Università di Genova
DIBRIS DIPARTIMENTO
DI INFORMATICA, BIOINGEGNERIA,
ROBOTICA E INGEGNERIA DEI SISTEMI

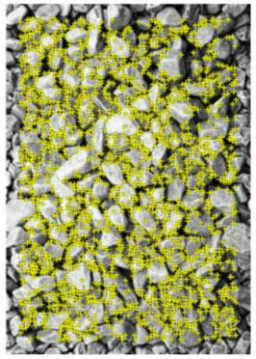
Augmented Reality

Lecture 9 – corners, blobs and disparity (stereo images)

Manuela Chessa – manuela.chessa@unige.it
Fabio Solari – fabio.solari@unige.it

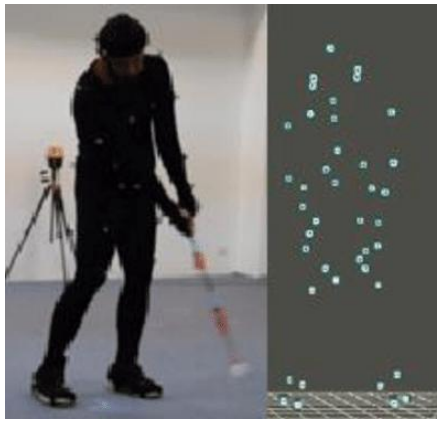
Summary

Corner detection



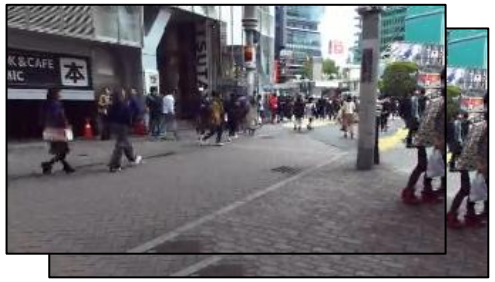
AR

Blob detection



avatar

Stereopsis



depth perception

3D reconstruction

Università di Genova
DIBRIS DIPARTIMENTO
DI INFORMATICA, BIOINGEGNERIA,
ROBOTICA E INGEGNERIA DEI SISTEMI

Corners

Feature points

- The first kind of features that humans may notice in images are *specific locations that attract attention*, such as mountain peaks, building corners, doorways, or interestingly shaped patches.

- These kinds of local features are often called *keypoints* or *interest points* or even *corners* and are often described by the *appearance of patches* of pixels surrounding the point location.

- Edges and lines provide information that is complementary to both keypoints and region-based descriptors.

Correspondence across views

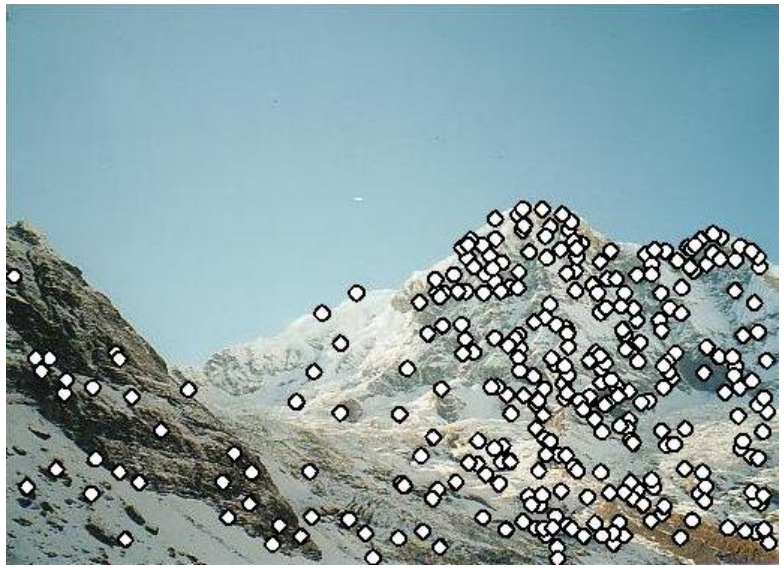
- Feature points are used for:
  - Image alignment
  - 3D reconstruction (robots, drones, AR)
  - Motion tracking (robots, drones, AR)
  - Indexing and database retrieval
  - Object recognition

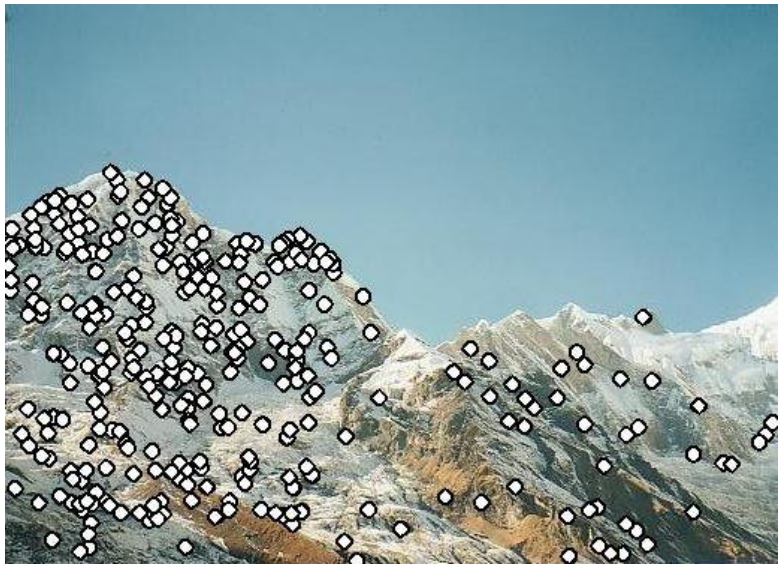
Example application

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

Example application

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

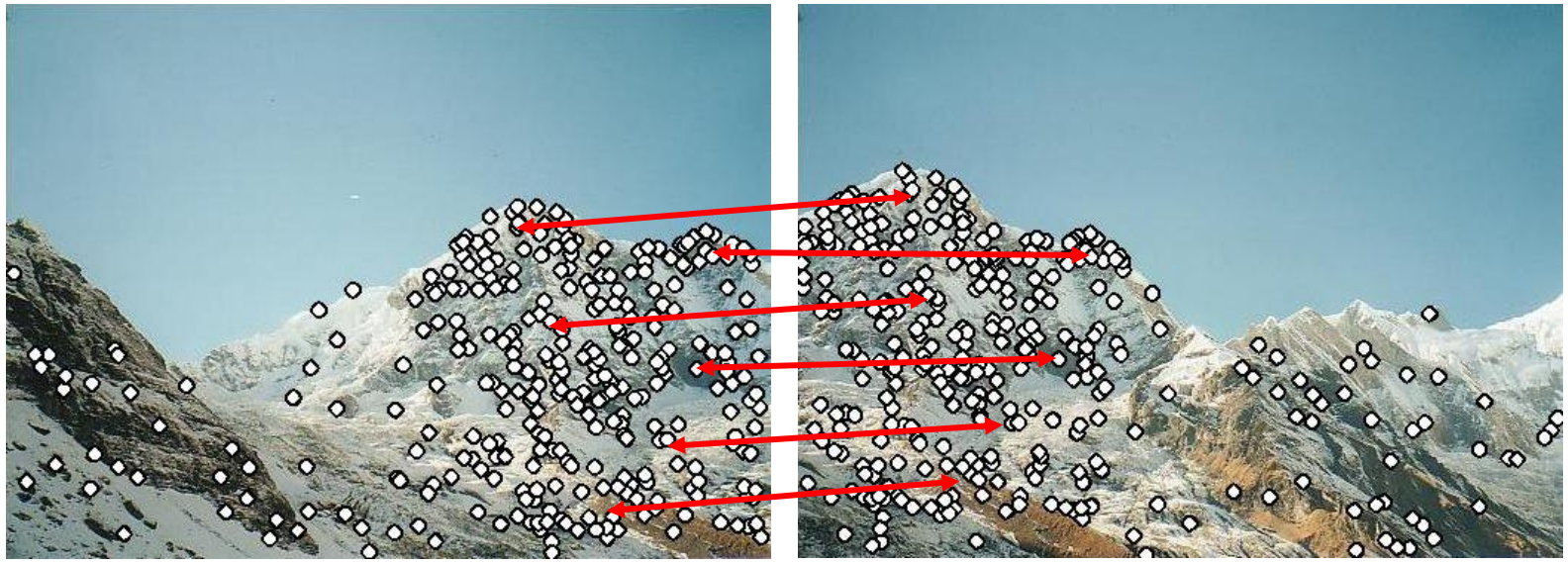




Step 1: extract keypoints

Example application

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



Step 1: extract keypoints

Step 2: match keypoint features

Example application

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

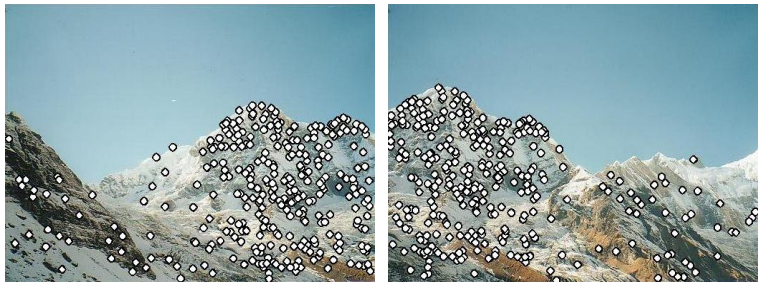
Step 1: extract keypoints

Step 2: match keypoint features

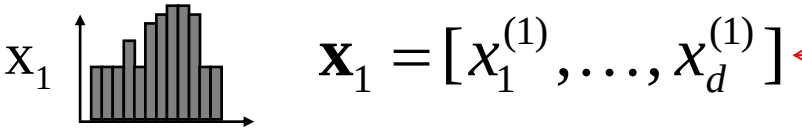
Step 3: align images

Local features: main components

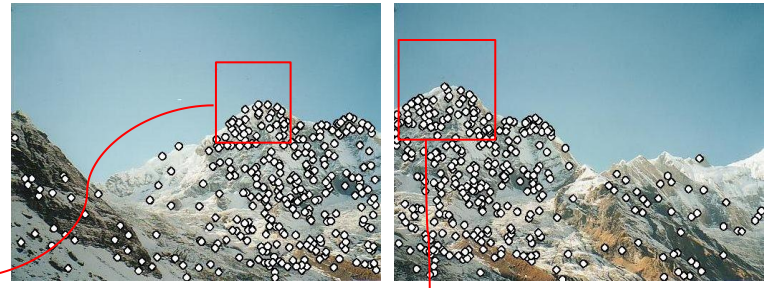
1) Detection:
Find a set of distinctive keypoints.

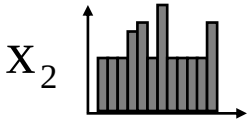


2) Description:
Extract feature descriptor around each interest point as vector.



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

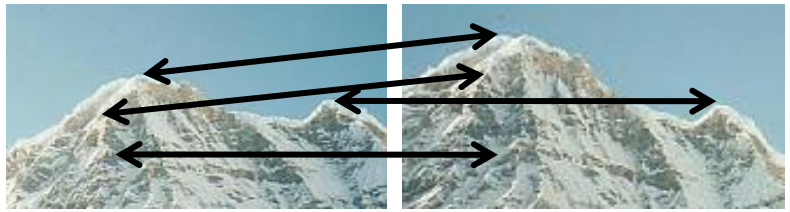




$$\mathbf{x}_2 = [x_1^{(2)}, \ldots, x_d^{(2)}]$$

3) Matching:
Compute distance between feature vectors to find correspondence.

$$d(\mathrm{x}_1, \mathrm{x}_2) < T$$

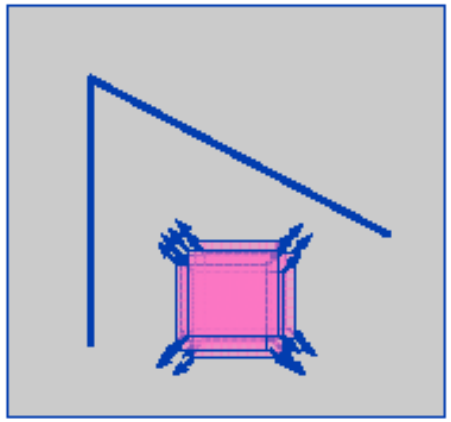


Detection: characteristics of good features

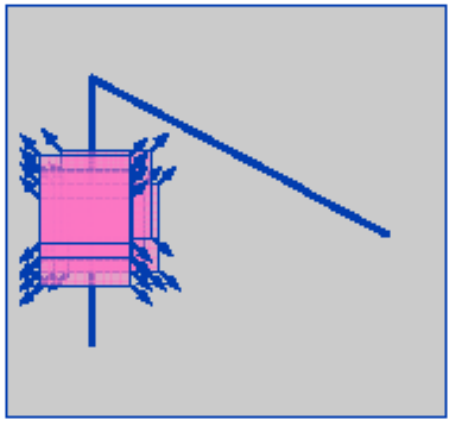
- Image feature extraction needs to be repeatable and accurate
  - Invariant to translation, rotation, scale changes
  - Robust or covariant to out-of-plane (affine) transformations
  - Robust to lighting variations, noise, blur, quantization
- Properties of image features:
- Locality: Features are local, therefore robust to occlusion and clutter.
- Quantity: We need a sufficient number of regions to cover the object and less than image pixels.
- Distinctiveness: The regions should contain “interesting” structure.
- Efficiency: Close to real-time performance.

Corner Detection: Basic Idea

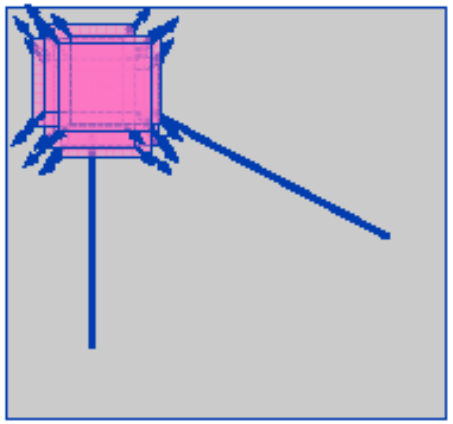
- We do not know which other image locations the feature will end up being matched against. *We know they attract attention: they differ from surroundings*.
- Thus, we can compute how stable a location is in appearance with respect to small variations in position (*u,v)*.
- *Compare image patch against local neighbors.*



"flat" region: no change in all directions



"edge": no change along the edge direction



"corner": significant change in all directions

Harris corner detector gives a *mathematical approach* for determining which case holds

Harris corner detector

- To measure the change of intensity for the shift (*u,v*):

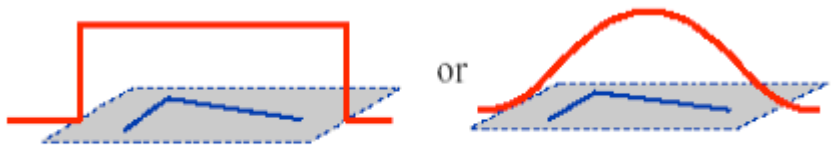
Shifted intensity

Intensity

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

For nearly constant patches, this does not change.
For very distinctive patches, this changes a lot.
Hence, we want patches where E(u,v) changes (in all directions).

Window function w =



1 in window, 0 outside

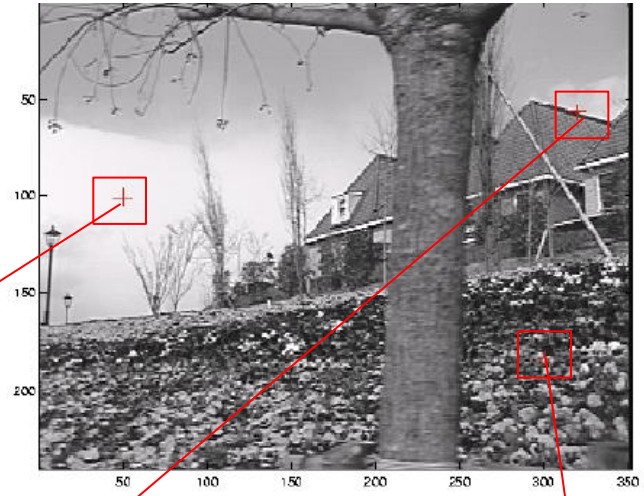
or

Gaussian

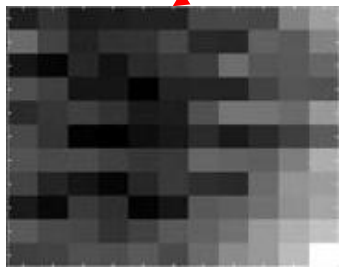
Harris corner detector

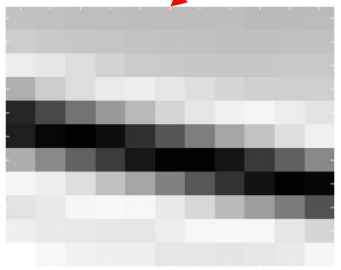
$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u, y+v) - I(x,y)]^2$$

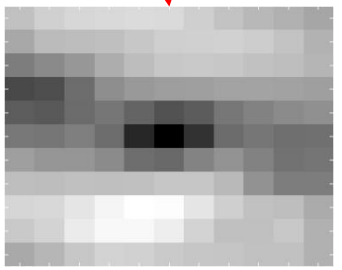
We want to discover how *E(u,v)* behaves for small shifts



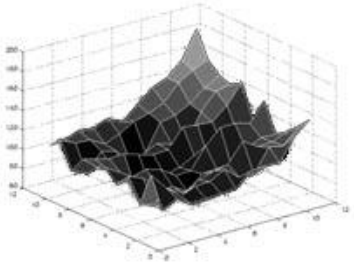
$E(u,v)$



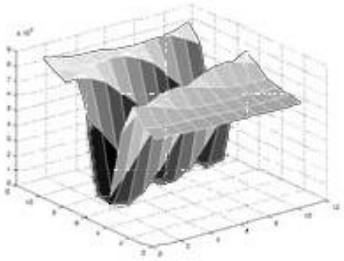




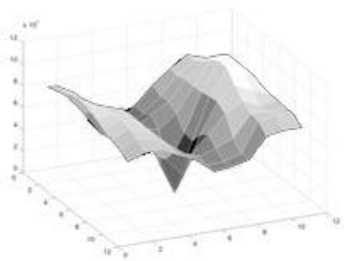
$E(u,v)$ (plotted as a surface)



Flat region



Edge point



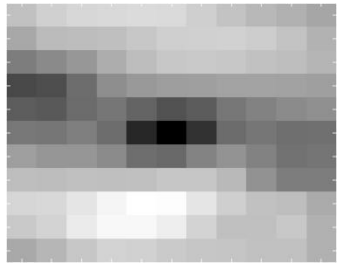
Good point

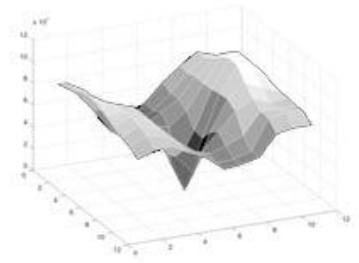
Harris corner detector

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

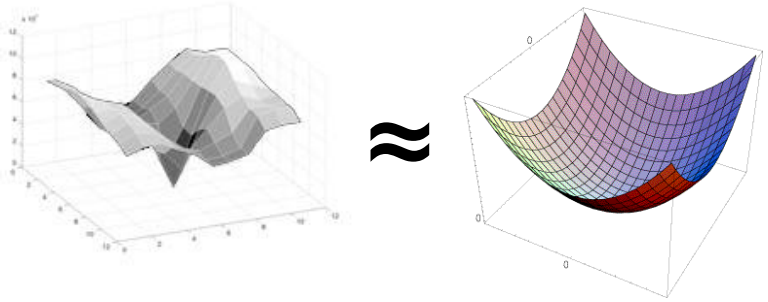
We want to discover how *E(u,v)* behaves for small shifts

Good point:
a corner point





Can we just **approximate** *E*(*u*,*v*) locally by a quadratic surface?



