COMPUTER VISION
Features

Emanuel Aldea <emanuel.aldea@u-psud.fr>
http://hebergement.u-psud.fr/emi/

Computer Science and Multimedia Master - University of Pavia

Why do we need invariant features in CV ?

Multiple views require reliable correspondences

Why do we need invariant features in CV ?

Multiple views require reliable correspondences

- how do we *usually* get multiple views ?

Why do we need invariant features in CV ?

Multiple views require reliable correspondences

- how do we *usually* get multiple views ?
  - we use multiple cameras simultaneously

Why do we need invariant features in CV ?

Multiple views require reliable correspondences

- how do we *usually* get multiple views ?
  - we use multiple cameras simultaneously
  - one camera is moving while acquiring data - and the scene is static

Why do we need invariant features in CV ?

Multiple views require reliable correspondences

- how do we *usually* get multiple views ?
  - we use multiple cameras simultaneously
  - one camera is moving while acquiring data - and the scene is static

A fundamental step for :

- estimating how cameras are located relatively to each other

Why do we need invariant features in CV ?

Multiple views require reliable correspondences

- how do we *usually* get multiple views ?
  - we use multiple cameras simultaneously
  - one camera is moving while acquiring data - and the scene is static

A fundamental step for :

- estimating how cameras are located relatively to each other
- recovering scene depth

Why do we need invariant features in CV ?

Multiple views require reliable correspondences

- how do we *usually* get multiple views ?
  - we use multiple cameras simultaneously
  - one camera is moving while acquiring data - and the scene is static

A fundamental step for :

- estimating how cameras are located relatively to each other
- recovering scene depth
- estimating ego-movement (visual odometry)

Why do we need invariant features in CV ?

Multiple views require reliable correspondences

- how do we *usually* get multiple views ?
  - we use multiple cameras simultaneously
  - one camera is moving while acquiring data - and the scene is static

A fundamental step for :

- estimating how cameras are located relatively to each other
- recovering scene depth
- estimating ego-movement (visual odometry)
- matching image content in general

Why do we need invariant features in CV ?

Multiple views require reliable correspondences

- how do we *usually* get multiple views ?
  - we use multiple cameras simultaneously
  - one camera is moving while acquiring data - and the scene is static

A fundamental step for :

- estimating how cameras are located relatively to each other
- recovering scene depth
- estimating ego-movement (visual odometry)
- matching image content in general

The foundations of Computer Vision are based on these tasks, and features play thus a significant role in this field.

Why do we need invariant features in CV ?

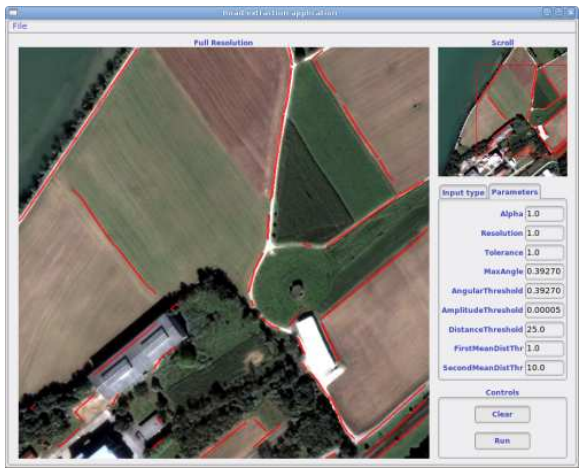
Why not use contours ?

- the processing effort is relatively low
- parametric curves may be extracted relatively easy as well (Hough)

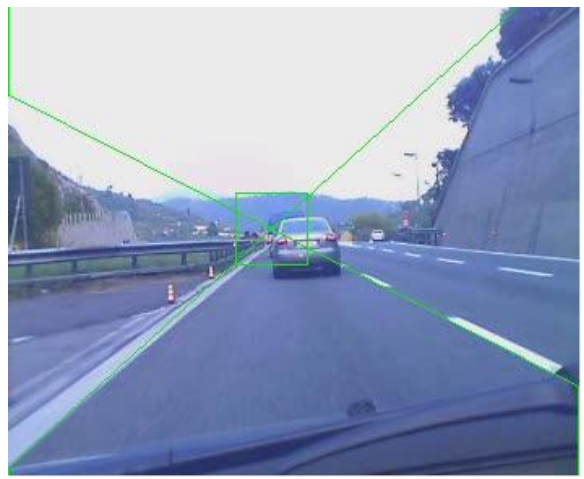
Why do we need invariant features in CV ?

Why not use contours ?

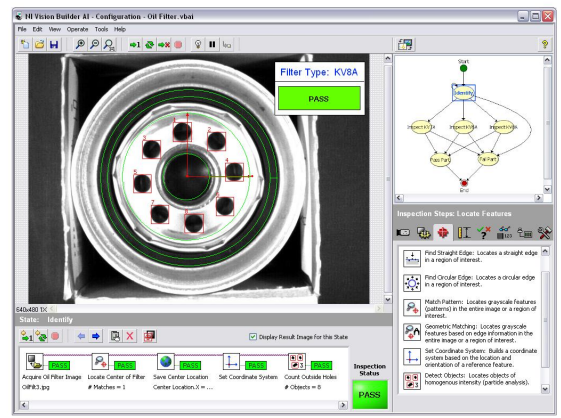
- the processing effort is relatively low
- parametric curves may be extracted relatively easy as well (Hough)
- various applications for specific environments :
  - road / panel / text detection
  - medical and satellite imagery
  - inspection for industrial vision



Aerial imagery



Lane detection



Industrial vision

Why do we need invariant features in CV ?

Why not use contours ?

- the processing effort is relatively low
- parametric curves may be extracted relatively easy as well (Hough)
- various applications for specific environments :
  - road / panel / text detection
  - medical and satellite imagery
  - inspection for industrial vision

