^i, {p_2}^i)$ $(i = 1, \dots, 256)$ within a squared patch around the keypoint
  - Create an empty 256-element descriptor
  - **for each** i^{th}**pair**
    - **if** $I_{{p_1}^i} < I_{{p_2}^i}$ **then *set*** i^{th} bit of descriptor to **1**
    - **else** to **0**

- The **pattern is generated randomly** (or by **machine learning**) only once; then, the same pattern is used for all patches
- Pros: **Binary descriptor**: allows **very fast** Hamming distance matching (count of the number of bits that are different in the descriptors matched)
- Cons: **Not scale/rotation invariant**

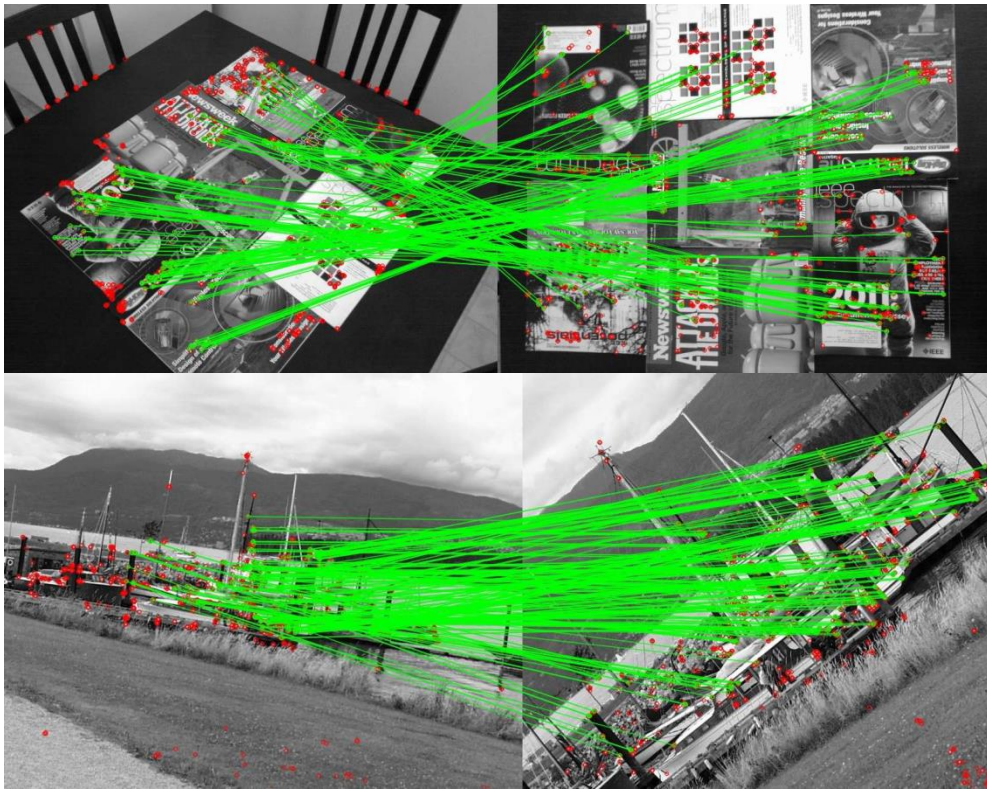


Pattern for intensity pair samples – generated randomly

Calonder, Lepetit, Strecha, Fua, BRIEF: Binary Robust Independent Elementary Features, ECCV'10

ORB descriptor [Rublee et al., ICCV 2011]

- ***O***riented FAST and ***R***otated ***B***RIEF
- Keypoint detector based on **FAST**
- **BRIEF** descriptors are *steered* according to keypoint orientation (to provide rotation invariance)
- Good Binary features are learned by minimizing the correlation on a set of training patches.