Taylor series expansion

First-order **Taylor** approximation for small shifts [*u, v*]:

$$I(x+u, y+v) \approx I(x,y) + I_x u + I_y v$$

$$E(u,v) = \sum w(x,y)[I_x u + I_y v]^2 = \sum I_x^2 u^2 + 2 I_x I_y uv + I_y^2 v^2$$

w(x,y)

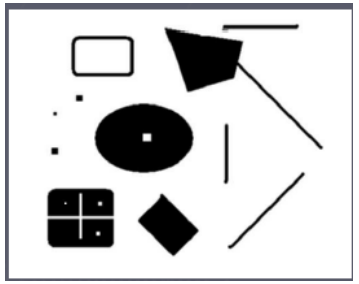
$$E(u,v) \approx [u \;\; v] M \begin{bmatrix} u \\ v \end{bmatrix}$$

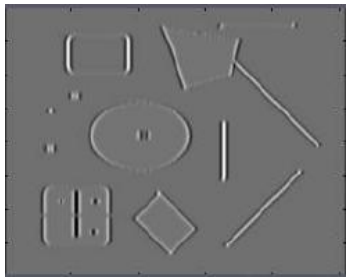
Harris corner detector

- Corners as distinctive interest points

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

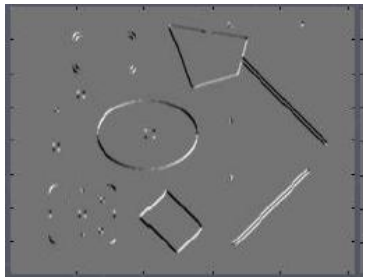
2 x 2 matrix of **image derivatives**
(averaged in neighborhood of a point)





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

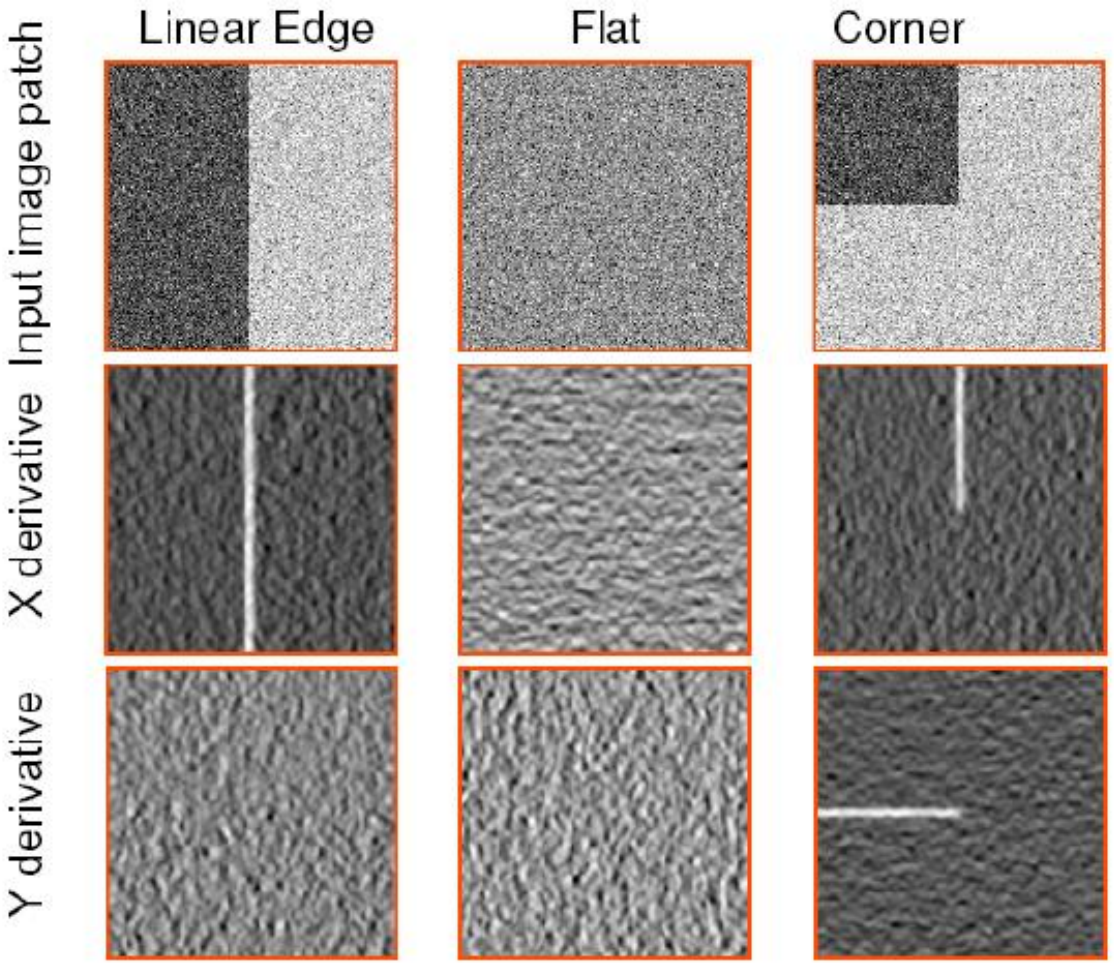
$$I_y \Leftrightarrow \frac{\partial I}{\partial y}$$

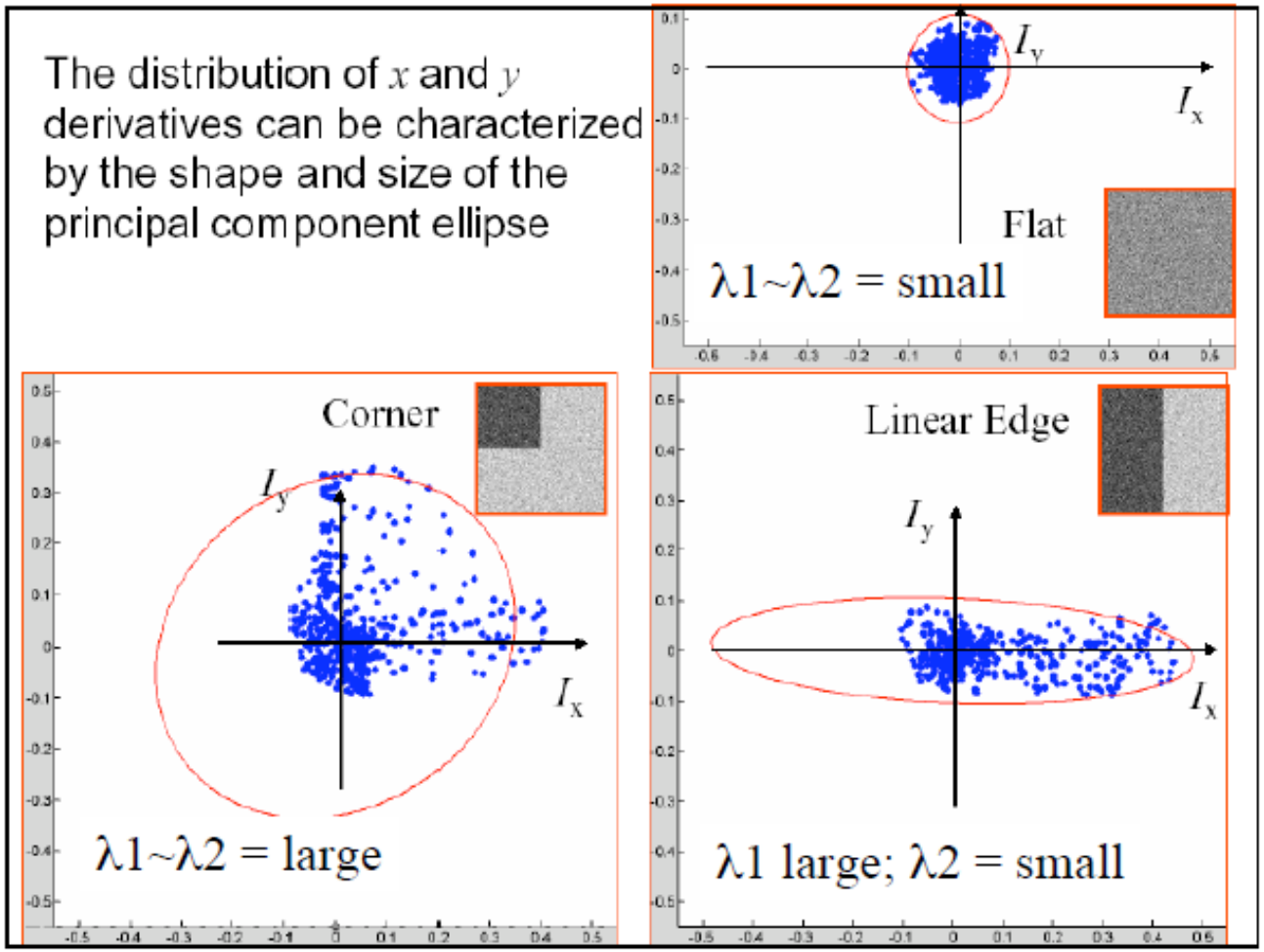


$$I_x I_y \Leftrightarrow \frac{\partial I}{\partial x}\frac{\partial I}{\partial y}$$

Harris corner detector: cases and 2D Derivatives

The **behavior of M matrix** is most easily understood by plotting the ellipses of the set of **gradient vectors** as points





Harris corner detector

- ***"flat" region***: both *eigenvalues* of this matrix will be small, because all terms will be small.

- ***"edge" region***: we expect to see *one large eigenvalue* associated with gradients at the edge and one small eigenvalue, because few gradients will run in other directions.

- ***"corner" region***: *both eigenvalues* will be large

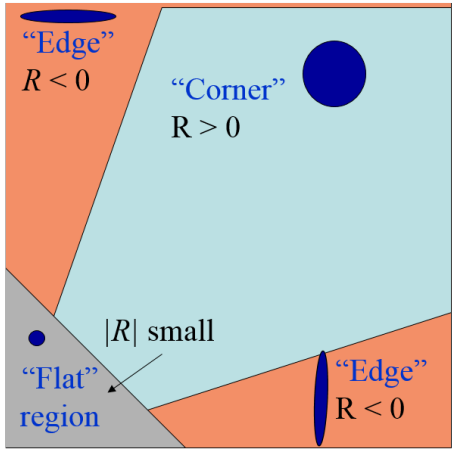
- **Algorithm steps**:
  - Compute M matrix within all image windows to get their R scores
  - Find points with large corner response (R > threshold)
  - Take the points of local maxima of R

Cornerness

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

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

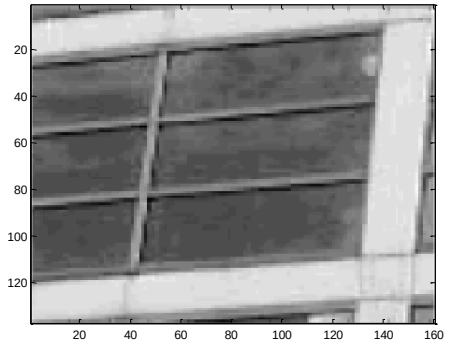
α: constant (0.04 to 0.06)



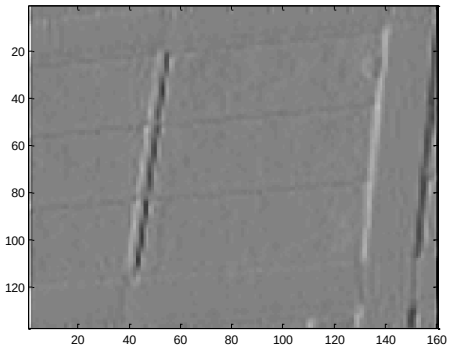
C.Harris and M.Stephens. "A Combined Corner and Edge Detector." *Proceedings of the 4th Alvey Vision Conference*: pages 147—151, 1988.

Harris corner detector: implementation

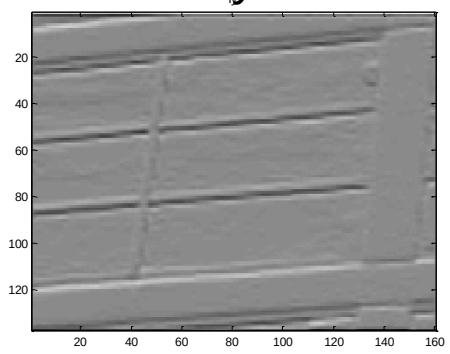
image



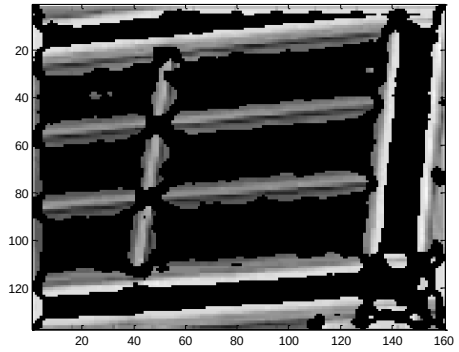
Ix



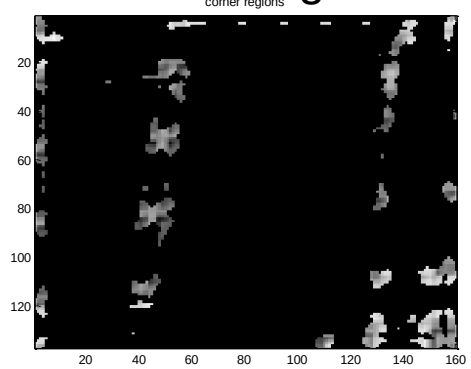
Iy



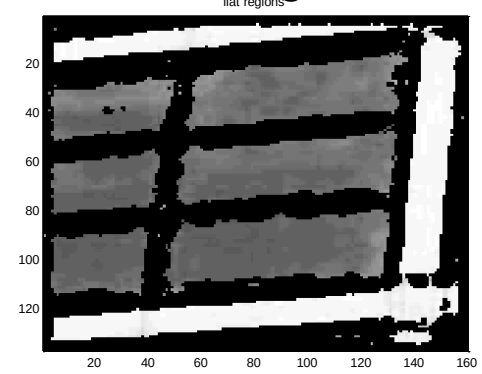
Edge regions



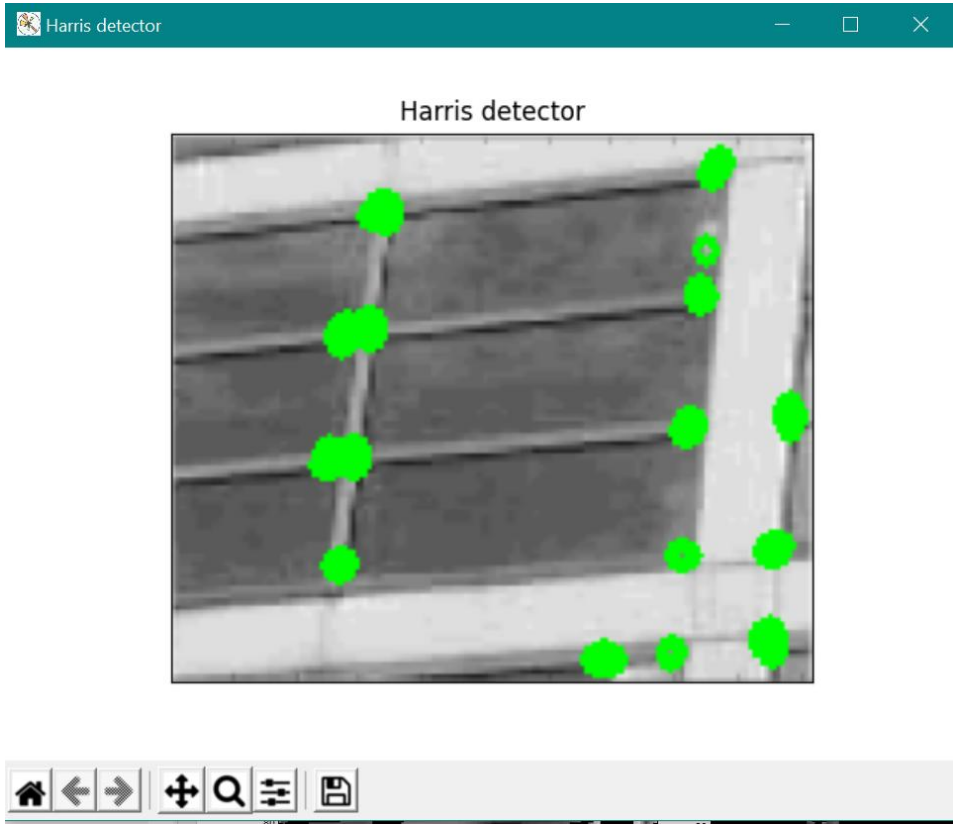
Corner regions



Fat regions



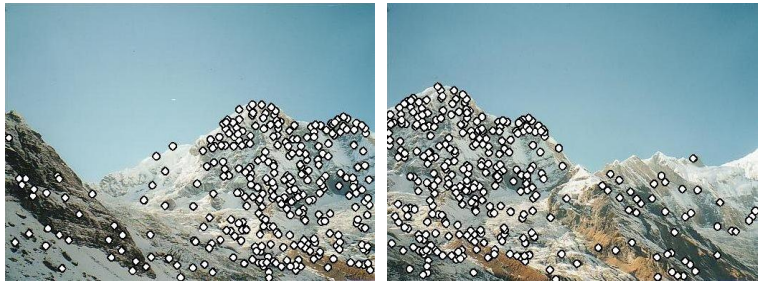
See Harris_Corner_detector.m



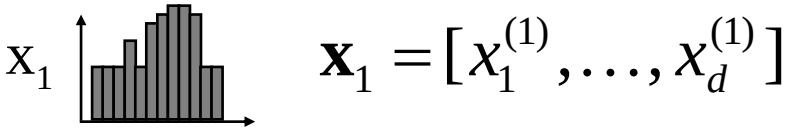
See Harris_Corner_detector.py

Local features: main components

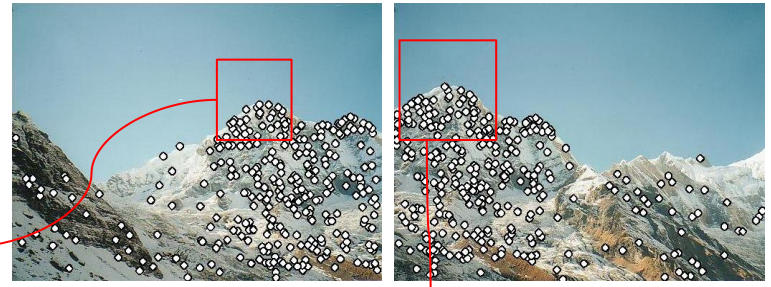
1) Detection:
Find a set of distinctive keypoints.