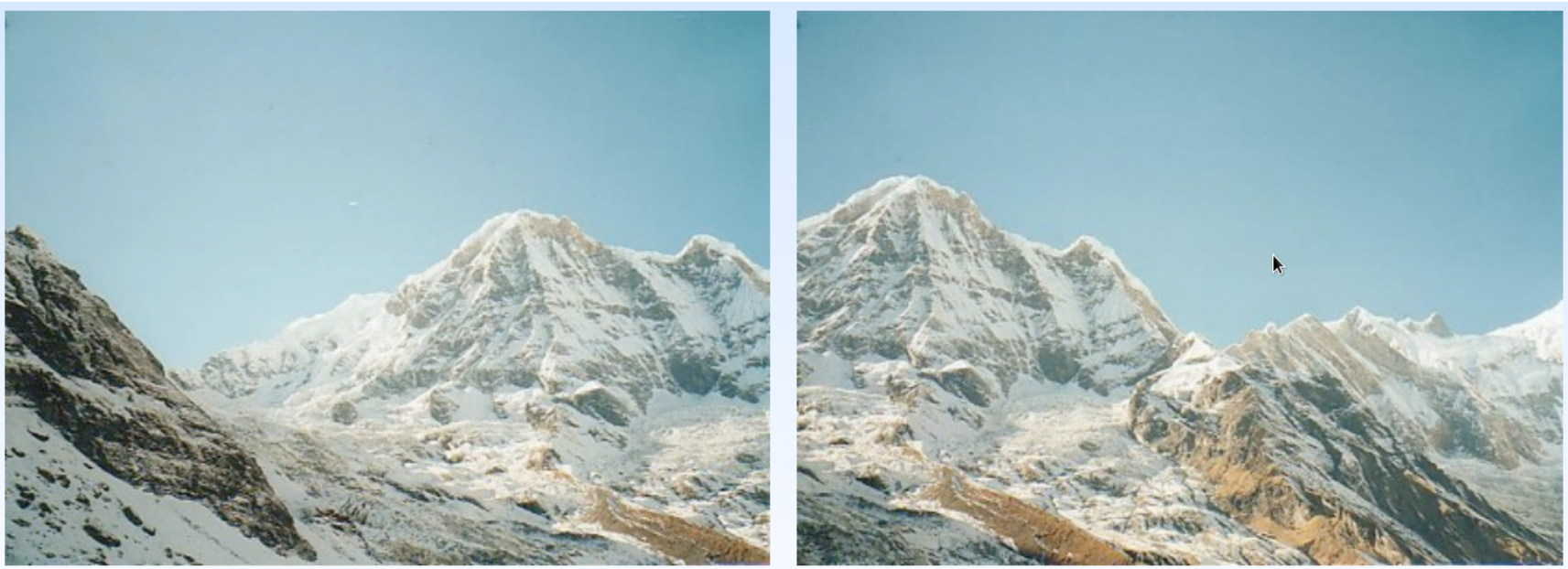
✓ Fast, specialized tasks

✓ Intensity variation invariant

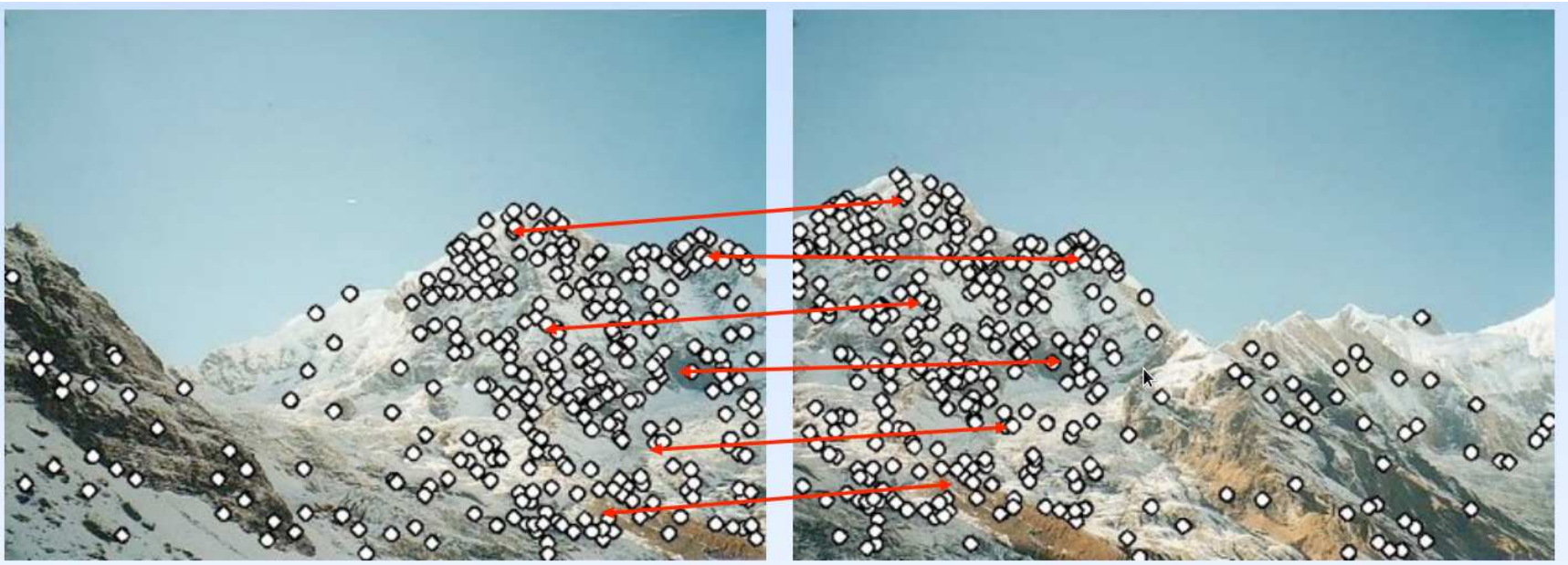
✗ Sensitive to other geometric transforms

✗ Problem for pattern recognition

Simple motivator - panoramic images



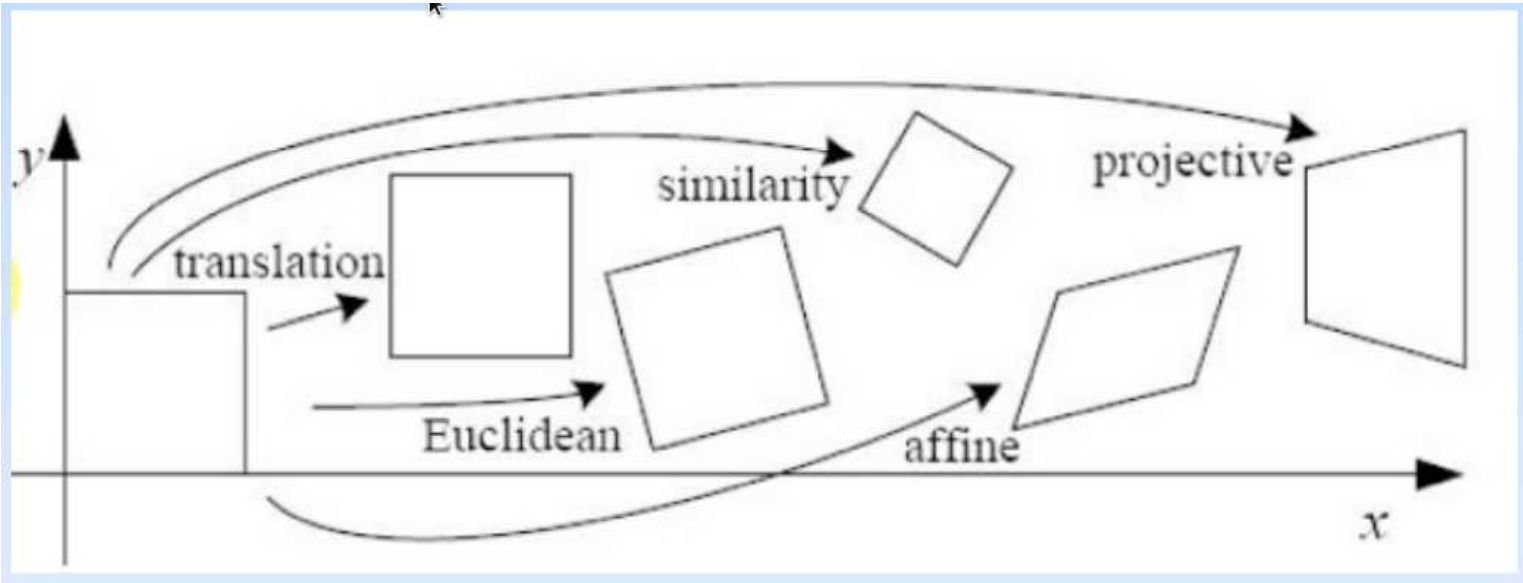
Simple motivator - panoramic images



Simple motivator - panoramic images



The core of the problem



- translation
- Euclidean (translation + rotation)
- similarity transform (tr. + rot. + scale)
- affine (rot. + scale + shear + translation)
- projective

Why we need invariance in CV

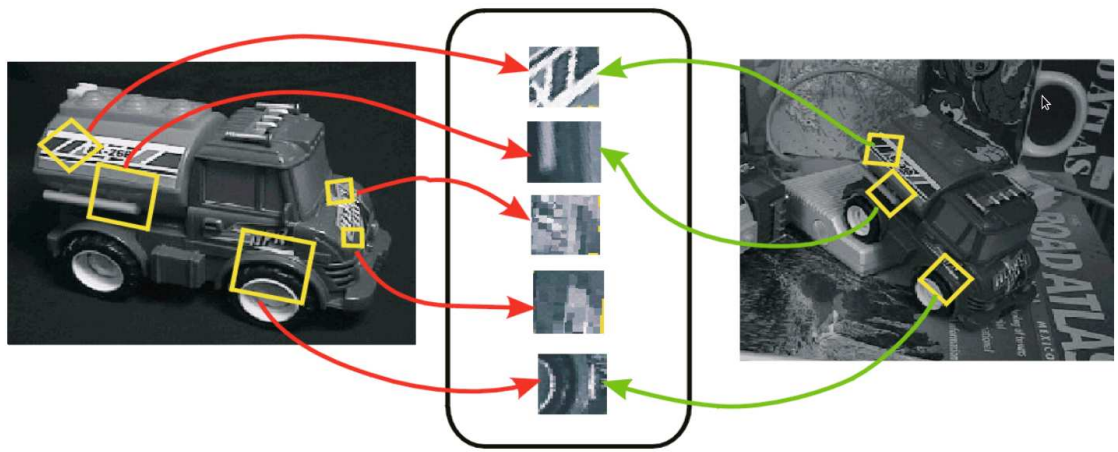
Objective

- identify structures which are invariant with respect to rotation, rescaling, etc.
- these structures are commonly called interest points or corners

Why we need invariance in CV

Objective

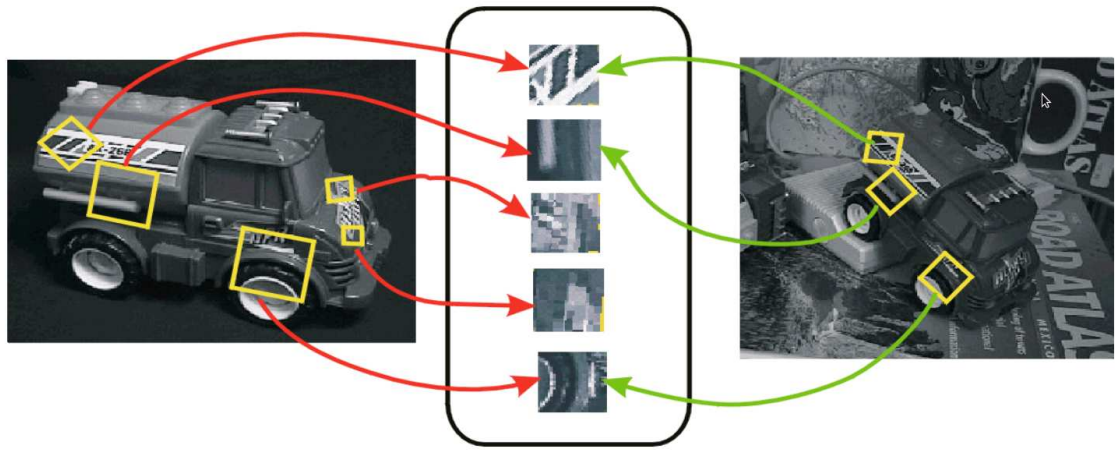
- identify structures which are invariant with respect to rotation, rescaling, etc.
- these structures are commonly called interest points or corners



Why we need invariance in CV

Objective

- identify structures which are invariant with respect to rotation, rescaling, etc.
- these structures are commonly called interest points or corners



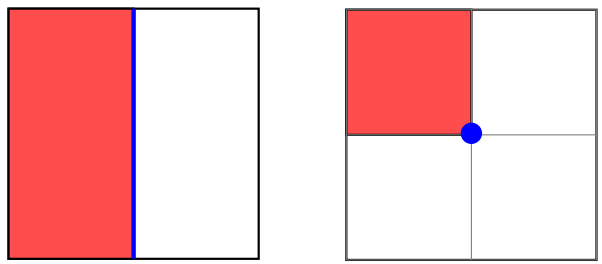
How to :

- identify them in a non supervised manner ?
- associate them robustly ?