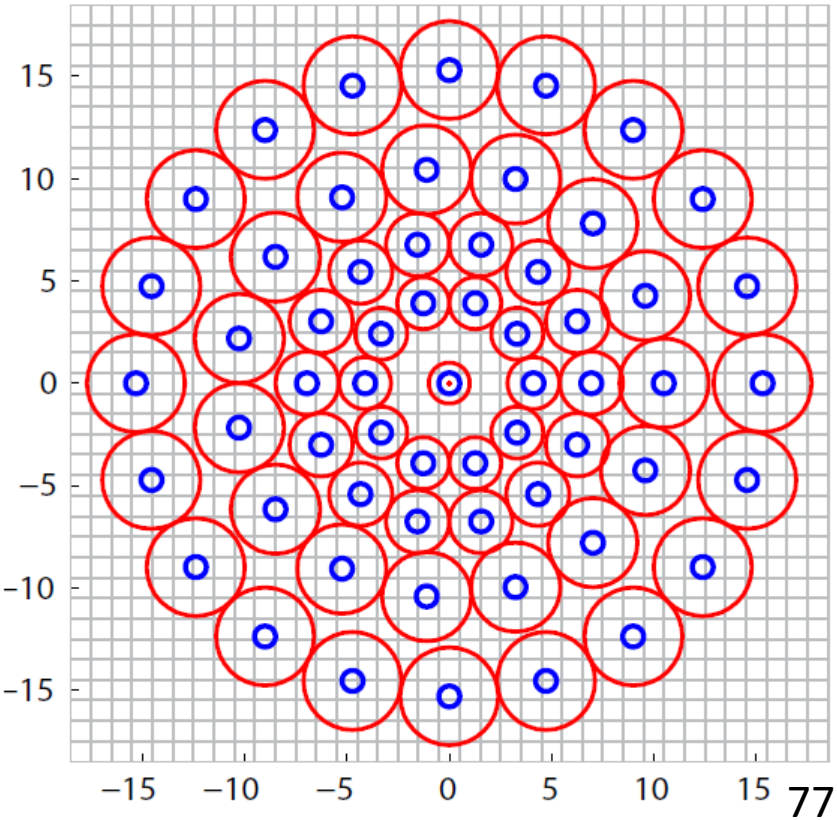
Rublee,Rabaud, Konolige, Bradski, (2011). "ORB: an efficient alternative to SIFT or SURF" (PDF). IEEE International Conference on Computer Vision (ICCV).

BRISK descriptor

[Leutenegger, Chli, Siegwart, ICCV 2011]

- **B**inary **R**obust **I**nvariant **S**calable **K**eypoints
- Detect corners in scale-space using FAST
- Rotation and scale invariant

- **Binary**, formed by pairwise intensity comparisons (like BRIEF)
- **Pattern** defines intensity comparisons in the keypoint neighborhood
- **Red circles**: size of the smoothing kernel applied
- **Blue circles**: smoothed pixel value used
- Compare short- and long-distance pairs for orientation assignment & descriptor formation
- Detection and descriptor speed: ~10 times faster than SURF
- Slower than BRIEF, but scale- and rotation- invariant

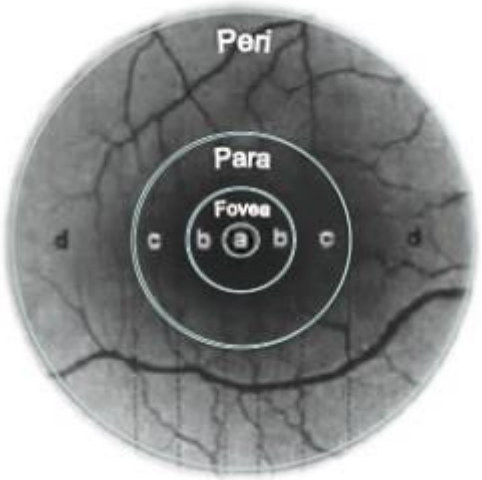


FREAK descriptor [Alahi, Ortiz, Vandergheynst, CVPR 2012]