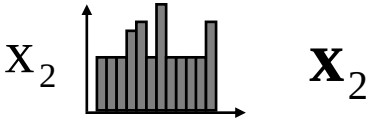


2) Description:
Extract feature descriptor around each interest point as vector.



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

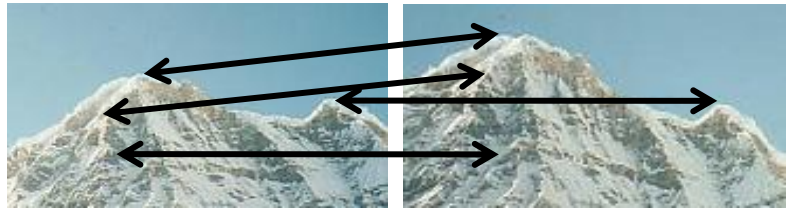




$$\mathbf{x}_2 = [x_1^{(2)}, \ldots, x_d^{(2)}]$$

3) Matching:
Compute distance between feature vectors to find correspondence.

$$d(\mathrm{x}_1, \mathrm{x}_2) < T$$

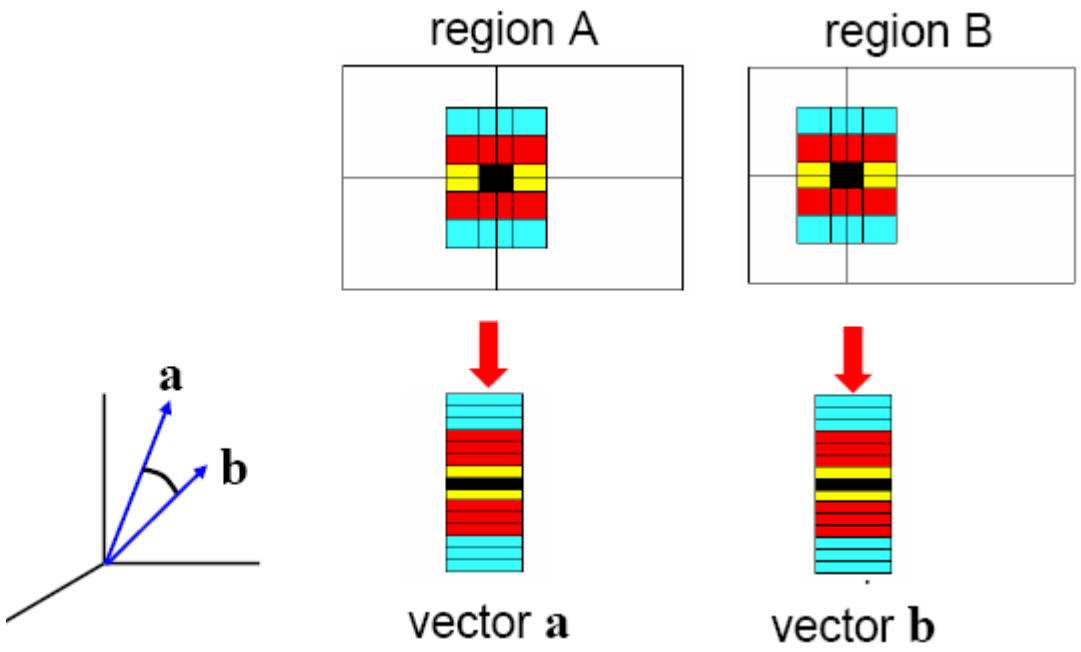


Local descriptors

- **Simplest descriptor**: The simplest way to describe the neighborhood around an interest point is to write down the list of intensities to form a feature vector
- But this is very sensitive to even small shifts, or rotations.

Write regions as vectors

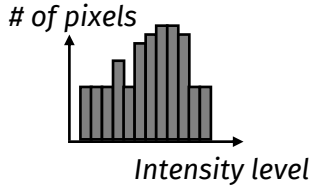
$$A \rightarrow \mathbf{a}, \quad B \rightarrow \mathbf{b}$$



Local descriptors

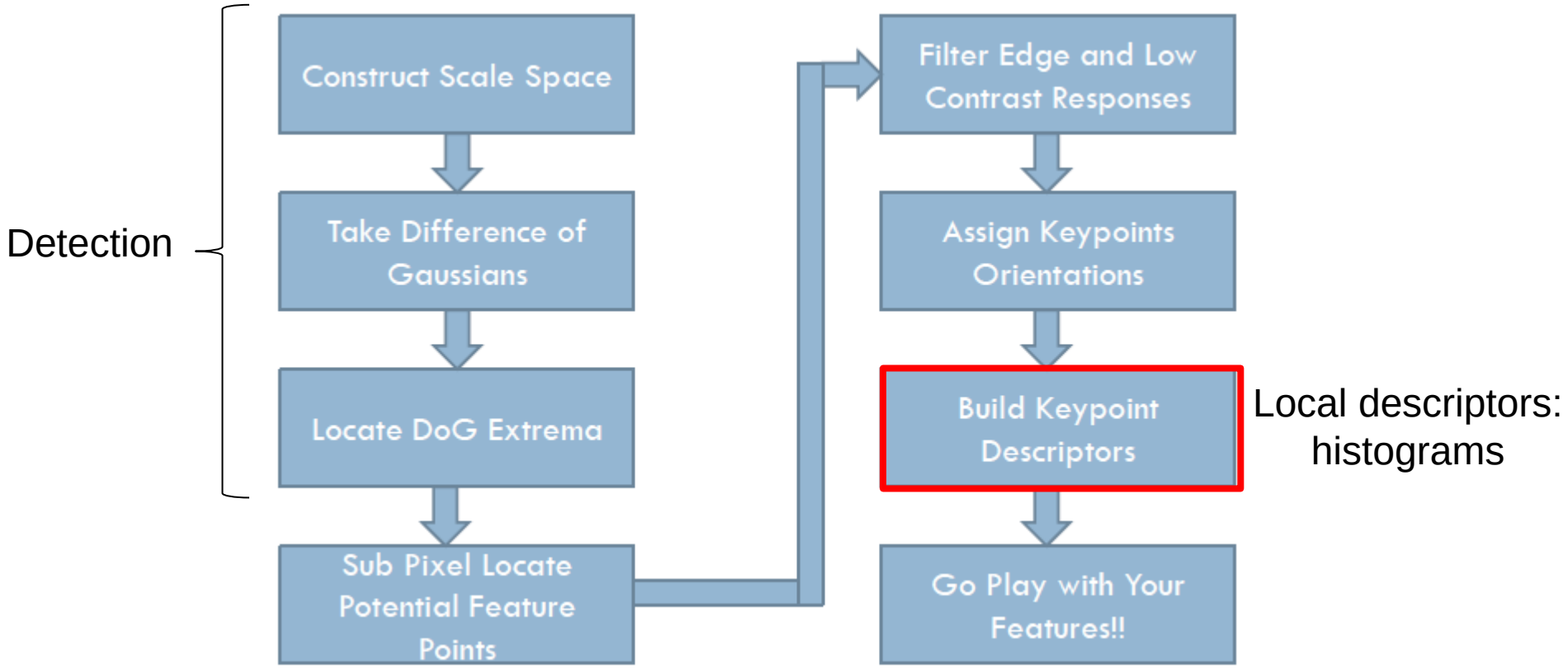
- **Template**: intensity patches around an interest point as descriptor by using *cross-correlation* (*see next slides*)
  - Small shifts can affect matching score a lot

- **Histograms**: (local) histograms around an interest point can be more stable



SIFT: overview of the algorithm

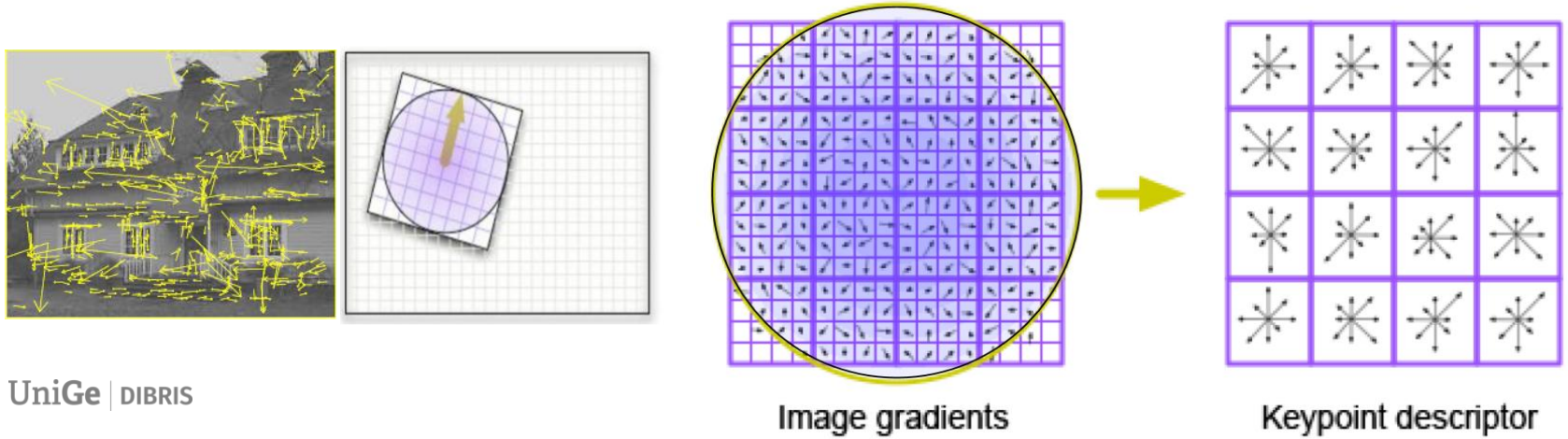
Scale **I**nvariant **F**eature **T**ransform



David G. Lowe. "Distinctive image features from scale-invariant keypoints." *International Journal of Computer Vision*, 60 (2), pp. 91-110, 2004.

SIFT: overview of the algorithm

- Find Difference of Gaussian scale-space extrema
- Post-processing (Discard low-contrast points and edges)
- Orientation estimation (compute orientation histogram)
- Descriptor extraction
  - Divide patch into 4x4 sub-patches: 16 cells
  - Compute histogram of gradient orientations (8 reference angles) for all pixels inside each sub-patch
  - Resulting descriptor: 4x4x8 = 128 dimensions



SIFT: overview of the algorithm

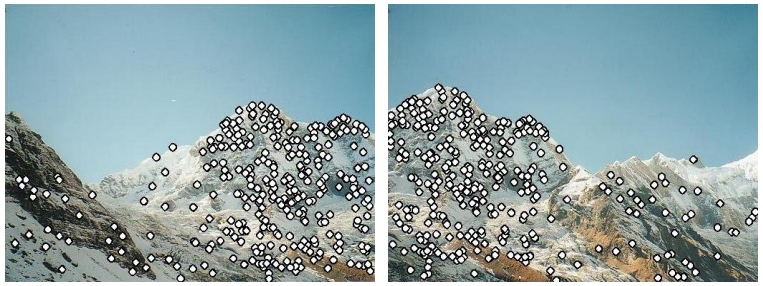
- The SIFT features are invariant to
  - image scaling (*scale-space extrema*) and
  - rotation (*orientation estimation*), and
  - partially invariant to change in illumination and 3D camera viewpoint.

- Illumination invariance:
  - Working in gradient space, so robust to $I = I + b$
  - Normalize vector to [0…1]
    - Robust to $I = \alpha I$ brightness changes

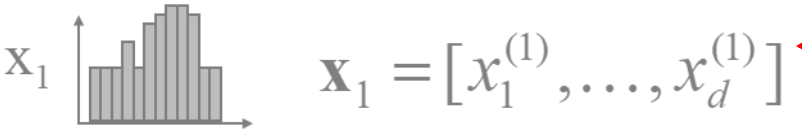
- See also other popular approaches, such as, SURF [Bay et al. 2006], and BRIEF [Calonder et al. 2010].

Local features: main components

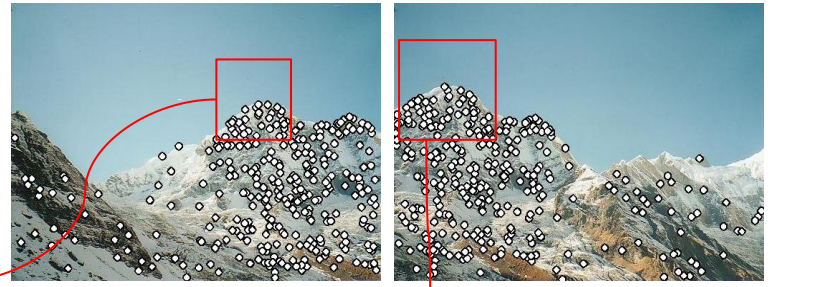
1) Detection:
Find a set of distinctive keypoints.

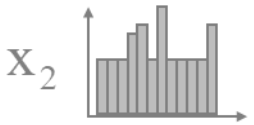


2) Description:
Extract feature descriptor around each interest point as vector.



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

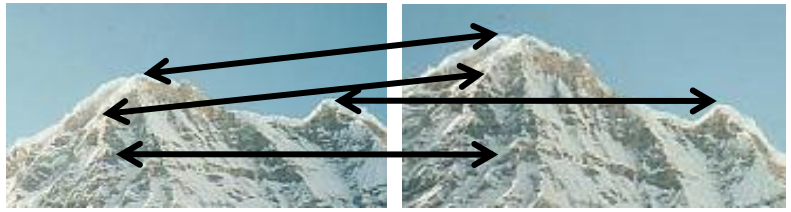




$$\mathbf{x}_2 = [x_1^{(2)}, \ldots, x_d^{(2)}]$$

3) Matching:
Compute distance between feature vectors to find correspondence.

$$d(\mathrm{x}_1, \mathrm{x}_2) < T$$



Matching Problem

- Two classes of algorithms to find correspondences:

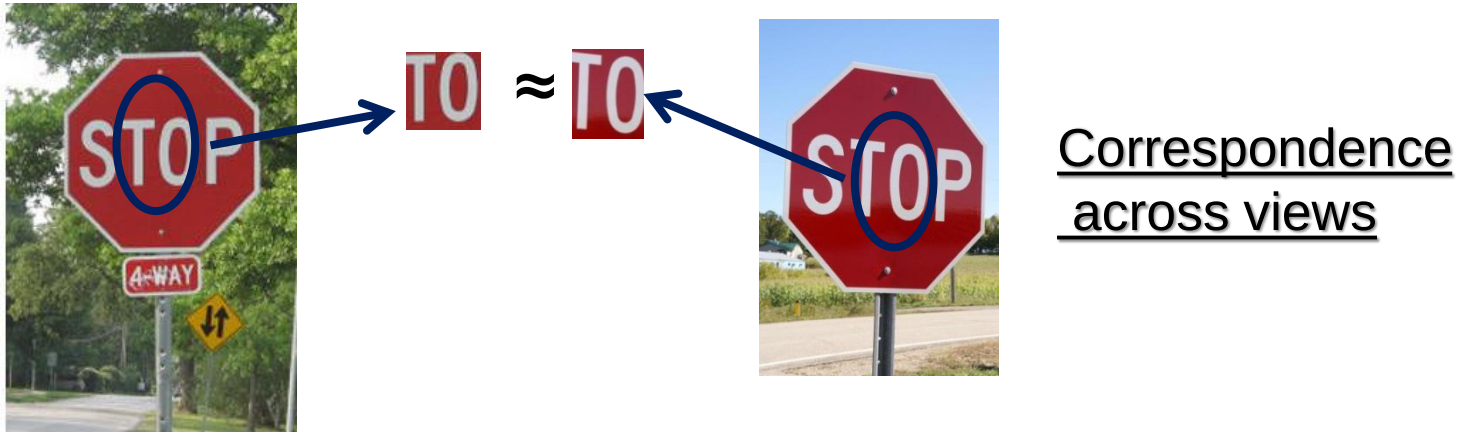
- **Correlation-based** algorithms
  - Produce a DENSE set of correspondences or SPARSE around the interest points
  - Only work where there are textures

- **Feature-based** algorithms
  - Produce a SPARSE set of correspondences around the interest points
  - Can be sensitive to feature “drop-outs”

Intensity Matching (correlation-based algorithms)

Template matching

- We cut little pictures (patches or windows) out from an image, then we convolve them with the other image.



- *If image pairs are "similar enough"*, then image patches are an appropriate technique for matching. Otherwise, other techniques (e.g. the SIFT feature descriptor) are more suitable.
- The template matching is a simple intensity matching.