Corner detectors : the basics

Definition

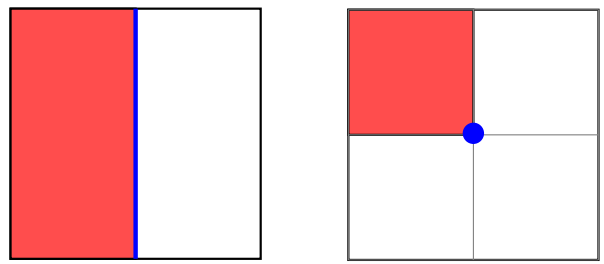
Corner : a location in the image which is characterized by strong intensity variation along two different directions.



Corner detectors : the basics

Definition

Corner : a location in the image which is characterized by strong intensity variation along two different directions.

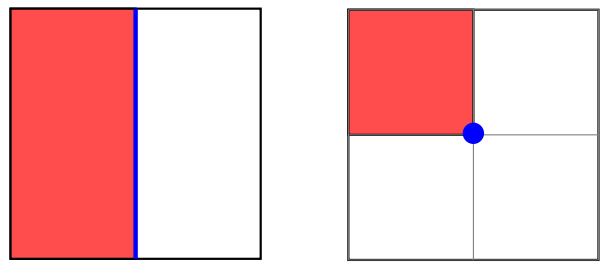


We will still need to compute the local image gradients

Corner detectors : the basics

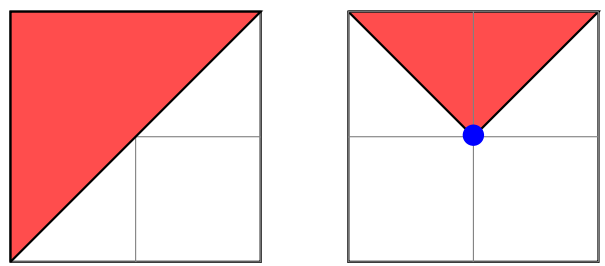
Definition

Corner : a location in the image which is characterized by strong intensity variation along two different directions.



We will still need to compute the local image gradients

- but it is not enough (to do it only in the image reference system) !



Corner detectors : the basics

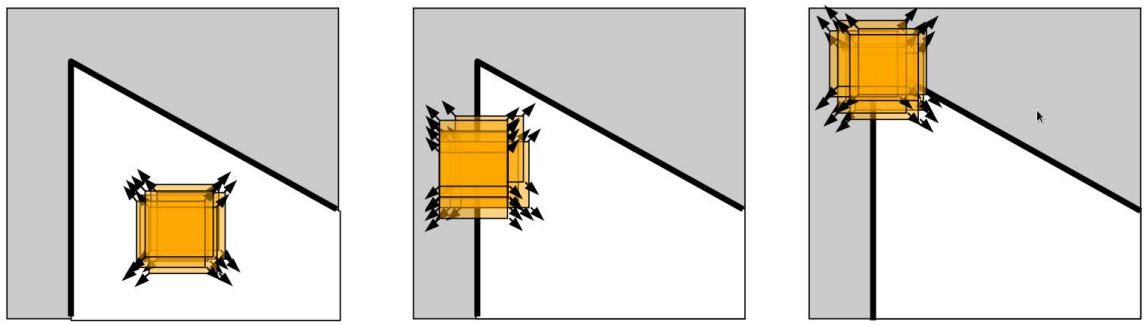
Definition

Strategy : the content of a patch centered in the corner should vary across all possible directions

Corner detectors : the basics

Definition

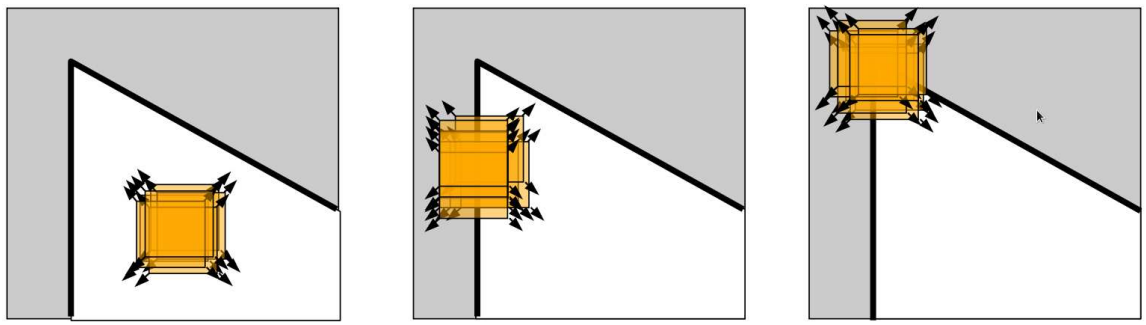
Strategy : the content of a patch centered in the corner should vary across all possible directions



Corner detectors : the basics

Definition

Strategy : the content of a patch centered in the corner should vary across all possible directions



Typical behavior :

- homogeneous regions : no change in patch content
- contours : no change along the contour
- corners : important change across all directions
- corner quality : defined by the smallest possible change
- proposed by Moravec in 1980

Corner detectors : the basics

Intensity change by shift of $(\Delta x, \Delta y)$

$$E(x, y, \Delta x, \Delta y) = \sum_x \sum_y w(x, y) \big[I(x, y) - I(x + \Delta x, y + \Delta y) \big]^2$$

Corner detectors : the basics

Intensity change by shift of $(\Delta x, \Delta y)$

$$E(x, y, \Delta x, \Delta y) = \sum_x \sum_y w(x, y) \big[\underbrace{I(x, y)}_{\text{intensity}} - I(x + \Delta x, y + \Delta y) \big]^2$$

Corner detectors : the basics

Intensity change by shift of $(\Delta x, \Delta y)$

$$E(x, y, \Delta x, \Delta y) = \sum_x \sum_y w(x, y) \big[\underbrace{I(x, y)}_{\text{intensity}} - \underbrace{I(x + \Delta x, y + \Delta y)}_{\text{shifted intensity}} \big]^2$$

Corner detectors : the basics

Intensity change by shift of $(\Delta x, \Delta y)$

$$E(x, y, \Delta x, \Delta y) = \sum_x \sum_y \underbrace{w(x, y)}_{\text{support}} \Big[\underbrace{I(x, y)}_{\text{intensity}} - \underbrace{I(x + \Delta x, y + \Delta y)}_{\text{shifted intensity}} \Big]^2$$

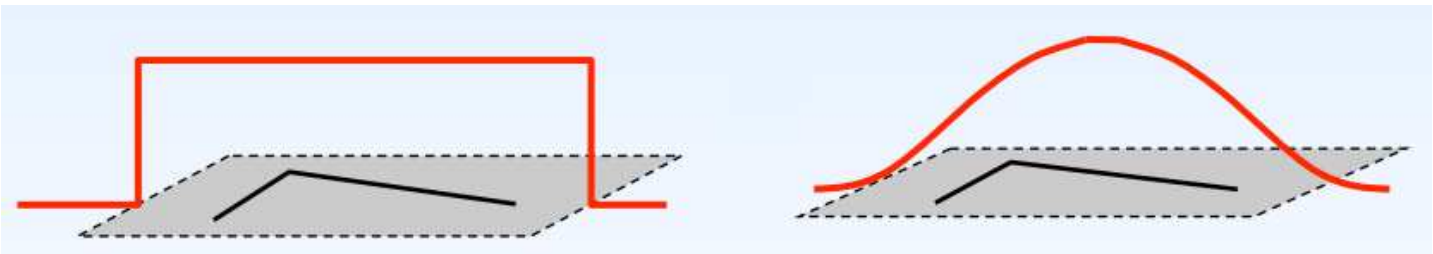


FIGURE – Possible choices for the support function $w(x, y)$

Corner detectors : the basics

Intensity change by shift of $(\Delta x, \Delta y)$

$$E(x, y, \Delta x, \Delta y) = \sum_x \sum_y \underbrace{w(x, y)}_{\text{support}} \Big[\underbrace{I(x, y)}_{\text{intensity}} - \underbrace{I(x + \Delta x, y + \Delta y)}_{\text{shifted intensity}} \Big]^2$$

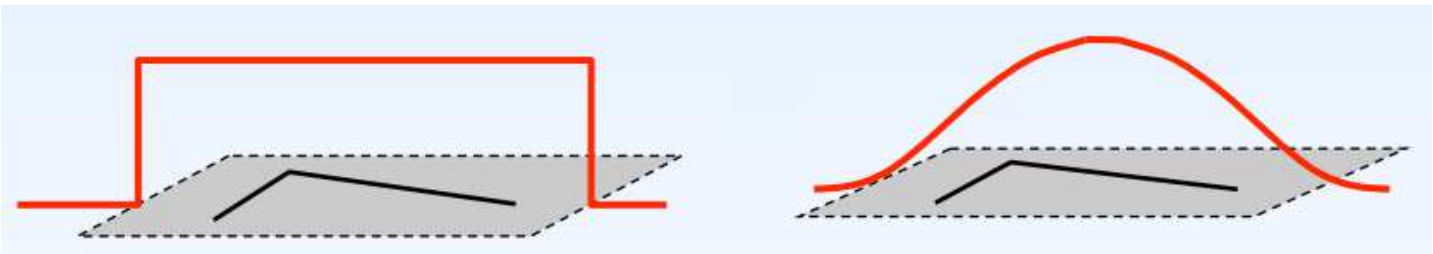


FIGURE – Possible choices for the support function $w(x, y)$

$E(x, y)$ large highlights a potential corner.