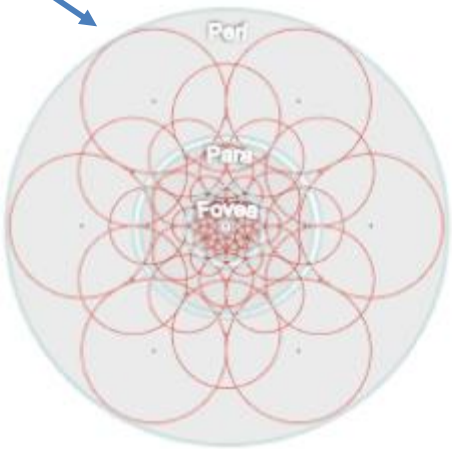
- **F**ast **R**etina **K**eypoint
- Rotation and scale invariant

- Fast, compact and robust keypoint descriptor
- **Sampling pattern** inspired by the human retina: higher density of points near the center.
- **Pairwise** intensity comparisons form **binary** strings.
- Pairs are **learned** (as in ORB).
- Circles indicate size of smoothing kernel.
- Orientation mechanism similar to BRISK.
- Coarse-to-fine matching (cascaded approach): first compare the first 128 bits; if distance smaller than threshold, proceed to compare the next bits, etc.

- Faster to compute, less memory and more robust than SIFT, SURF or BRISK.



Human retina



FREAK sampling pattern

Alexandre Alahi, Raphael Ortiz, Pierre Vandergheynst, FREAK: Fast Retina Keypoint, CVPR 2012

Recap Table

Detector	Localization Accuracy of the detector	Descriptor that can be used	Efficiency	Relocalization & Loop closing
Harris	++++	Patch SIFT BRIEF ORB BRISK FREAK	+++ + ++++ ++++ +++ ++++	+ +++++ +++ ++++ +++ ++++
Shi-Tomasi	++++	Patch SIFT BRIEF ORB BRISK FREAK	++ + ++++ ++++ +++ ++++	+ +++++ +++ ++++ +++ ++++
FAST	++++	Patch SIFT BRIEF ORB BRISK FREAK	++++ + ++++ ++++ +++ ++++	+++ +++++ +++ ++++ +++ ++++
SIFT	+++	SIFT	+	++++
SURF	+++	SURF	++	++++

Summary (things to remember)

- Similarity metrics: NCC (ZNCC), SSD (ZSSD), SAD (ZSAD), Census Transform
- Point feature detection
  - Properties and invariance to transformations
    - Challenges: rotation, scale, view-point, and illumination changes
  - Extraction
    - Moravec
    - Harris and Shi-Tomasi
      - Rotation invariance
  - Automatic Scale selection
  - Descriptor
    - Intensity patches
      - Canonical representation: how to make them invariant to transformations: rotation, scale, illumination, and view-point (affine)
    - Better solution: Histogram of oriented gradients: SIFT descriptor
  - Matching
    - (Z)SSD, SAD, NCC, Hamming distance (last one only for binary descriptors)
      ratio 1^{st} /2^{nd} closest descriptor
  - Depending on the task, you may want to trade off repeatability and robustness for speed: approximated solutions, combinations of efficient detectors and descriptors.
    - Fast corner detector: FAST;
    - Keypoint descriptors faster than SIFT: SURF, BRIEF, ORB, BRISK

Reading

- Ch. 4.1 and Ch. 8.1 of Szeliski book
- Ch. 4 of Autonomous Mobile Robots book
- Ch. 13.3 of Peter Corke book

Understanding Check

Are you able to answer:

- How does automatic scale selection work?
- What are the good and the bad properties that a function for automatic scale selection should have or not have?
- How can we implement scale invariant detection efficiently? (show that we can do this by resampling the image vs rescaling the kernel).
- What is a feature descriptor? (patch of intensity value vs histogram of oriented gradients). How do we match descriptors?
- How is the keypoint detection done in SIFT and how does this differ from Harris?
- How does SIFT achieve orientation invariance?
- How is the SIFT descriptor built?
- What is the repeatability of the SIFT detector after a rescaling of 2? And for a 50 degrees viewpoint change?
- Illustrate the 1st to 2nd closest ratio of SIFT detection: what's the intuitive reasoning behind it? Where does the 0.8 factor come from?