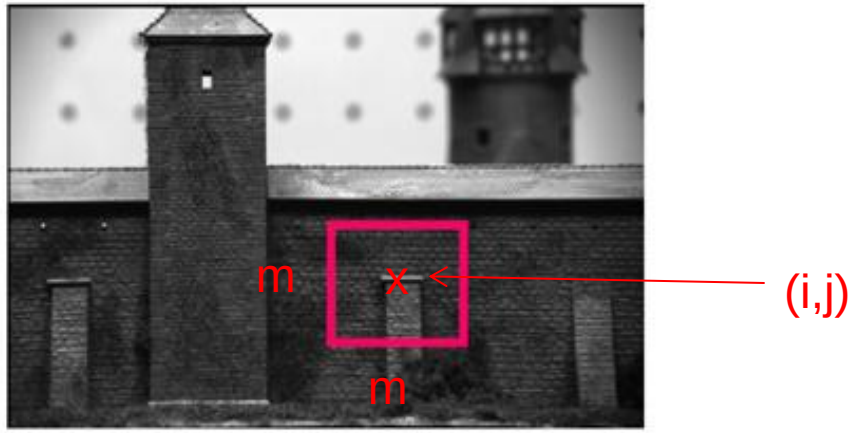
Intensity Matching: correlation

$$C_{fg}(i,j) = \sum_{(u,v)\in R_m} f(u+i, v+j)\, g(u,v)$$

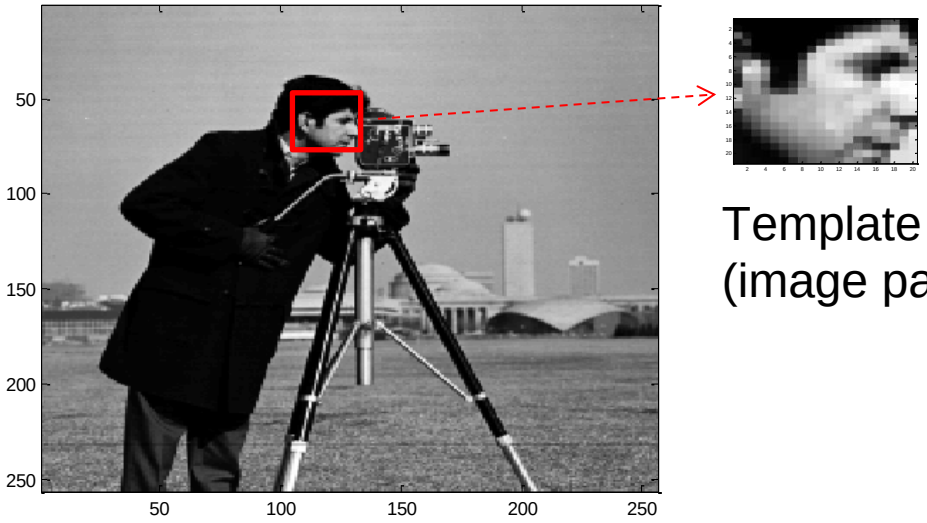
In signal processing, cross-correlation is a measure of similarity

$$R_m(i,j) = \{u,v \mid x - \frac{m}{2} \le u \le x + \frac{m}{2}, y - \frac{m}{2} \le v \le y + \frac{m}{2}$$

If we are doing exhaustive search over all image patches in the same image, this becomes *crosscorrelation* (simplest problem) of a template with an image.



Intensity Matching: correlation



image

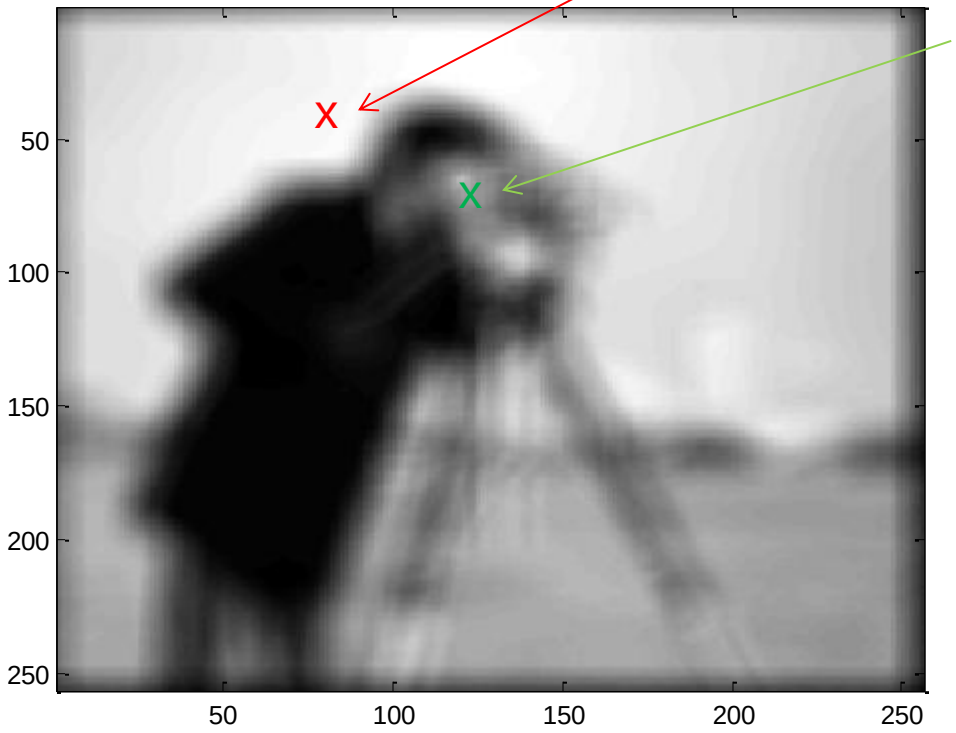
Template
(image patch)

See Template_Matching.m

$$score(i, j) = C_{fg}(i, j)$$

Note that score image looks a lot like a blurry version of image.

This clues us into the problem with straight correlation with an image template.



Score map

Intensity Matching: correlation

Consider a correlation of a template with an image

$$C^1{}_{fg}(i,j) = \sum_{(u,v)\in R_m} f(u+i, v+j)\, g(u,v)$$

Consider a correlation of a template with an image that is *twice* as bright

$$C^2{}_{fg}(i,j) = \sum_{(u,v)\in R_m} f(u+i, v+j)\, 2g(u,v)$$

Larger score, *regardless* of what the template is

$$C^2{}_{fg}(i,j) = 2C^1{}_{fg}(i,j)$$

Thus, subtract off the mean value of the template.

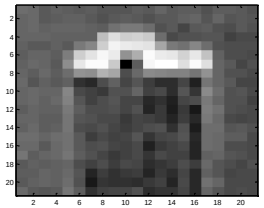
In this way, *the correlation score is higher only when darker parts of the template overlap darker parts of the image, and brighter parts of the template overlap brighter parts of the image*.

Better! But highest score is still not the correct match (see next slide), since we should also consider the *local image contrast*

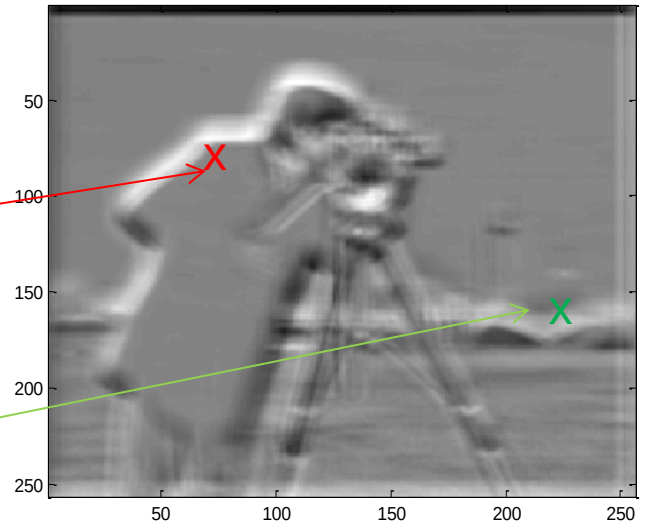
Note: highest score is best within local neighborhood of correct match.

Intensity Matching: correlation

image



zero-mean template
(image patch)

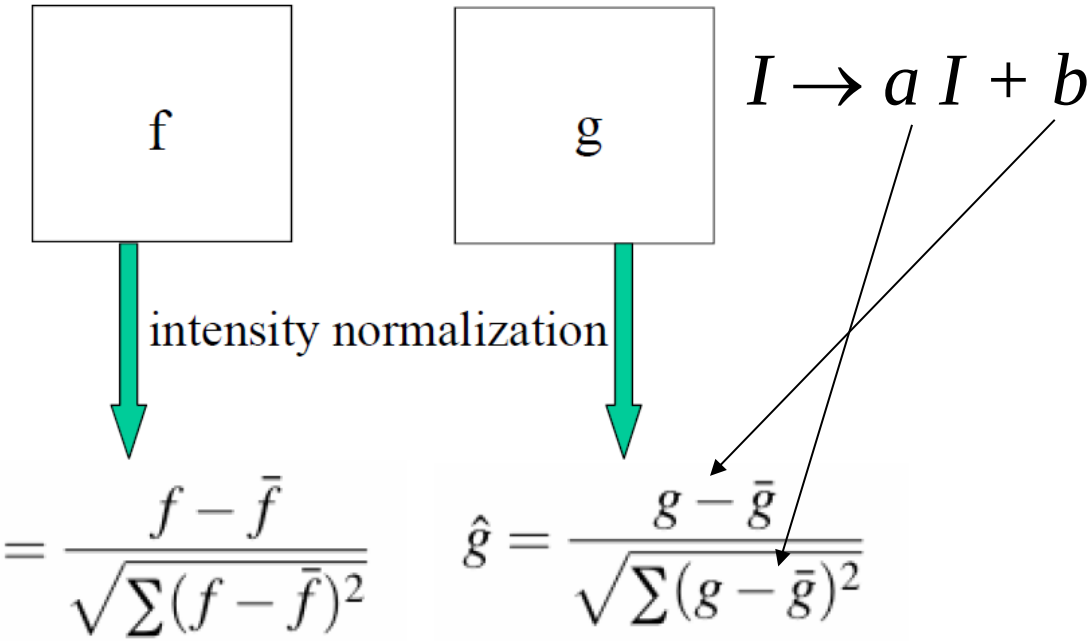


Highest score

Correct match

Score map

Normalization (*by mean and contrast*)
counteracts affine intensity change



$$\mathrm{NCC(f,g)} = C_{fg}(\hat{f}, \hat{g}) = \sum_{[i,j] \in R} \hat{f}(i,j)\hat{g}(i,j)$$

Matching by using vectors of features

- Euclidean distance:

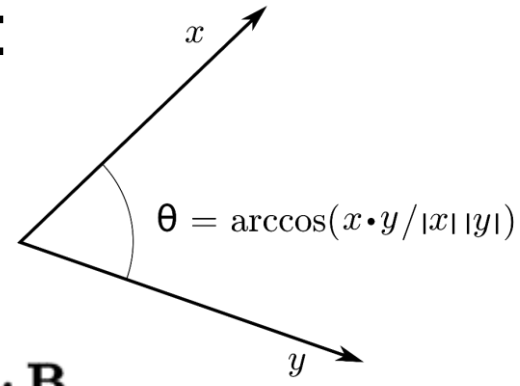
$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \mathrm{d}(\mathbf{q},\mathbf{p}) = \sqrt{(q_1-p_1)^2+(q_2-p_2)^2+\cdots+(q_n-p_n)^2}$$

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

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

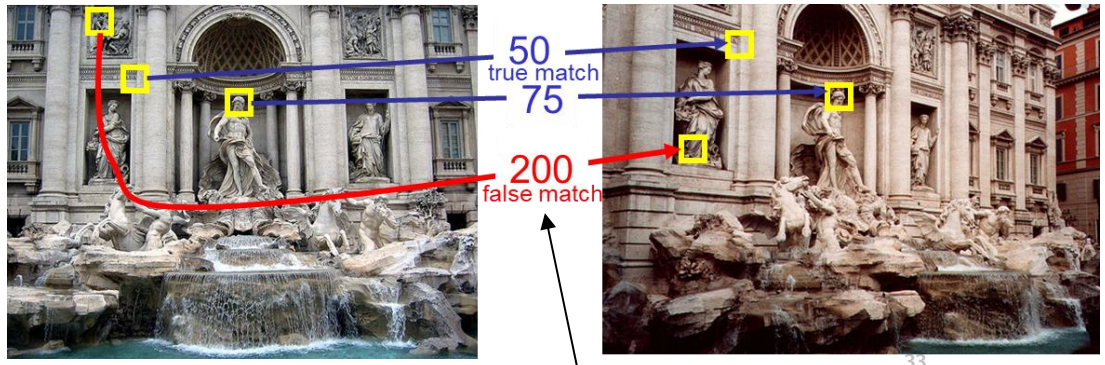
- Cosine similarity:

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



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

- Criteria:
  - Compute distance in feature space, e.g., Euclidean distance between 128-dim SIFT descriptors
  - Match point to lowest distance (nearest neighbor)
  - Ignore anything higher than threshold (no match!)
- Problems:
  - Threshold is hard to pick
  - Non-distinctive features could have lots of close matches, only one of which is correct



Università di Genova
DIBRIS DIPARTIMENTO
DI INFORMATICA, BIOINGEGNERIA,
ROBOTICA E INGEGNERIA DEI SISTEMI

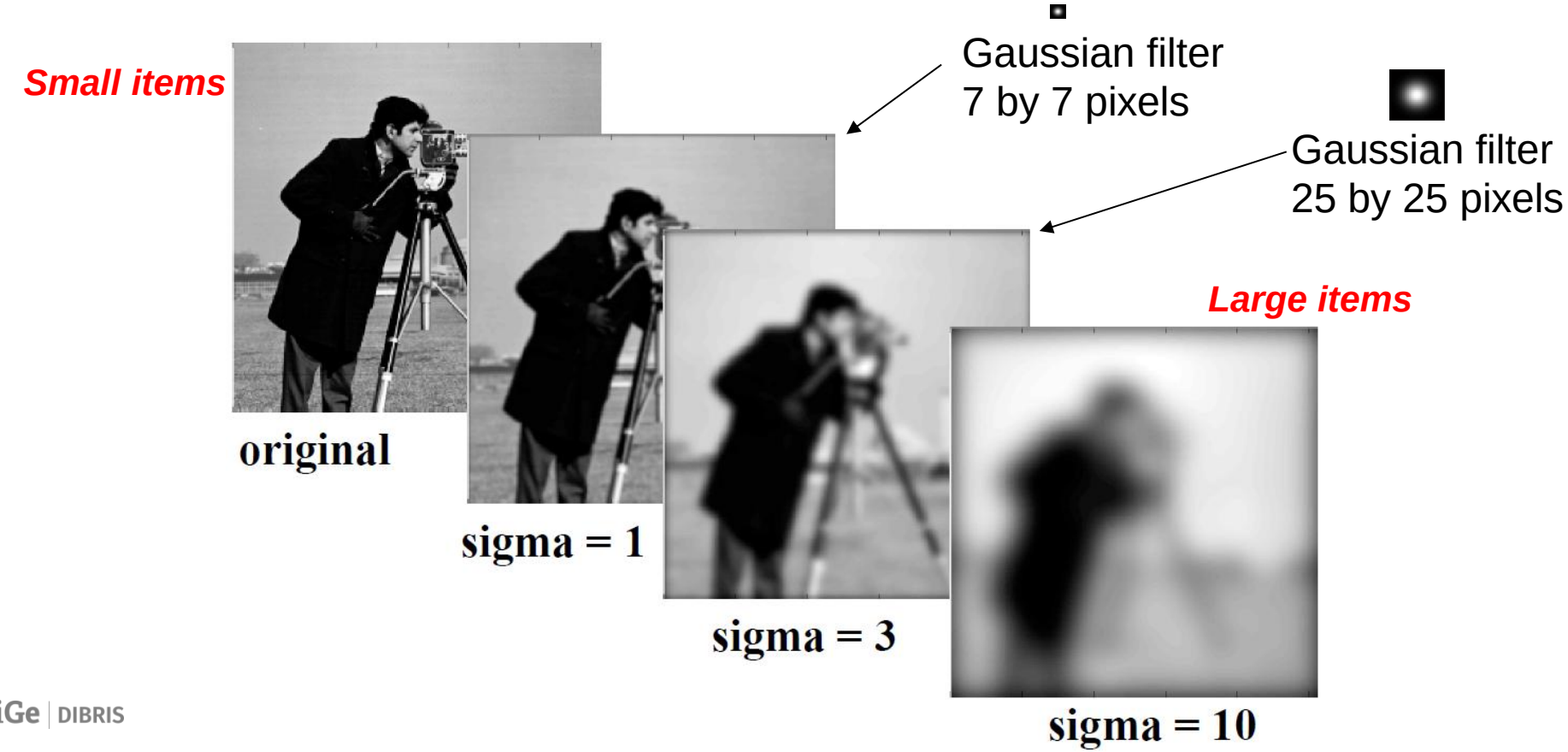
Blobs

Multi-Resolution Representation

- We may wish to change the resolution of an image before proceeding further:
  - We may need to interpolate a small image to make its resolution match the desired one.
  - We may want to reduce the size of an image to speed up the execution of an algorithm.
  - Sometimes, we do not even know what the *appropriate resolution* for the image should be: e.g. the task of finding a face in an image (template size is constant, but image size changes). Since we do not know the scale at which the face will appear in the image, we need to generate a whole pyramid (set) of differently sized images and scan each one for possible faces.