Corner detectors : the basics

Intensity change by shift of $(\Delta x, \Delta y)$

$$E(x, y, \Delta x, \Delta y) = \sum_x \sum_y \underbrace{w(x, y)}_{\text{support}} \Big[\underbrace{I(x, y)}_{\text{intensity}} - \underbrace{I(x + \Delta x, y + \Delta y)}_{\text{shifted intensity}} \Big]^2$$

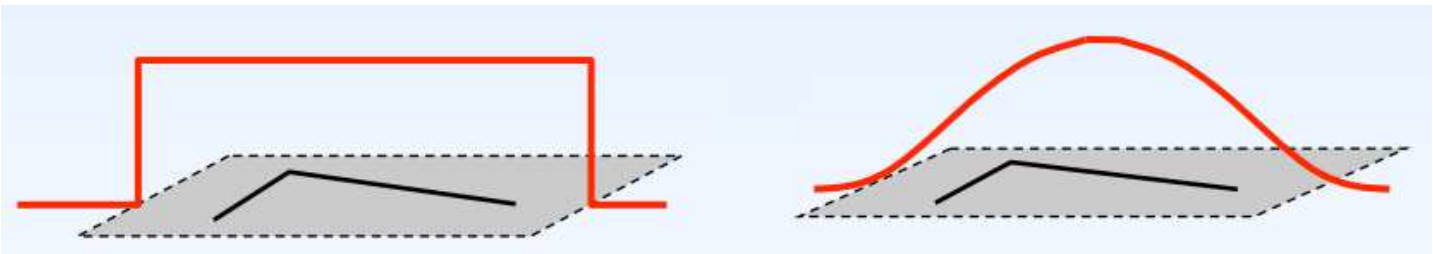


FIGURE – Possible choices for the support function $w(x, y)$

Costly if we do not use any tricks

- what is approximately the computational cost for an image of side N if we implement this method naively using a patch of side K ?

Corner detectors : the basics

First order approximation by Taylor series development

$$f(x + \Delta x, y + \Delta y) = f(x, y) + f_x(x, y)\Delta x + f_y(x, y)\Delta y$$

Corner detectors : the basics

First order approximation by Taylor series development

$$f(x+\Delta x, y+\Delta y) = f(x,y) + f_x(x,y)\Delta x + f_y(x,y)\Delta y$$

We use this approximation to rewrite the intensity variation due to shift :

$$\begin{aligned}
\sum \left[I(x+\Delta x, y+\Delta y) - I(x,y)\right]^2 &\approx \sum \left[I(x,y) + \Delta x I_x(x,y) + \Delta y I_y(x,y) - I(x,y)\right]^2 \\
&\approx \sum \Delta x^2 I_x^2 + 2\Delta x \Delta y I_x I_y + \Delta y^2 I_y^2 \\
&\approx \sum [\Delta x \Delta y] \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \\
&\approx [\Delta x \Delta y] \left(\sum \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \right) \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}
\end{aligned}$$

Corner detectors : the basics

First order approximation by Taylor series development

$$f(x+\Delta x, y+\Delta y) = f(x,y) + f_x(x,y)\Delta x + f_y(x,y)\Delta y$$

We use this approximation to rewrite the intensity variation due to shift :

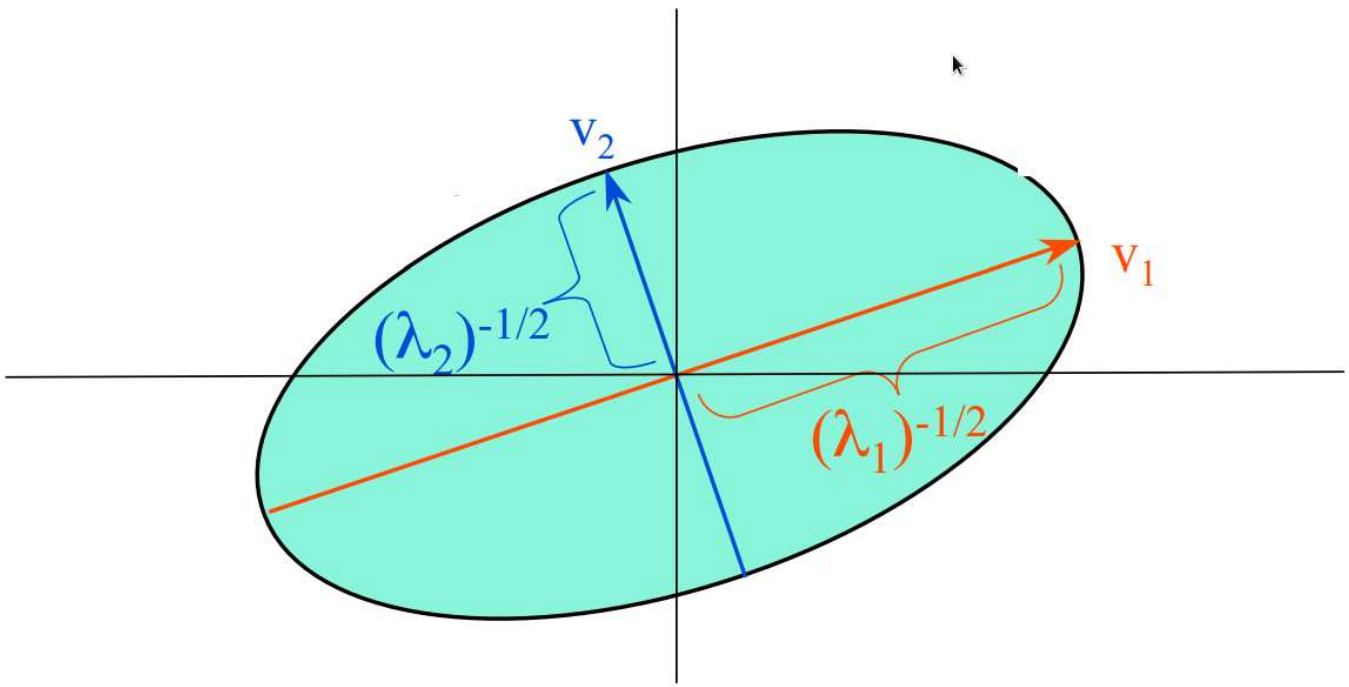
$$\begin{aligned}
\sum \left[I(x+\Delta x, y+\Delta y) - I(x,y)\right]^2 &\approx \sum \left[I(x,y) + \Delta x I_x(x,y) + \Delta y I_y(x,y) - I(x,y)\right]^2 \\
&\approx \sum \Delta x^2 I_x^2 + 2\Delta x \Delta y I_x I_y + \Delta y^2 I_y^2 \\
&\approx \sum [\Delta x \Delta y] \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \\
&\approx [\Delta x \Delta y] \left(\sum \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \right) \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}
\end{aligned}$$

$$\begin{aligned}
E(x,y,\Delta x,\Delta y) &\approx [\Delta x \Delta y] \left(\sum g(\sigma_I) \star \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \right) \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \\
&\approx [\Delta x \Delta y] \underbrace{\begin{bmatrix} \langle I_x^2 \rangle & \langle I_x I_y \rangle \\ \langle I_x I_y \rangle & \langle I_y^2 \rangle \end{bmatrix}}_{\text{structure tensor}} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}
\end{aligned}$$

Corner detectors : the structure tensor

Properties

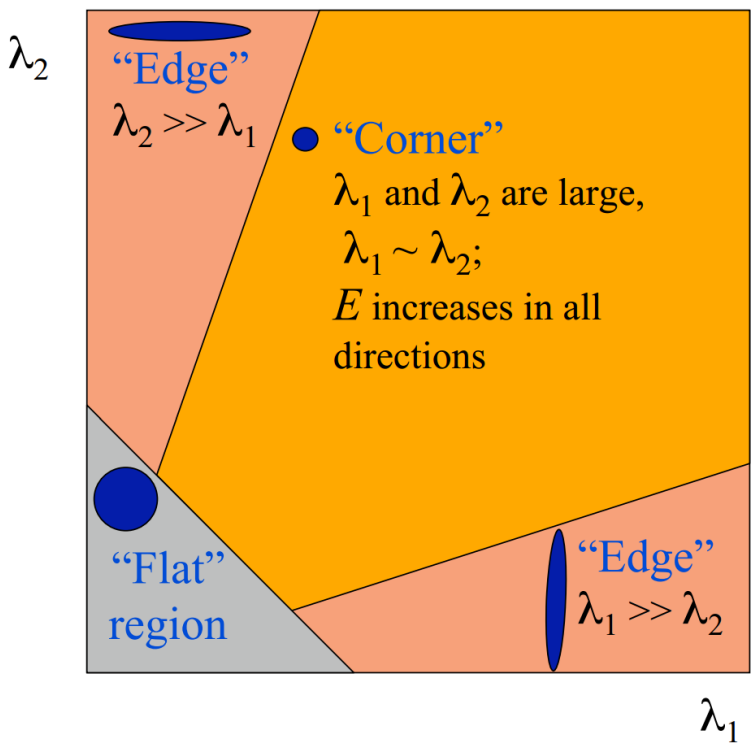
- the eigenvectors highlight the main directions of gradient variation around the location we consider (see the ellipse of constant change)
- ex. : if $\lambda_2 > \lambda_1$, strong variation along v_2 and smaller variation in the direction of v_1
- if corner, λ_1, λ_2 are large



Corner detectors : the structure tensor

Properties

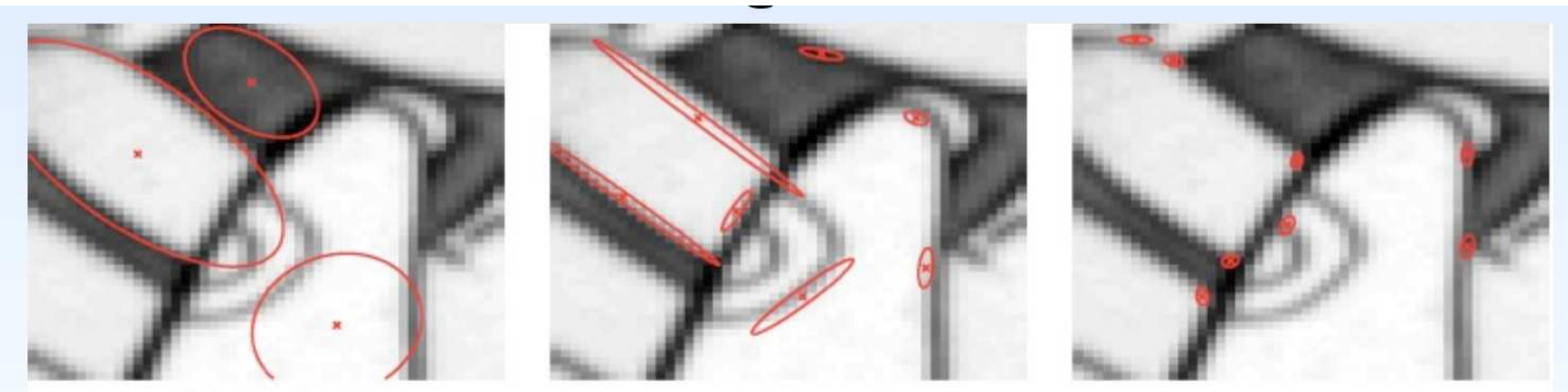
- the eigenvectors highlight the main directions of gradient variation around the location we consider (see the ellipse of constant change)
- ex. : if $\lambda_2 > \lambda_1$, strong variation along v_2 and smaller variation in the direction of v_1
- if corner, λ_1, λ_2 are large



Corner detectors : the structure tensor

Properties

- the eigenvectors highlight the main directions of gradient variation around the location we consider (see the ellipse of constant change)
- ex. : if $\lambda_2 > \lambda_1$, strong variation along v_2 and smaller variation in the direction of v_1
- if corner, λ_1, λ_2 are large



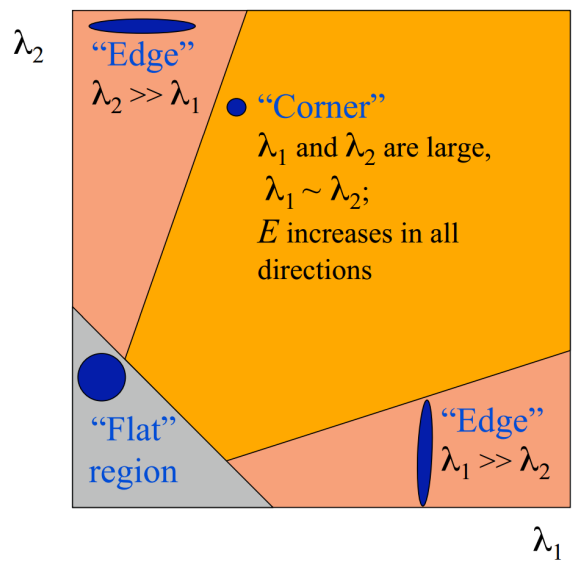
Corner detectors : the structure tensor

Decision based on the tensor eigenvalues

- one may compute λ_1, λ_2 explicitly, but too costly
- prefered method :

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

- the value of parameter α is usually 0.04 - 0.06
- interesting eigenvalues $=$ local maxima of R



Corner detectors : Harris detector

Main algorithm steps

1. compute gradients $I_x = \frac{\partial}{\partial x} g(\sigma_D) \star I$, $I_y = \frac{\partial}{\partial y} g(\sigma_D) \star I$
2. compute the structure tensor :

$$M = g(\sigma_I) \star \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}$$

3. compute the response function R :