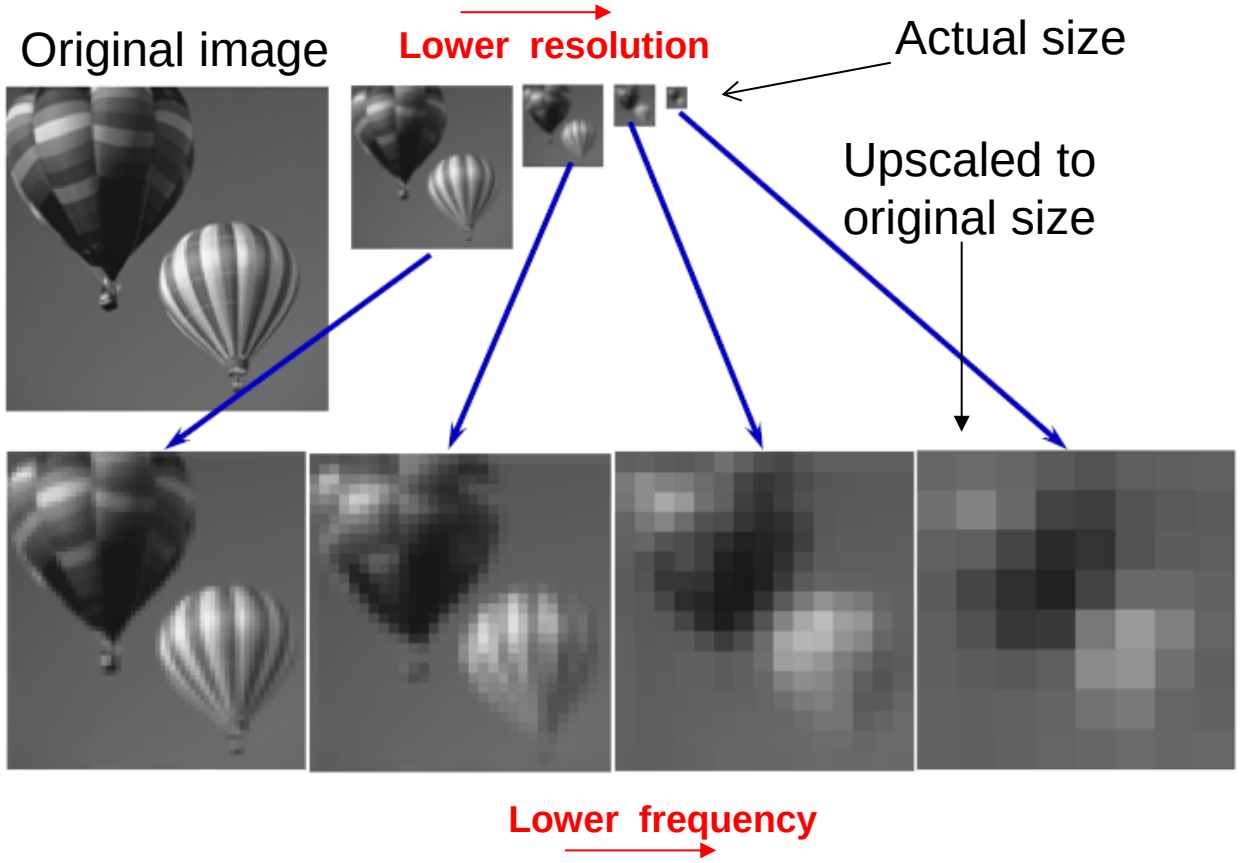
Multi-Resolution Representation: scale

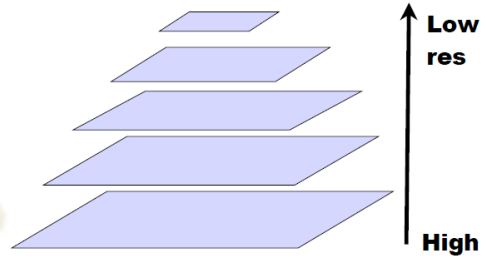
- Gaussian smoothing at different scales forms a <u>multi-resolution representation:</u> different content of frequency in the images (*lower frequency for larger sigma*), i.e. **different details** too.



Pyramid Representation

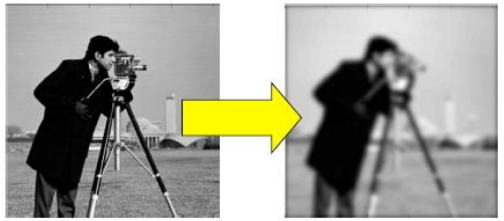
Because a large amount of smoothing limits the frequency content of features in the image, we do not need to keep all the pixels around!



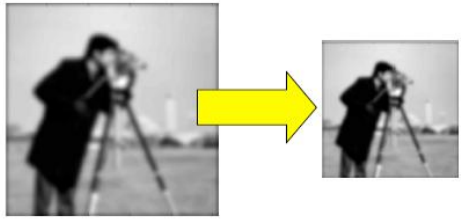


Basic Functions:

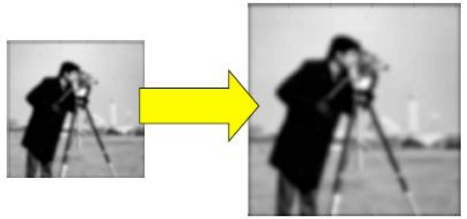
Blur (convolve with Gaussian to smooth image)



DownSample (reduce image size by half)



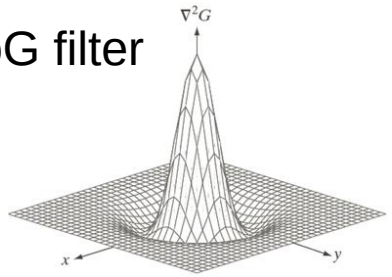
Upsample (double image size)



Laplacian of Gaussian (LoG) Filter

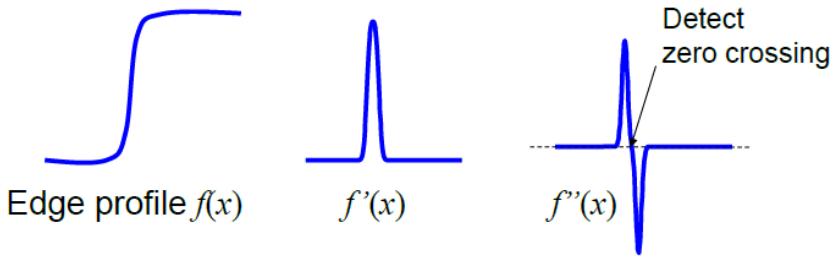
$$\nabla^2 f(x,y) = \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2}$$

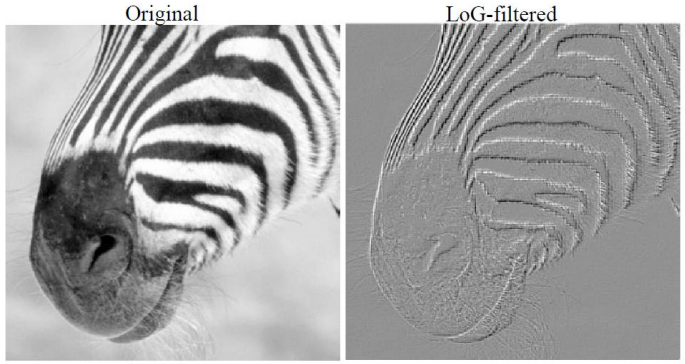
LoG filter



Where f(x,y) is a Gaussian

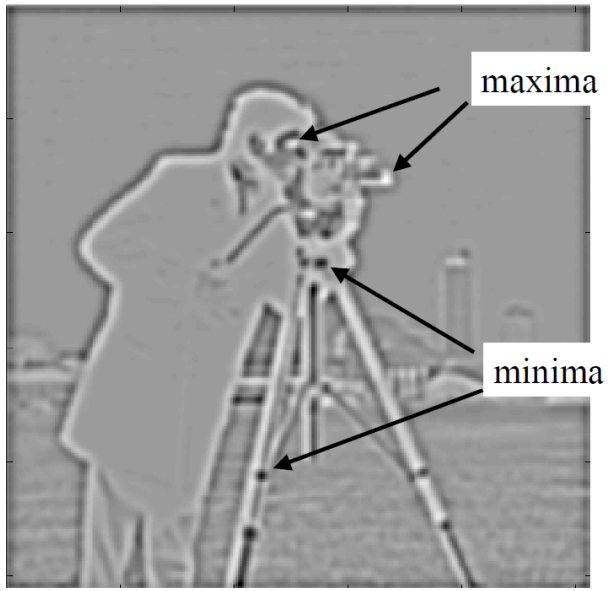
Zero-Crossings as an Edge Detector



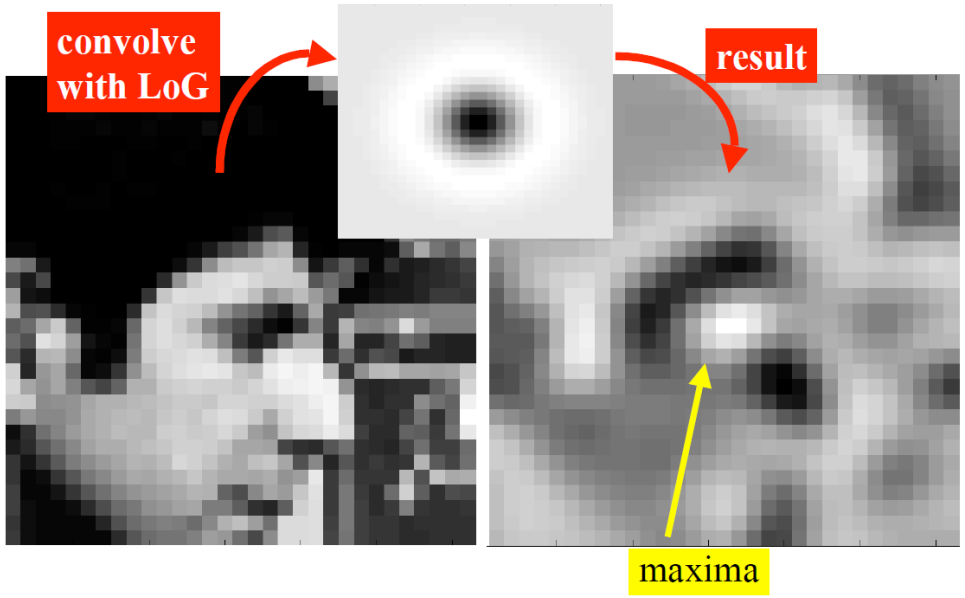


How can an edge finder also be used to find blobs in an image?

LoG
sigma = 2

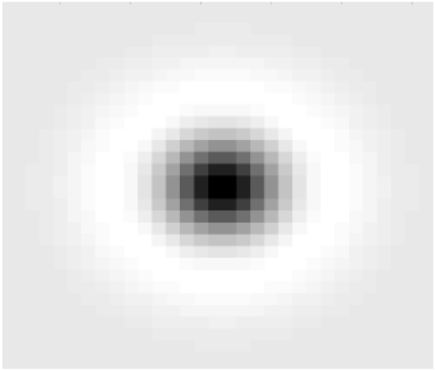


LoG filtered image



LoG filter: Blob Detection

LoG looks a bit like an eye.



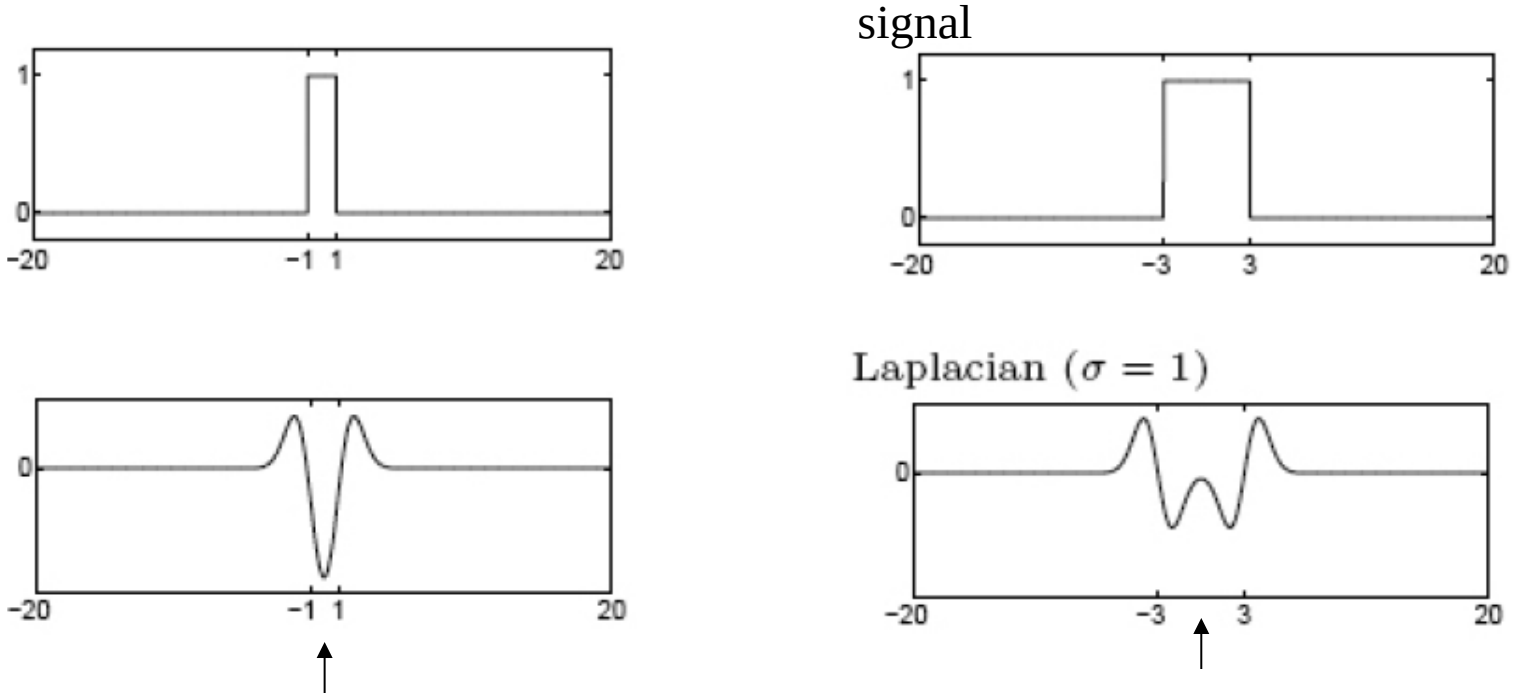
and it responds maximally in the eye region!

Blob detection extracts regions that differ in properties compared to surrounding regions

Convolution (and cross correlation) with a filter can be viewed as comparing a little "picture" of what you want to find against all local regions in the mage.

Blob detection

Blob = superposition of nearby edges



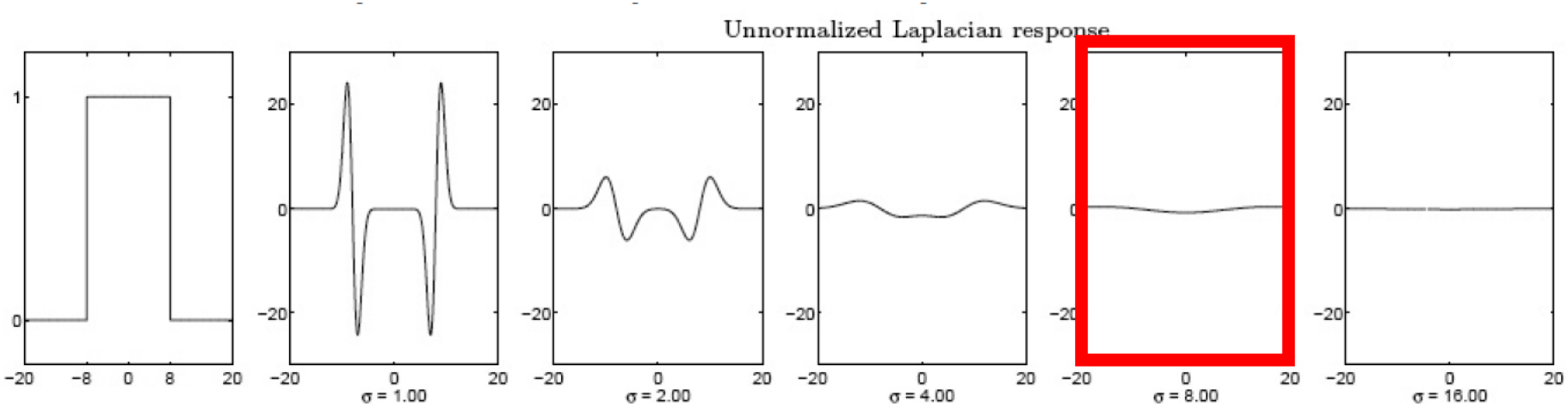
Maximum (abs of)

No longer maximum (abs of)

Scale selection: magnitude of the Laplacian response will achieve a maximum at the center of the blob, *provided the scale of the Laplacian is "matched" to the scale of the blob*.

Scale selection

We want to find the **characteristic scale** of the blob by convolving it with Laplacians at several scales and looking for the maximum response.



Signal

Increasing σ →

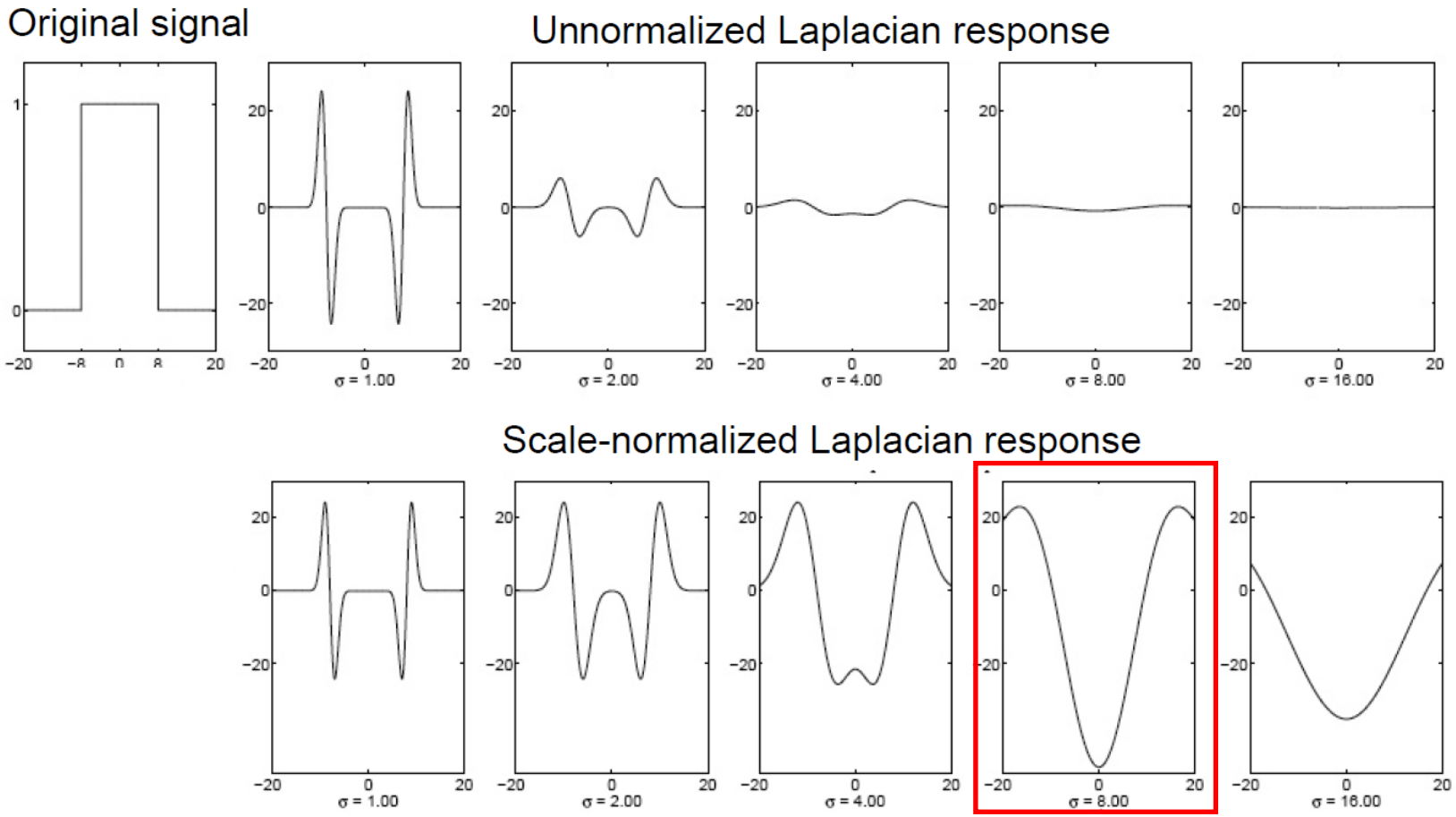
This should give the max response

The response decreases as σ increases: to obtain scale normalization, it must be multiplied by σ^2.