$$R = det(M) - \alpha trace^2(M)$$

4. apply thresholding to R
5. non maximal suppression on the values of R

Corner detectors : example



FIGURE – Initial pair

Corner detectors : example

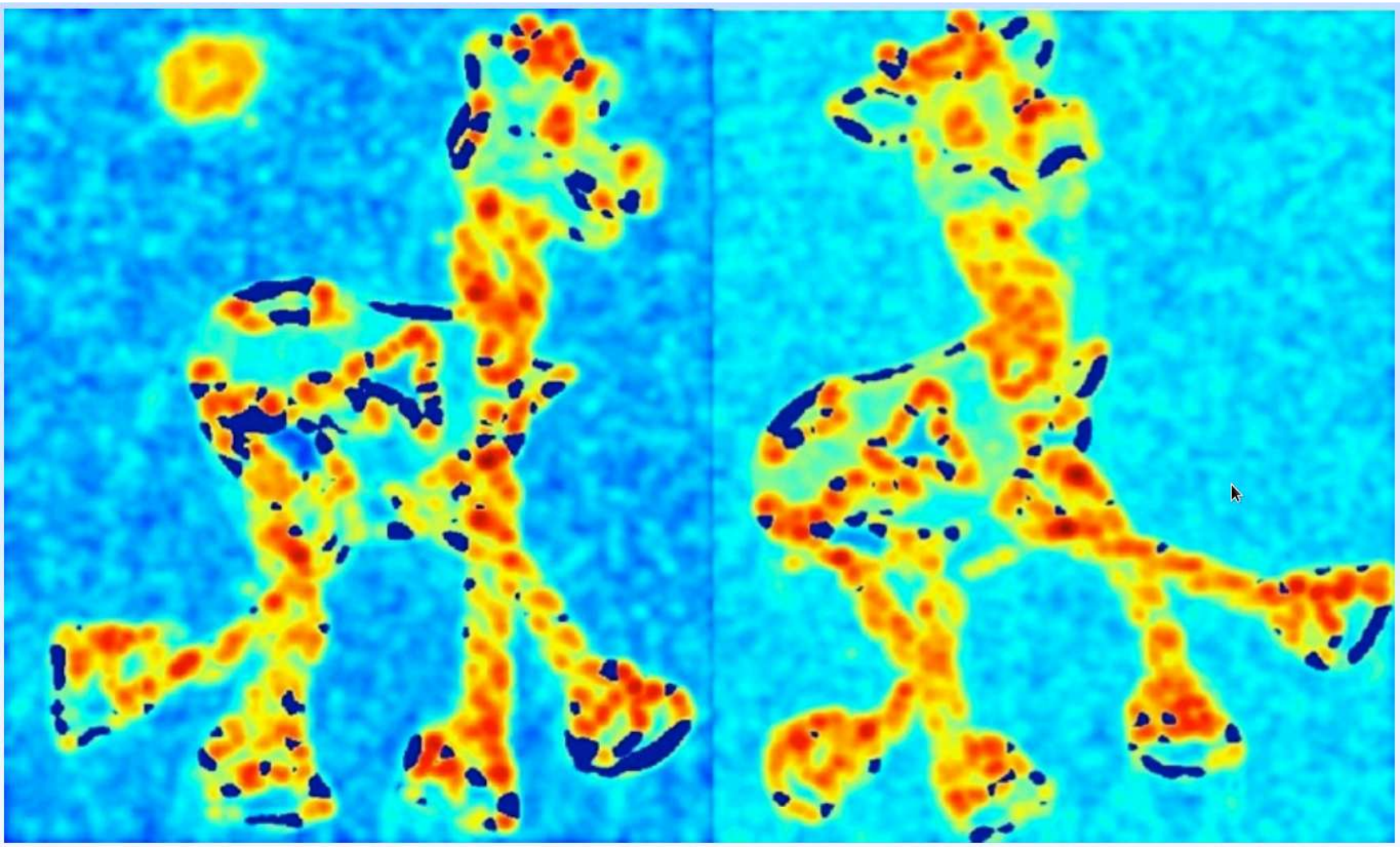


FIGURE – response function R

Corner detectors : example

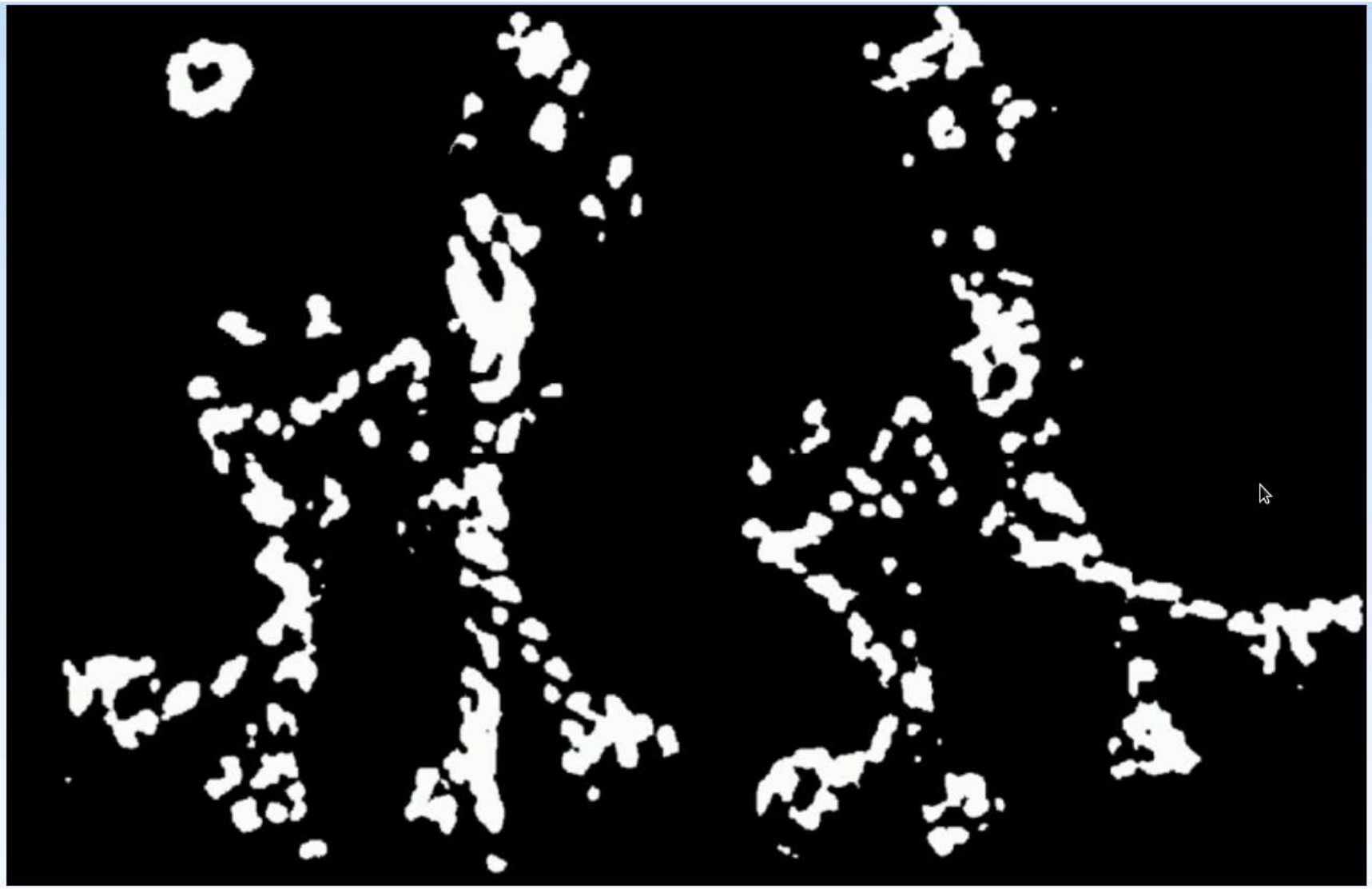


FIGURE – Thresholding R

Corner detectors : example

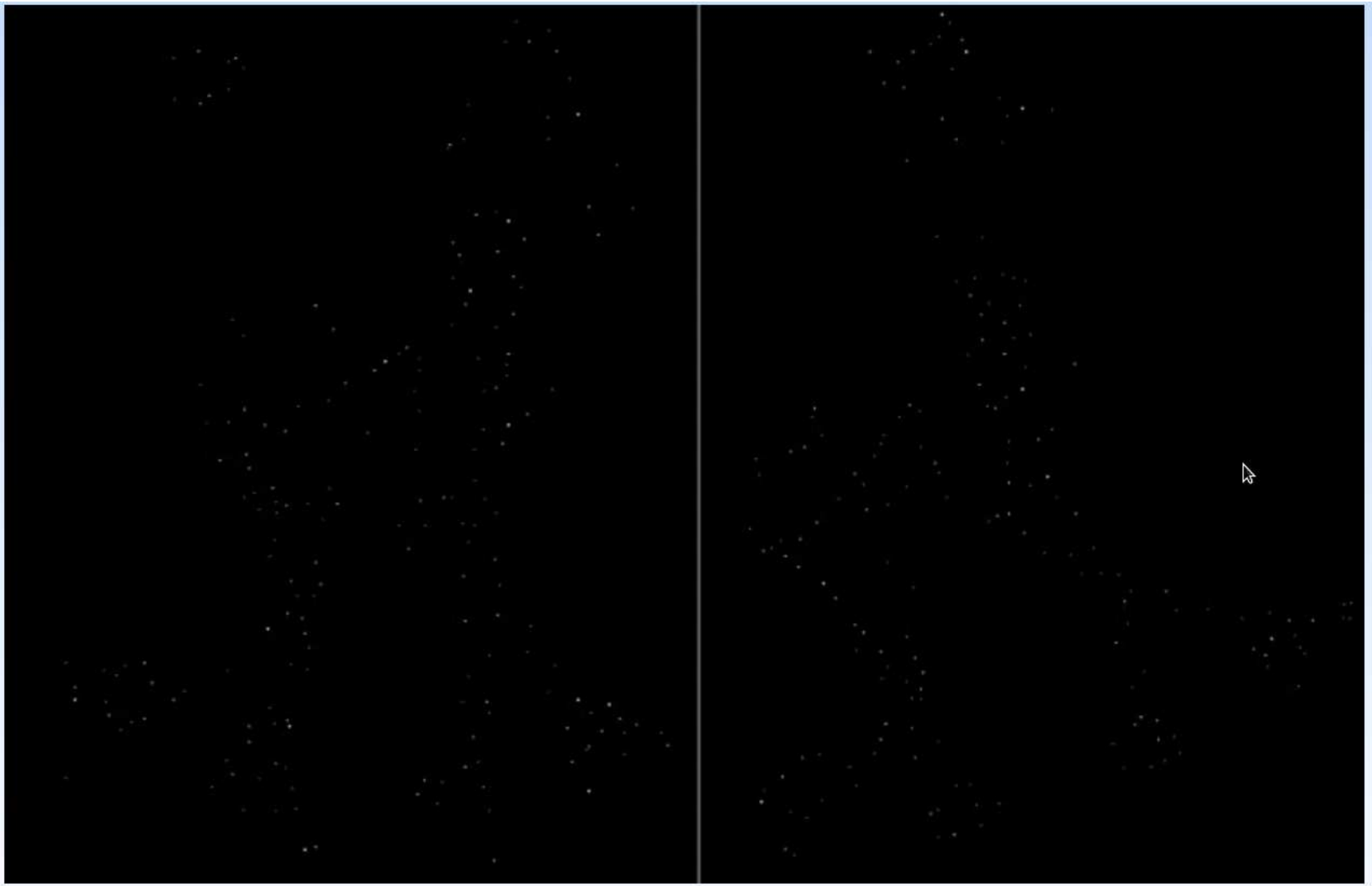


FIGURE – Non maximal suppression on R

Corner detectors : example

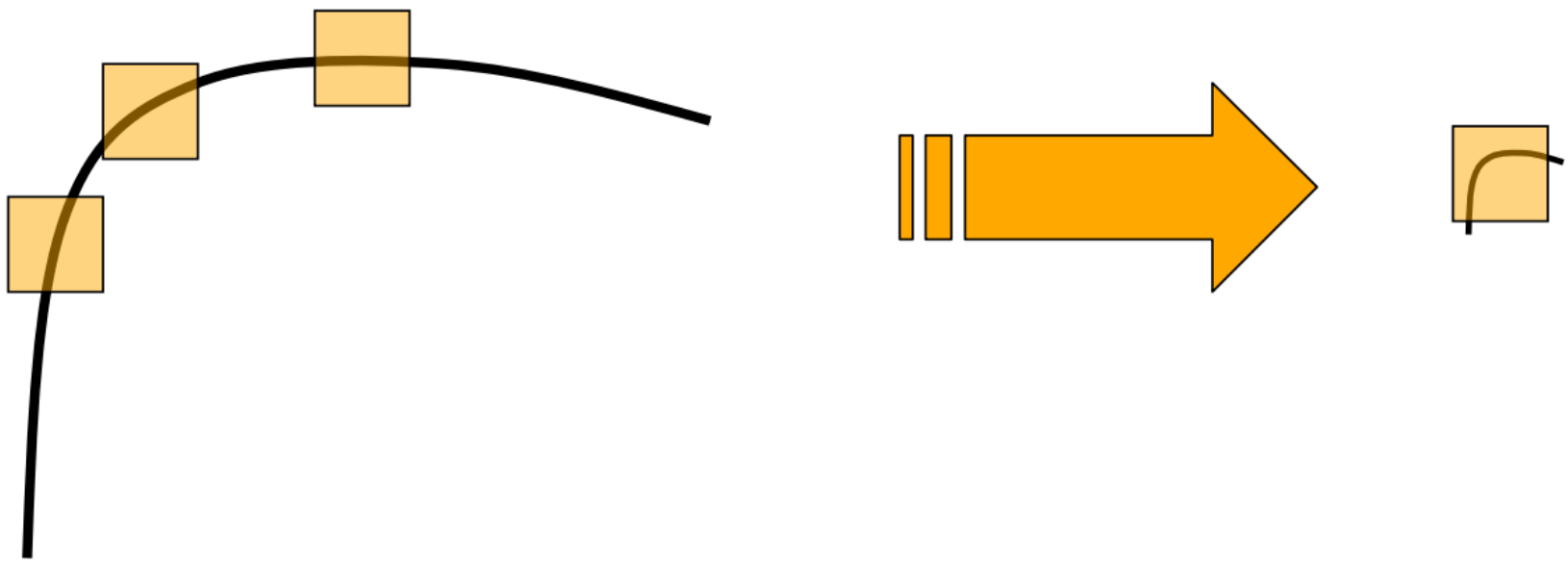


FIGURE – Detector results

Conclusion : Harris detector

Conclusions

- ✓ rotation invariant detector
- ✓ intensity change invariant
- ✗ not robust to scale change
- ✗ no descriptor provided for matching



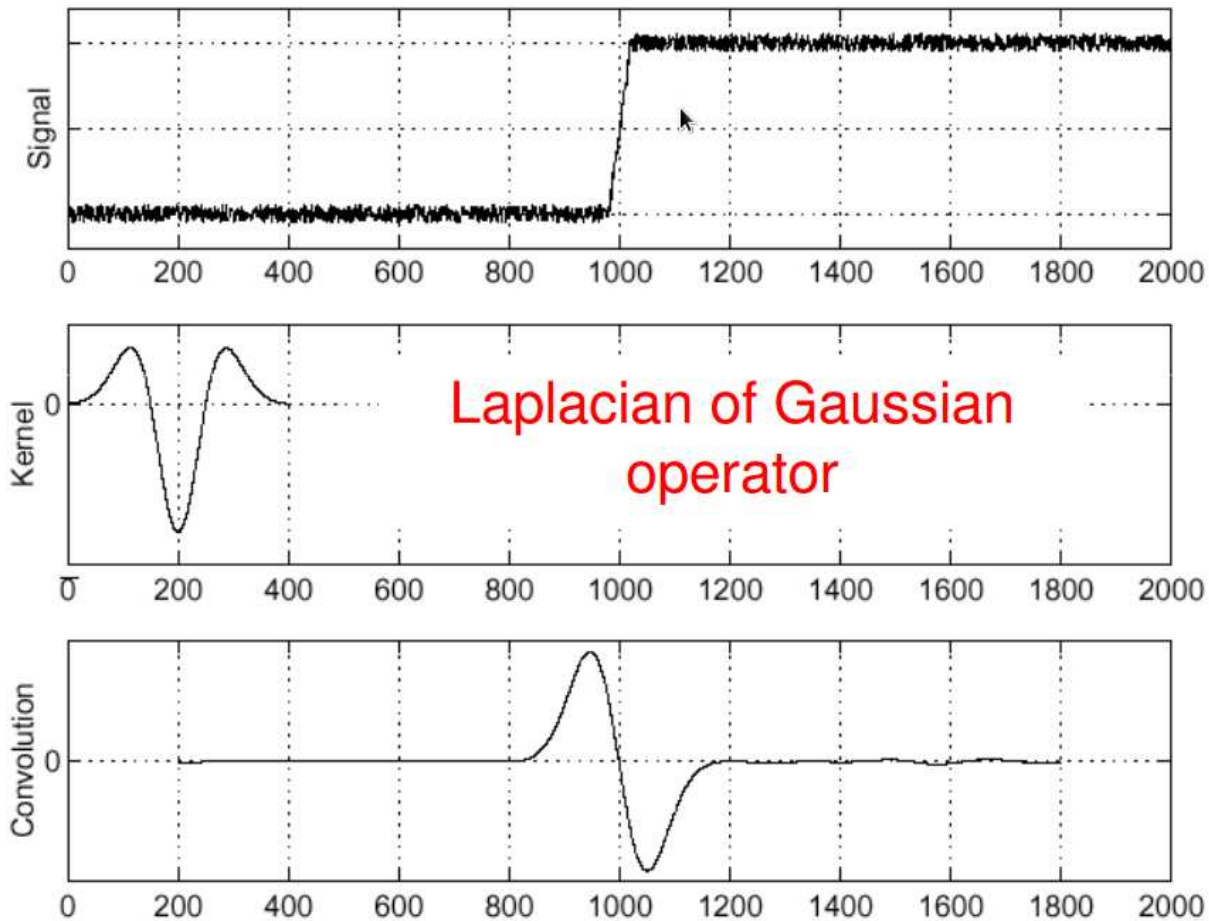
The characteristic scale

Short intro to Laplacian filtering :

f

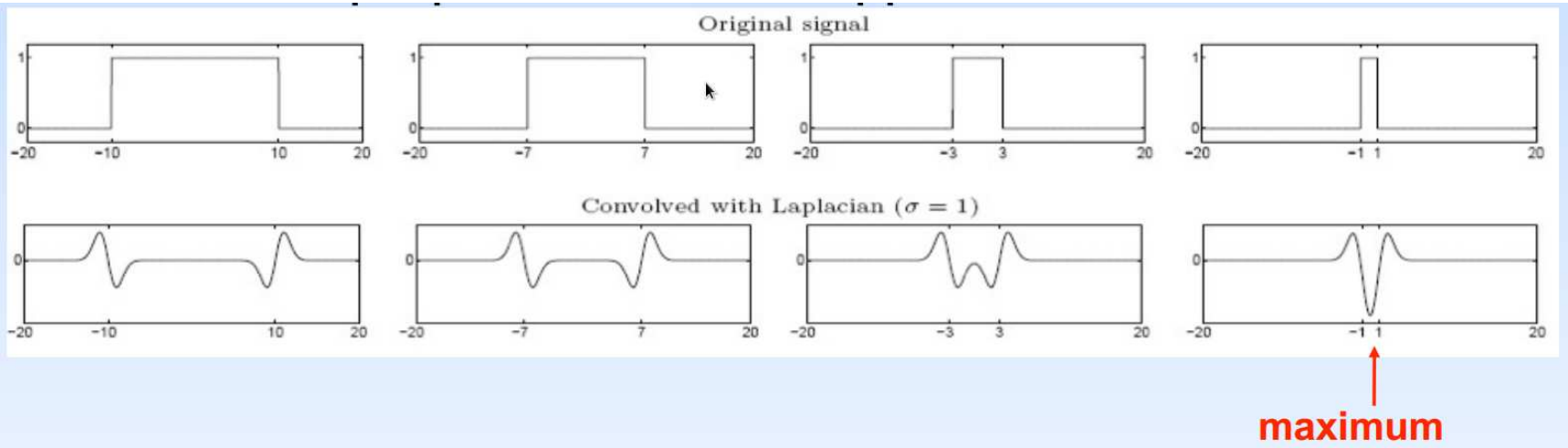
$\frac{\partial^2}{\partial x^2} h$

$(\frac{\partial^2}{\partial x^2} h) \star f$



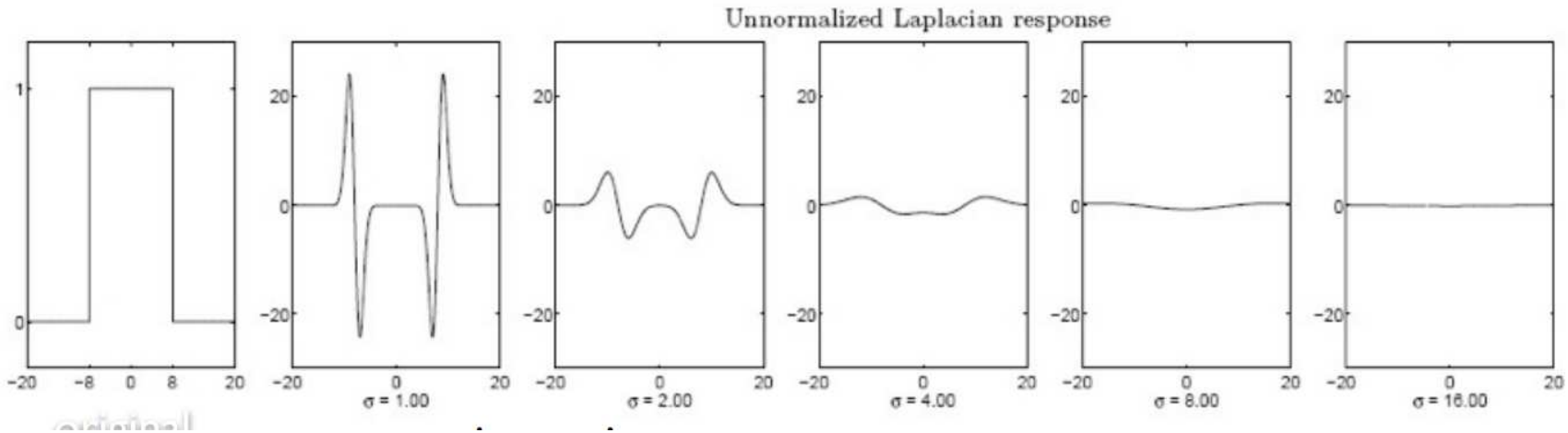
Gaussian filter + Laplace (LoG) - zero crossing

The characteristic scale



The Laplacian response - maximal if the Laplacian scale corresponds to the scale of the variation in the image space

The characteristic scale



If one varies σ, the Laplacien response varies as well ; the operation has to be normalized by a multiplication by σ^2