$$\nabla^2 g = \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2} = \frac{1}{2\pi\sigma^2}\left(\frac{x^2+y^2-2\sigma^2}{\sigma^4}\right)e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Scale-normalized: $\nabla^2_{\text{norm}} g = \sigma^2\left(\frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2}\right)$

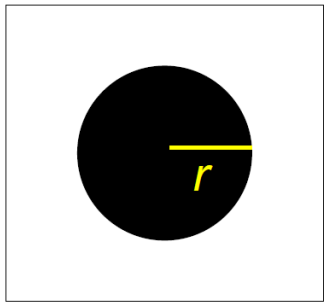
Scale selection: scale normalization



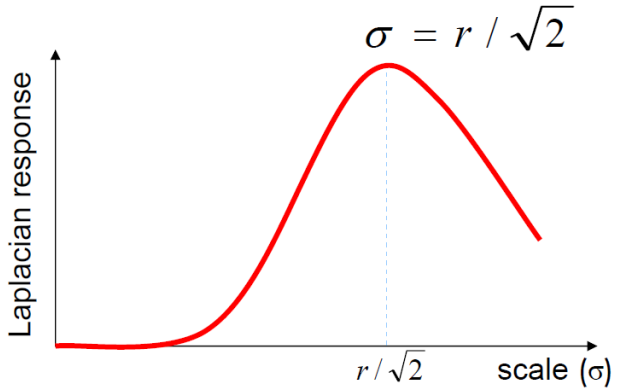
This is the max response

Characteristic scale

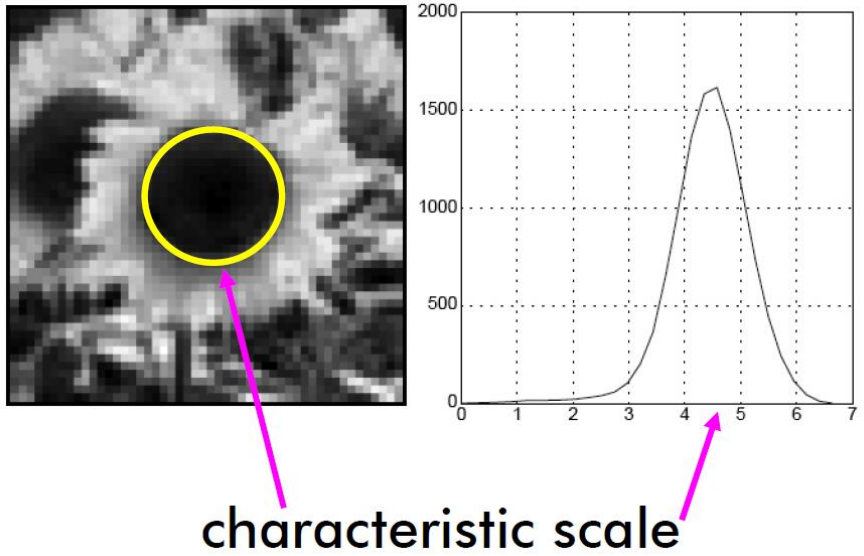
For a binary circle of radius r, the Laplacian achieves a maximum at $\sigma = r / \sqrt{2}$



image



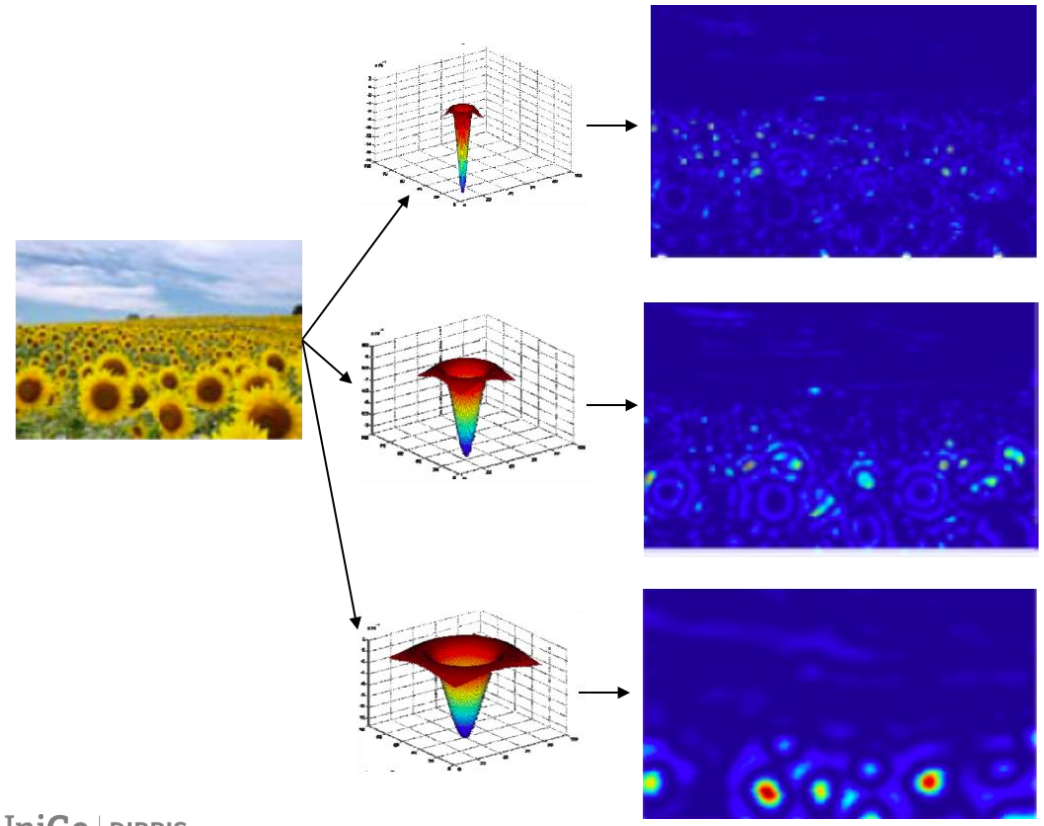
We define the **characteristic scale** as the scale that produces peak of Laplacian response.

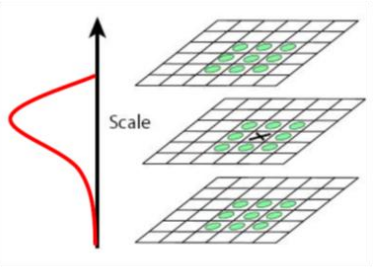


T. Lindeberg (1998). "Feature detection with automatic scale selection". International Journal of Computer Vision, 30(2): pp 77--116

Scale-space blob detector

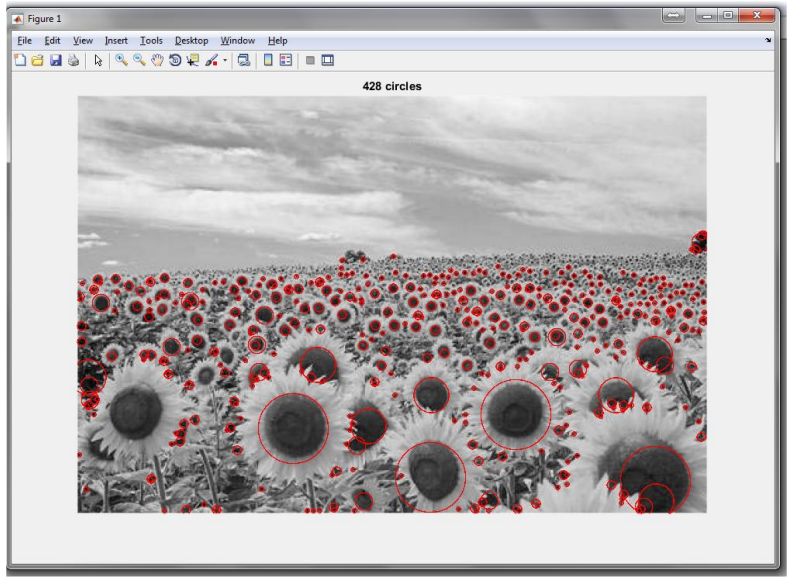
1. Convolve image with scale-normalized Laplacian at several scales
2. Find maxima of squared Laplacian response in scale-space
3. Non-maxima suppression in scale-space





List of (x, y, σ)

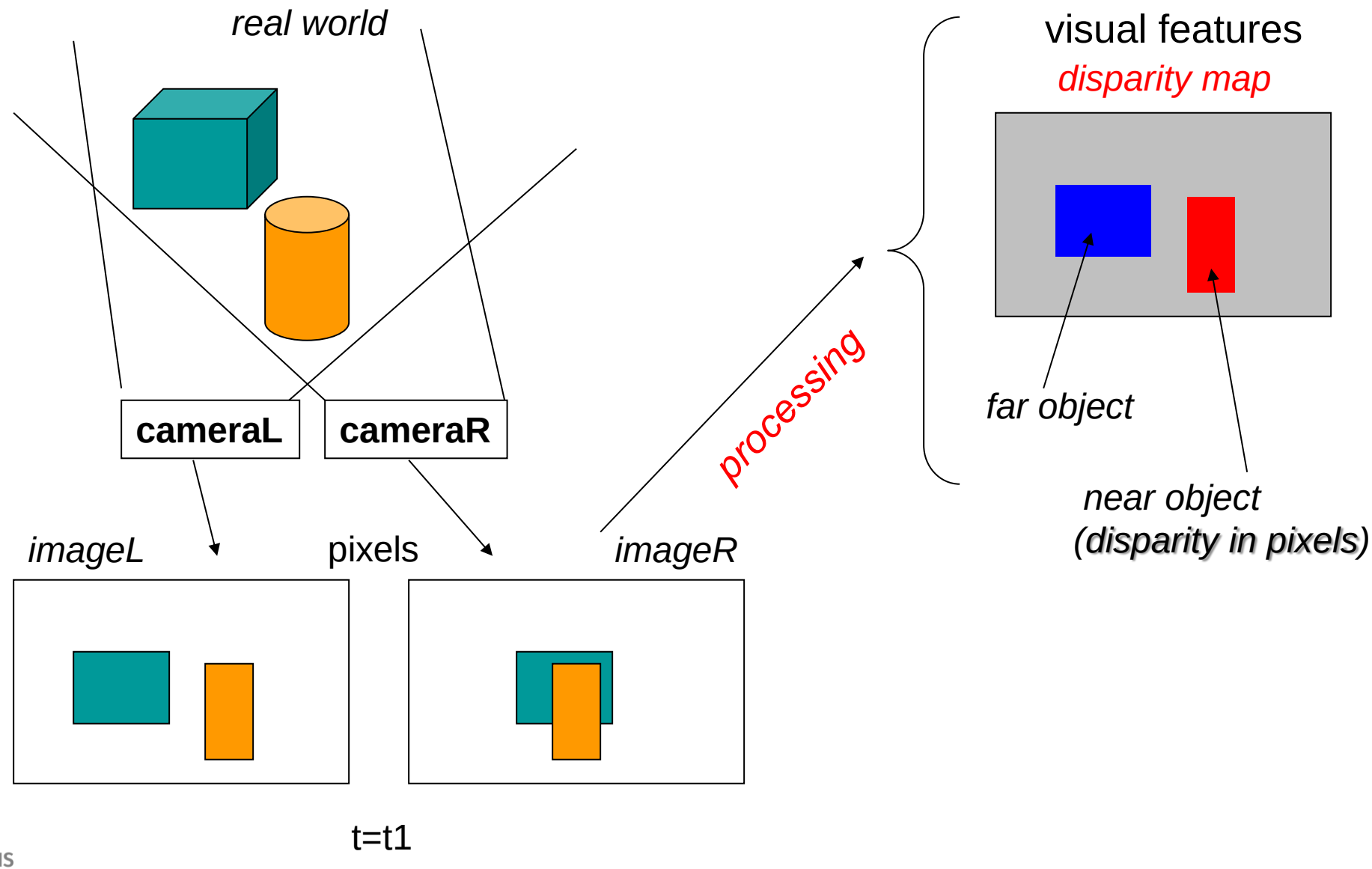
See blobs_detection.m



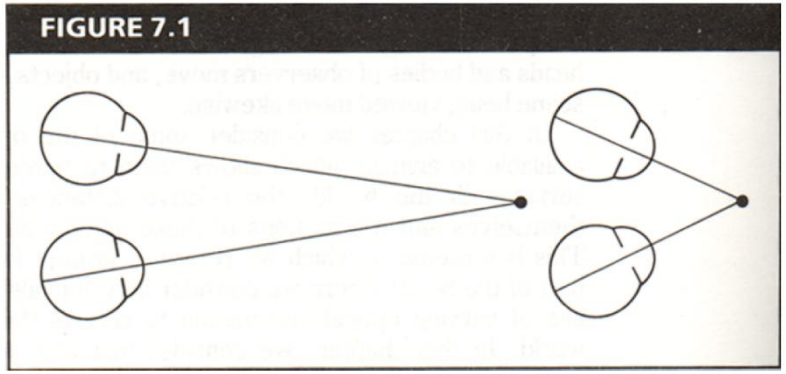
Università di Genova
DIBRIS DIPARTIMENTO
DI INFORMATICA, BIOINGEGNERIA,
ROBOTICA E INGEGNERIA DEI SISTEMI

Disparity (stereo images)

Disparity (stereo images): stereopsis



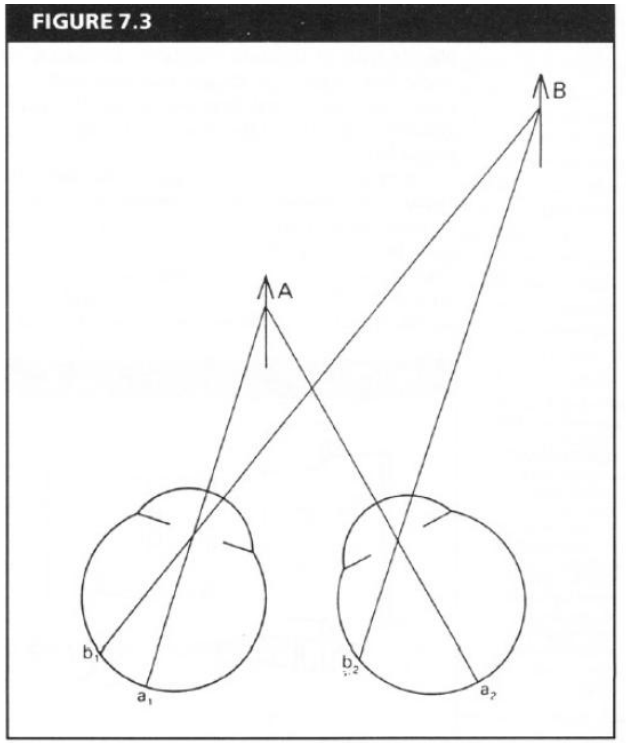
Human stereopsis



From Bruce and Green, Visual Perception, Physiology, Psychology and Ecology

Human eyes **fixate** on one point in 3D space by rotating, so that corresponding images form in centers of **foveas**.

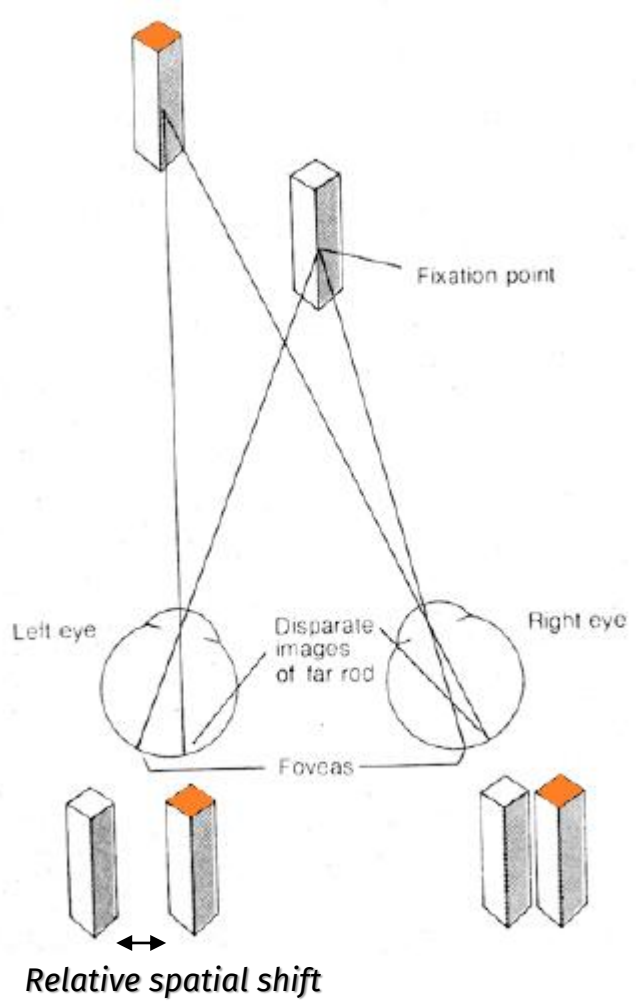
Disparity occurs when eyes fixate on one object; other objects appear at **different visual angles**.



From Bruce and Green, Visual Perception, Physiology, Psychology and Ecology

Human stereopsis

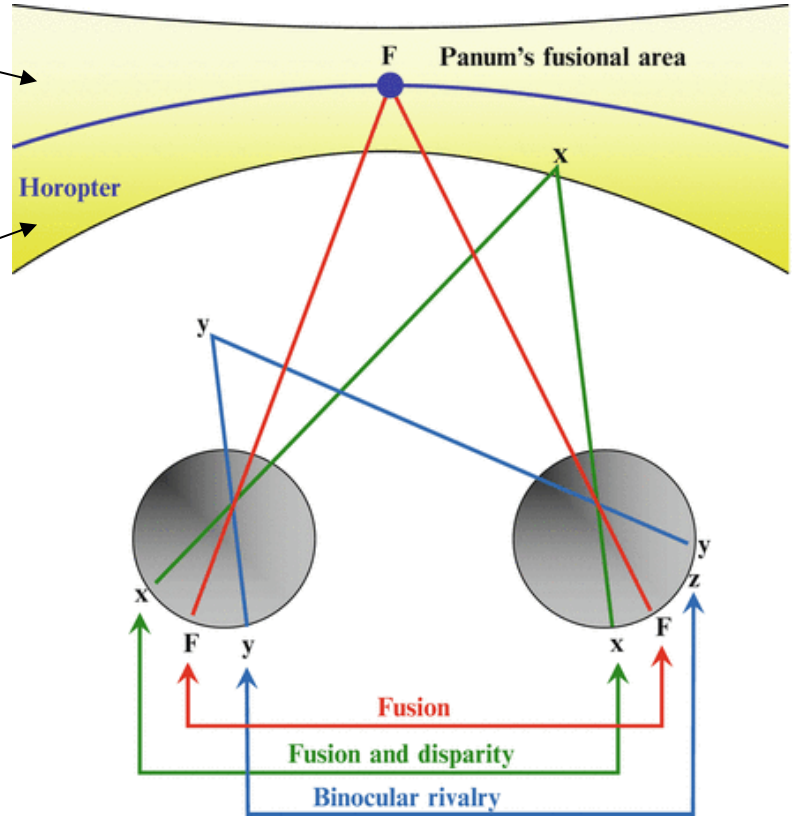
Binocular disparity



The **horopter** is the locus of points in space that have the same disparity as fixation.

Uncrossed disparity

Crossed disparity



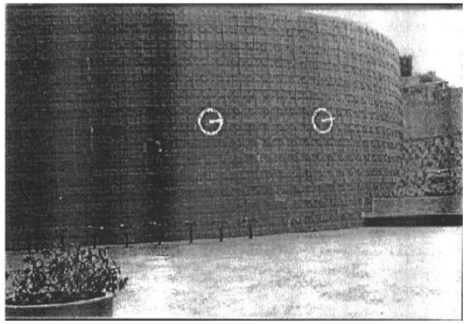
Human stereopsis

- It is hard to recover 3D information from 2D images without extra knowledge, *nevertheless there are monocular cues*

Shading

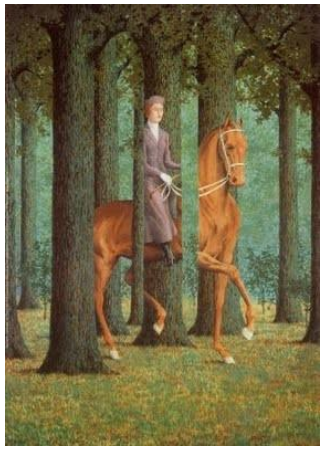
Perspective

Texture



Motion

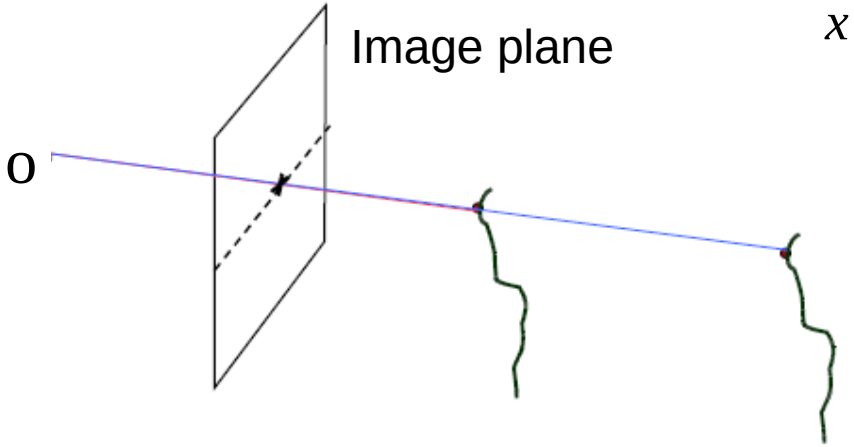
Occlusion

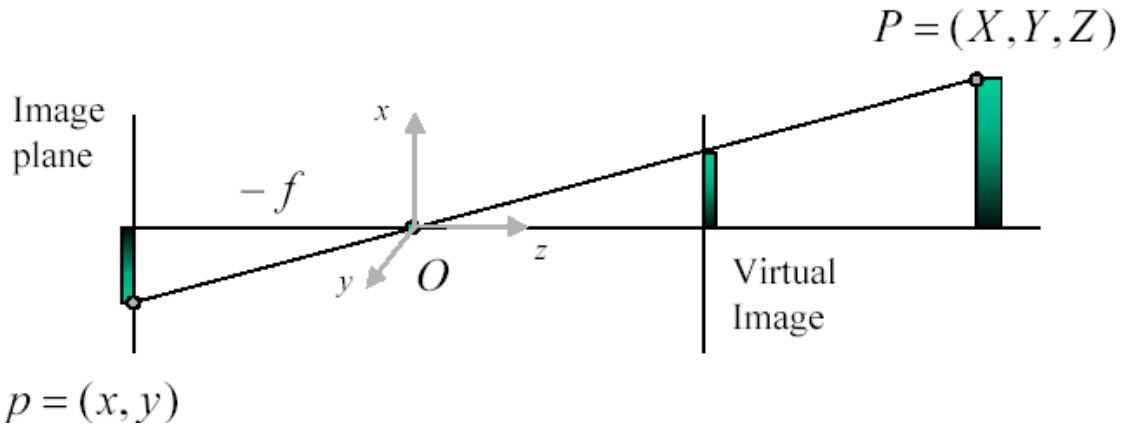


Stereopsis

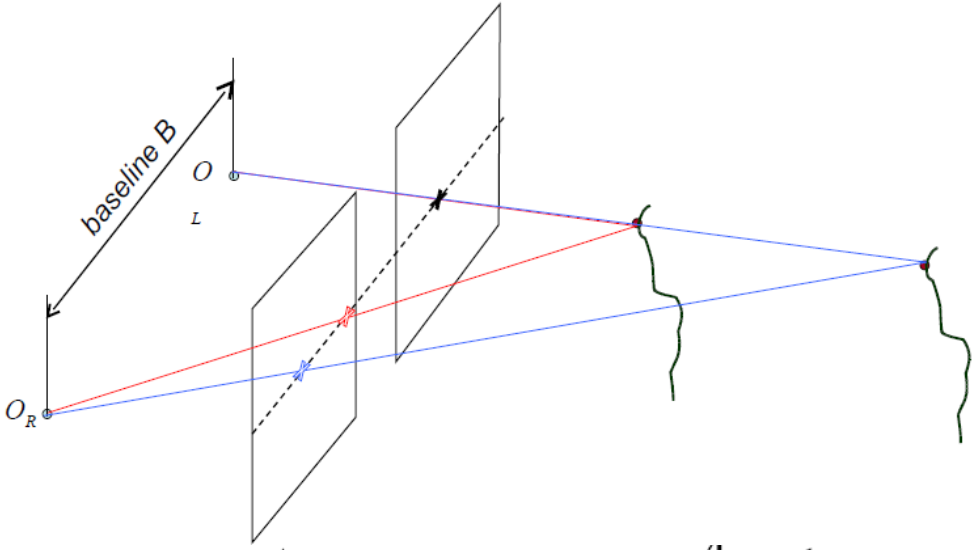
- It is hard to recover 3D information from 2D images (*perspective equation*) *without extra knowledge.*

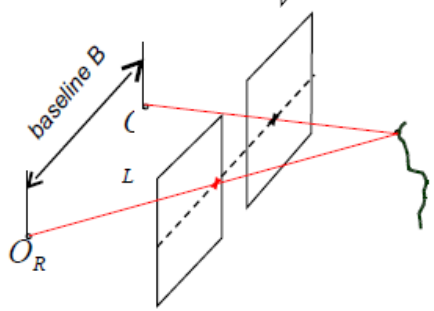
$$x = -f\frac{X}{Z}$$



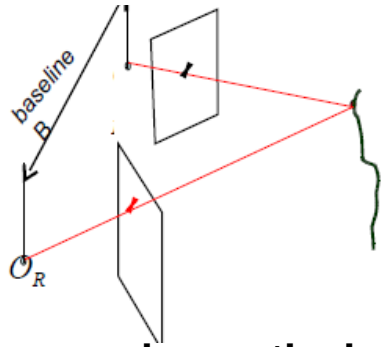


- Much of geometric vision, therefore, is based on information from ***2 camera locations*** (*binocular*).





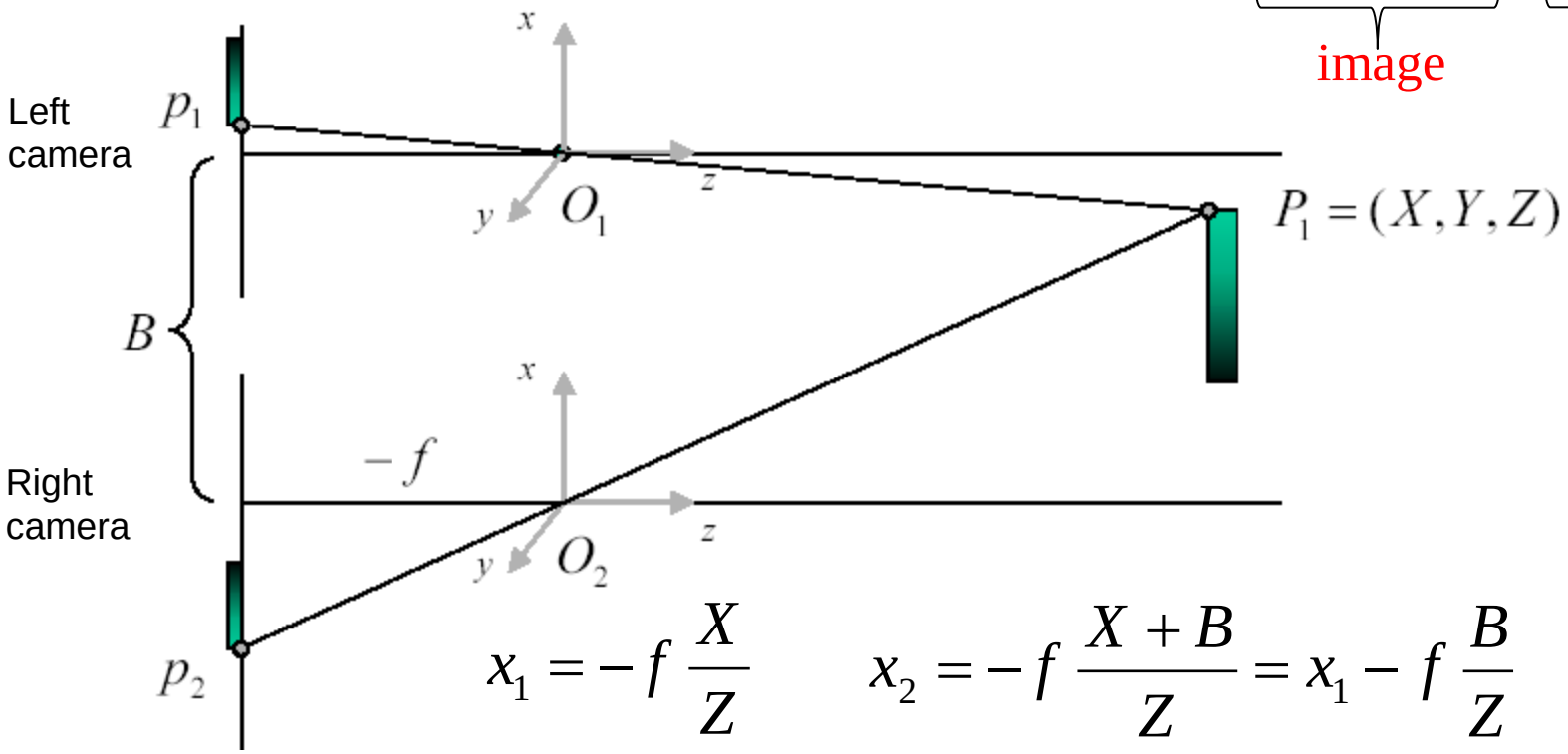
parallel optical axes (*non-verged system*)



converging optical axes (*verged system*)

Stereo for parallel optical axes and calibrated cameras

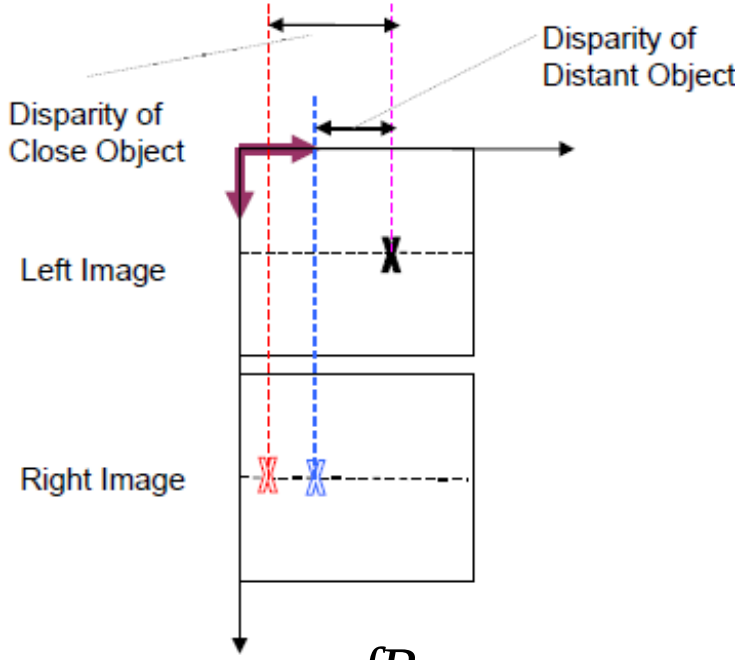
To derive the expression for Z as a function of $\underbrace{x_1 \text{ and } x_2}_{\text{image}}, \underbrace{f \text{ and } B}_{\text{system}}$



$$x_1 = -f\frac{X}{Z} \qquad x_2 = -f\frac{X+B}{Z} = x_1 - f\frac{B}{Z}$$

$$Z = \frac{fB}{x_1 - x_2}$$

We define the **disparity** $d = x_1 - x_2$

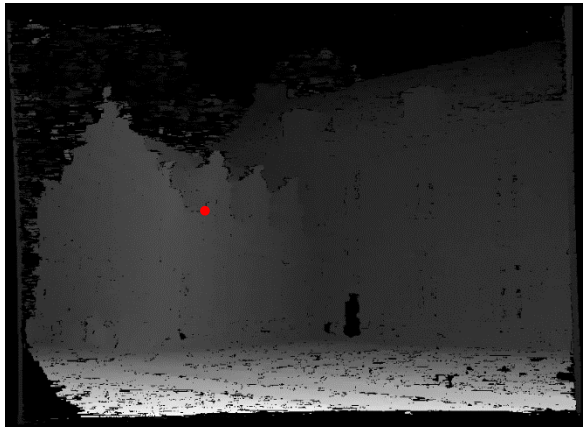


$$d = \frac{fB}{Z}$$

Depth from disparity

Left image I(x,y)

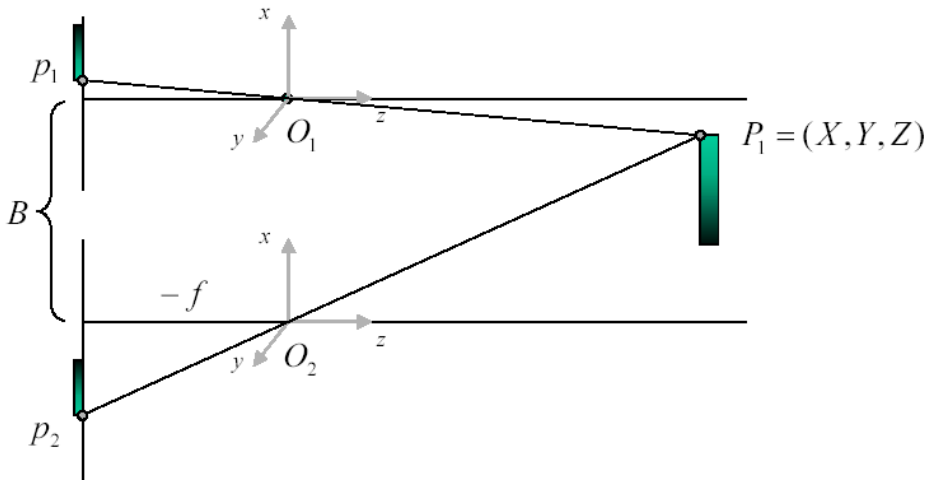
Disparity map D(x,y)



Right image I´(x´,y´)

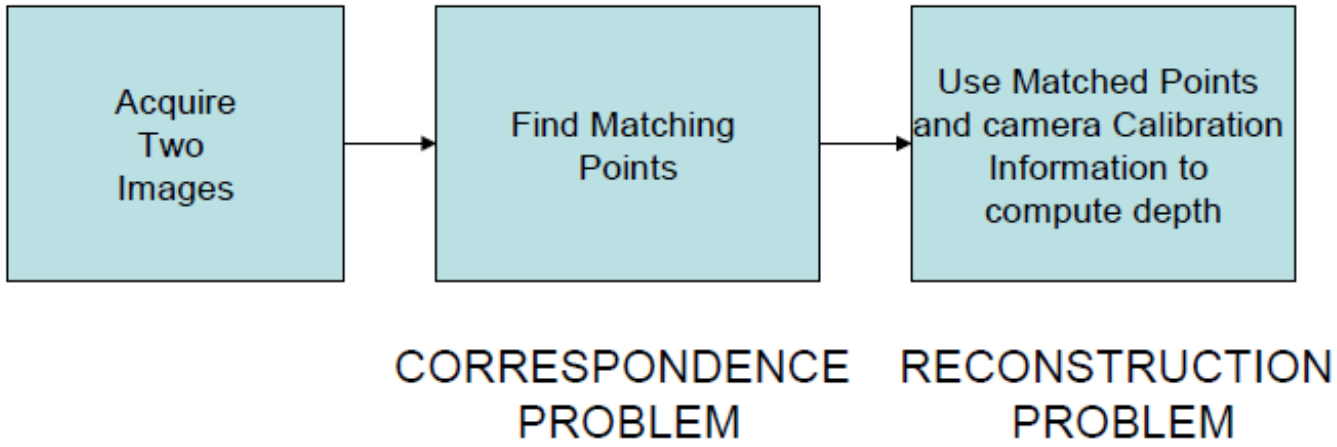
$$(x´,y´)=(x+D(x,y),y)$$

So, if we could find the **corresponding points** in two images, we could **estimate relative depth** by triangulation (*it gives reconstruction as intersection of two rays*)



Two problems of stereo

- Correspondence problem: Given two images, how can you find corresponding points that match?
- Reconstruction problem: Given matching points between images, how can you reconstruct the 3D scene? [here, we need *calibration*]

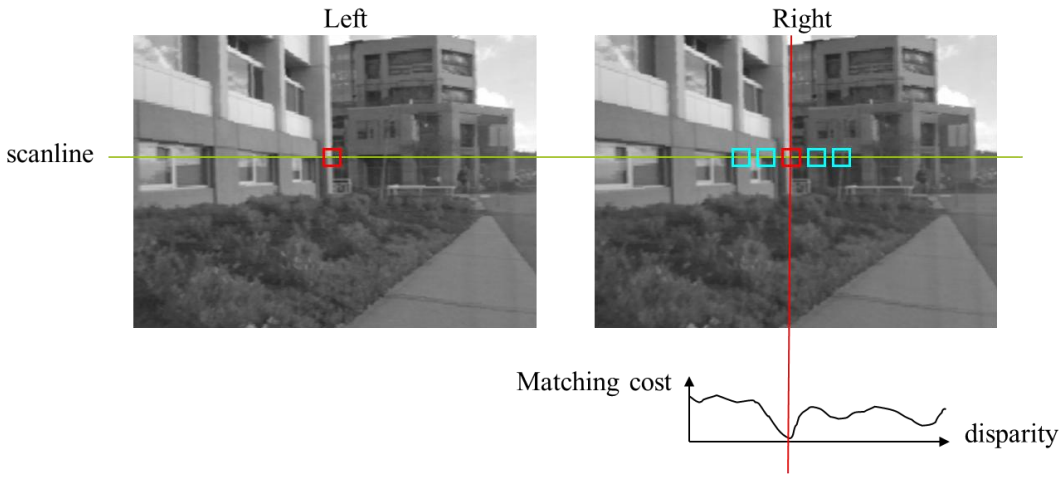


Correspondence problem

- Region-based
  - Region-based (*dense*) matching only works where there is texture
    - compute a confidence measure for regions
    - apply continuity or match ordering constraints
  - Region matching can be sensitive to changes in surface orientation

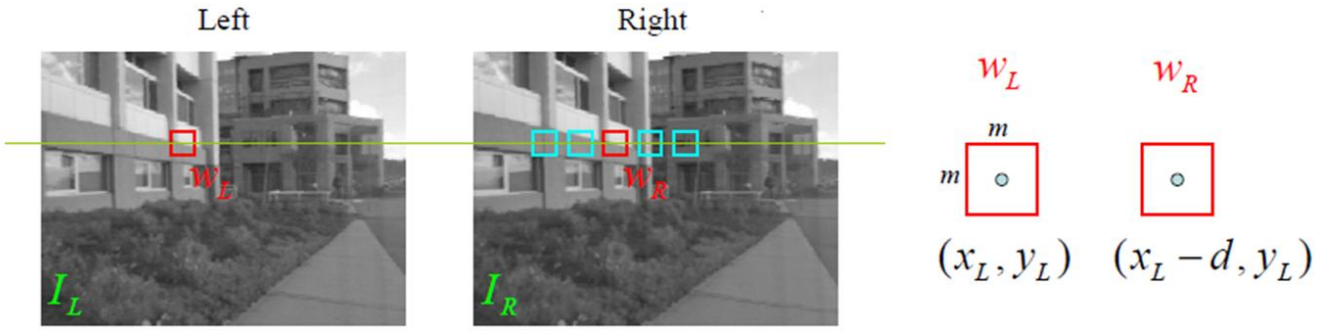
- Feature-based
  - Feature-based leads to sparse disparity maps
    - interpolation to fill in gaps
    - scale-space approaches to fill in gaps
  - Feature-based can be sensitive to feature “drop-outs”

Region-based correspondence



- *Slide a window along the right scanline (**epipolar line**) and compare contents of that **window** with the **reference window** in the left image*

- **Matching** cost: sum of squared differences (SSD) or normalized cross correlation



w_L and w_R are corresponding m by m windows of pixels.

We define the window function :

$$R_m(x,y) = \{u,v \mid x - \tfrac{m}{2} \le u \le x + \tfrac{m}{2}, y - \tfrac{m}{2} \le v \le y + \tfrac{m}{2}\}$$

The SSD cost measures the intensity difference as a function of disparity :

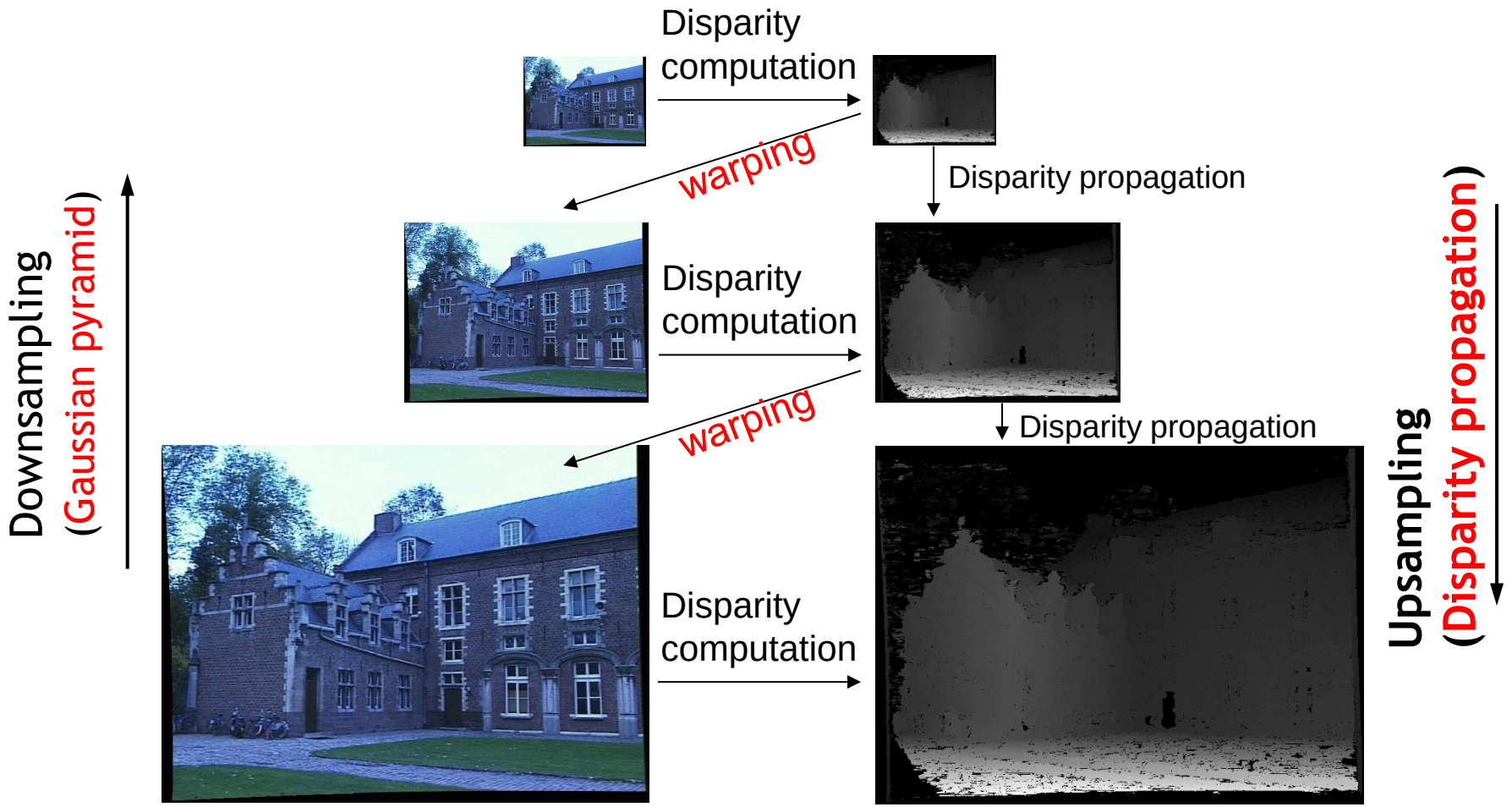
$$C_r(x,y,d) = \sum_{(u,v) \in R_m(x,y)} [I_L(u,v) - I_R(u-d,v)]^2$$

Hierarchical stereo matching (pyramid representation)

Coarse-to-fine approach

Allows **faster** computation

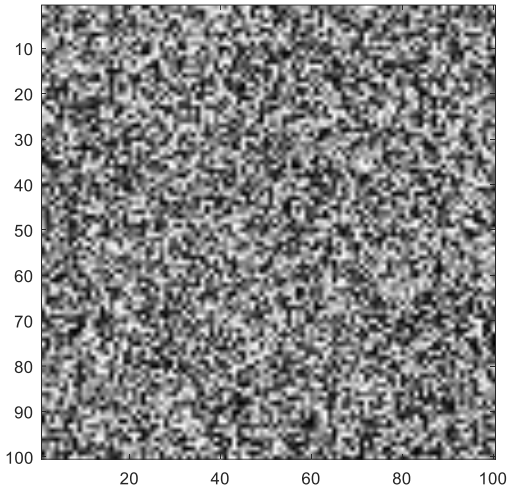
Deals with **large** disparity ranges



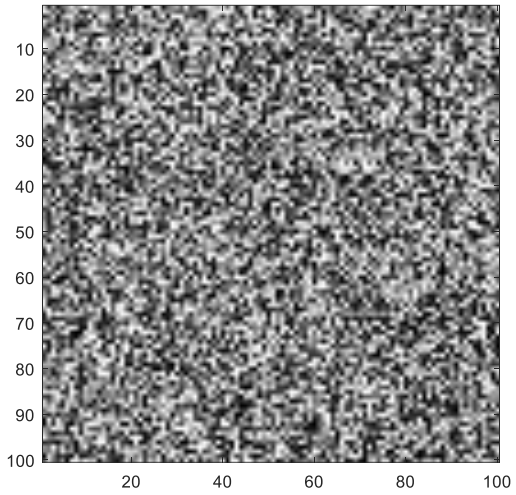
Example of (dense) disparity map computation

See sampleDisp.m

Left RDS

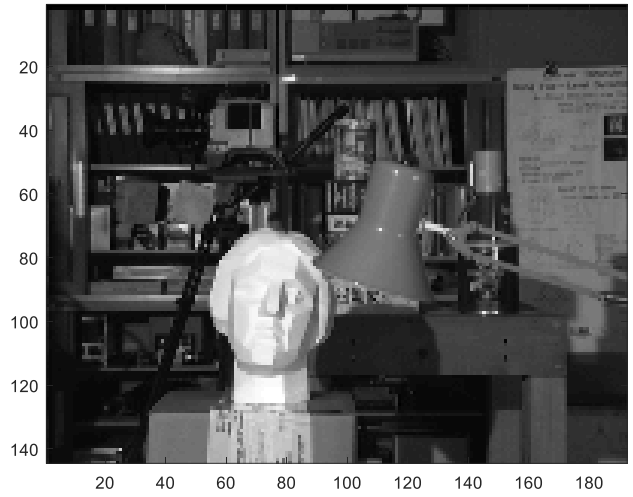


Right RDS

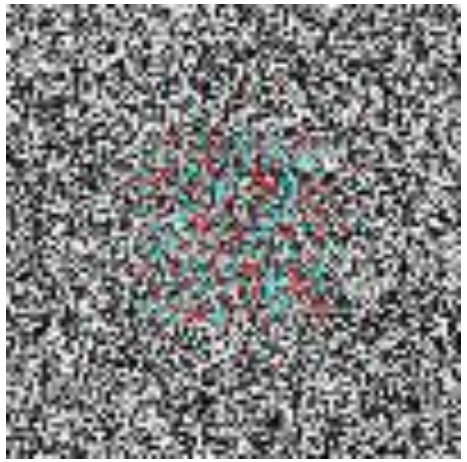


Left image

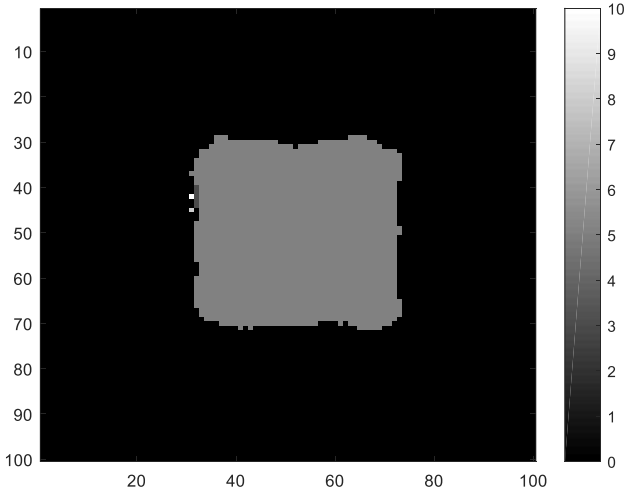
Right image



Anaglyph visualization

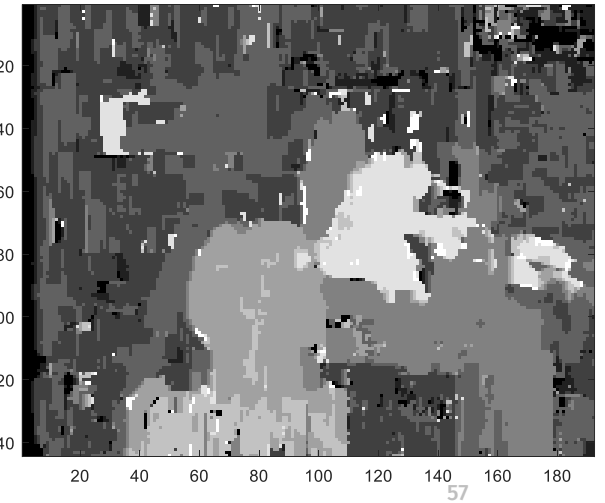


Disparity map (pixels)

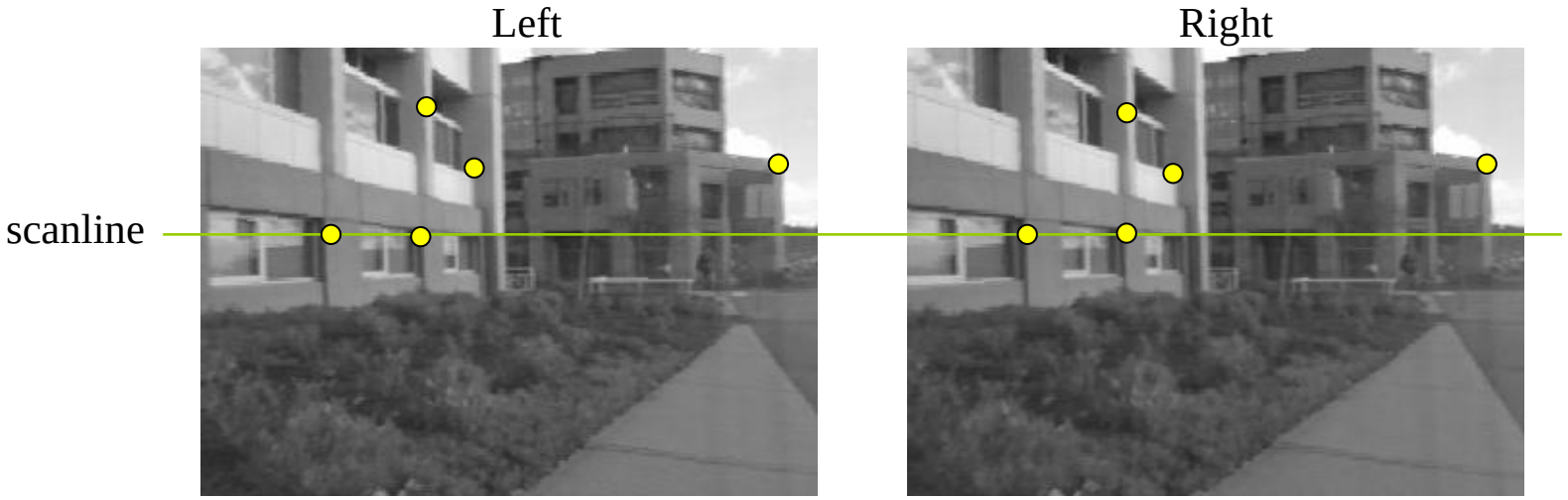


Anaglyph visualization

Disparity map



Feature-based correspondence



- Restrict search to **sparse** set of detected features (***keypoints***)
- Rather than pixel values (or lists of pixel values) use *feature* ***descriptors*** and an associated *feature* ***distance***
- Still narrow search by **epipolar lines**

Stereo camera

https://www.stereolabs.com/zed/

- The stereo cameras have their importance on the market for current applications
- An example is the ZED camera
  - Video: from 2.2K @15 fps to WVGA @ 100fps
  - Depth: native video resolution; depth range 0.3 - 25 m; Depth FOV 90° (H) x 60° (V)
  - Motion: pose update rate up to 100Hz; Pose Drift: Translation: 0.35% Rotation: 0.023°/m; Technology Real-time depth-based visual odometry and SLAM (see next lectures)
  - Sensor Resolution: dual 4M pixels sensors with large 2-micron pixels

```
// Initialize and open the camera
sl.Camera zed = new sl.Camera(0);
InitParameters init_param = new sl.InitParameters();
zed.Open(ref init_param);

// Create matrices to store the depth and point cloud
sl.Mat depth = new sl.Mat();
sl.Mat point_cloud = new sl.Mat();
sl.RuntimeParameters runtime_param = new sl.RuntimeParameters();

while(true){
    // Grab a frame and retrieve depth and point cloud
    zed.Grab(ref runtime_param);
    zed.RetrieveMeasure(depth, sl.MEASURE.DEPTH);
    zed.RetrieveMeasure(point_cloud, sl.MEASURE.XYZRGBA);
}
```