The characteristic scale

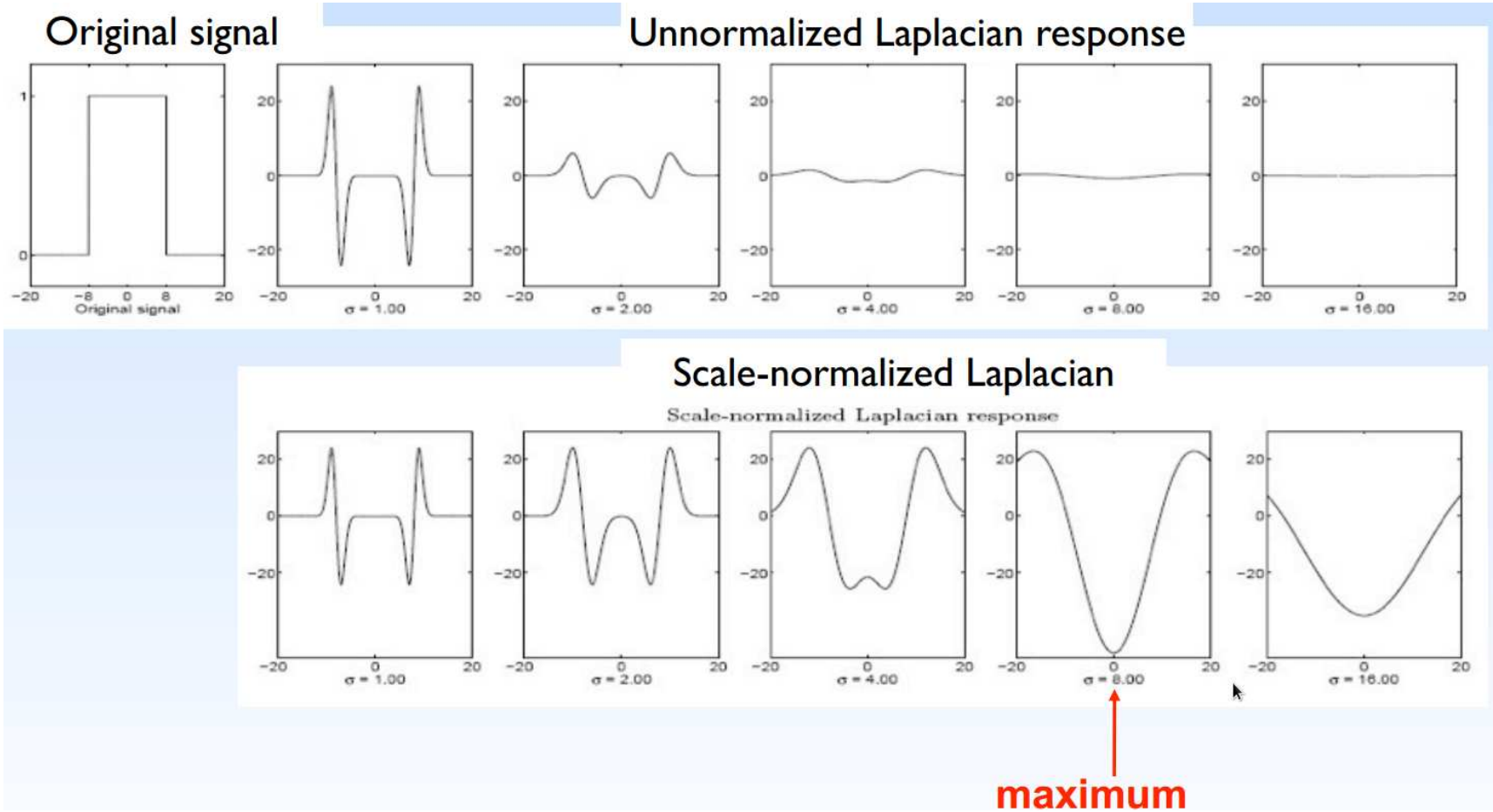
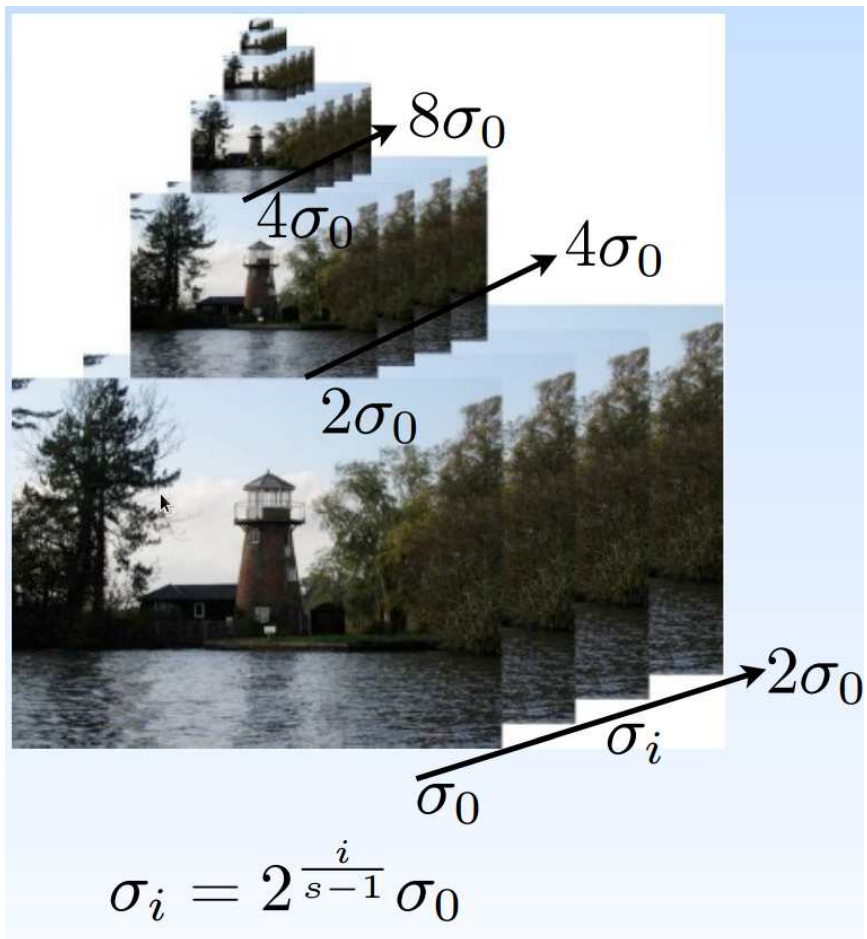


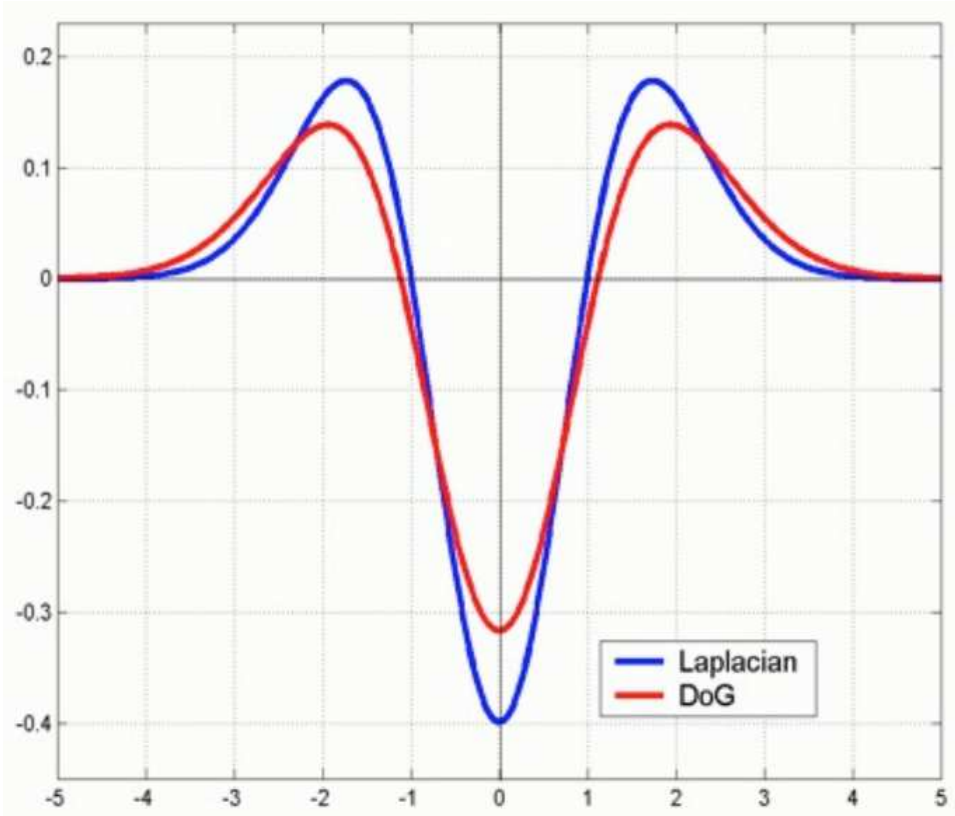
FIGURE – Multi scale normalized Laplacian response

The pyramid representation





Approximating the Laplacian



Laplacian :

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

Difference of Gaussians :

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

The SIFT detector

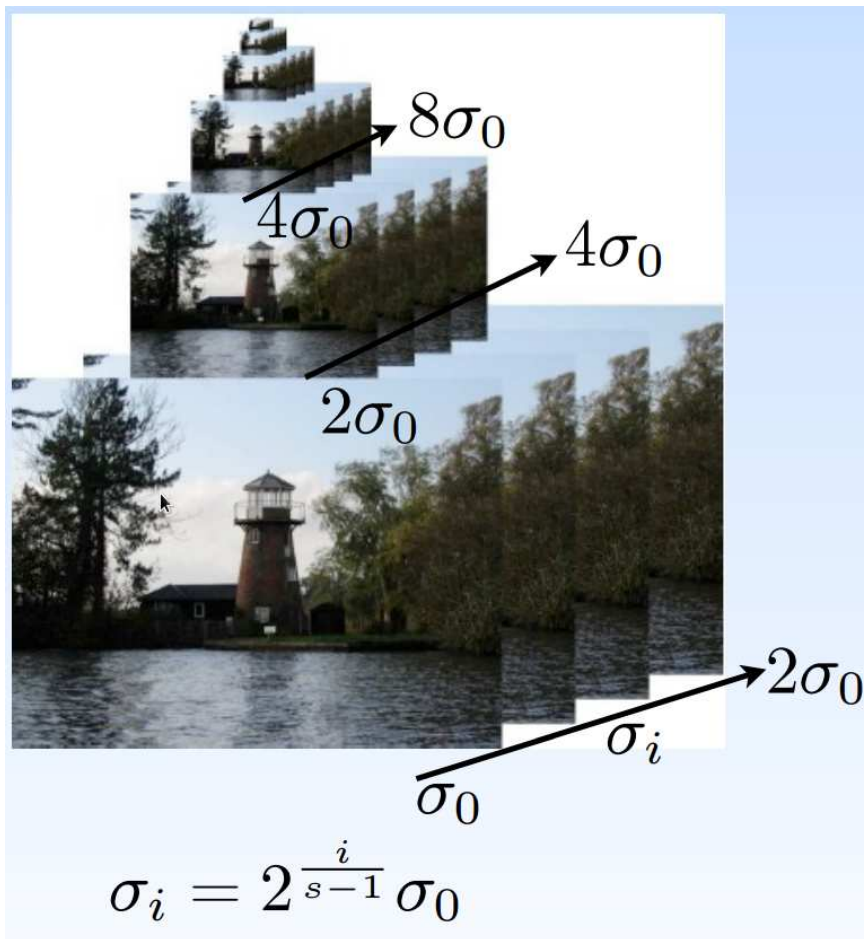
Scale Invariant Feature Transform

- high performance
- very costly
- the descriptor is integrated (it is also provided by the algorithm)

1. Construction of the scale space
2. Computing the DoGs
3. Computing the characteristic scale
4. Sub-pixel localization
5. Eliminating contour responses
6. Computing the orientation
7. Computing the descriptor

The pyramid representation

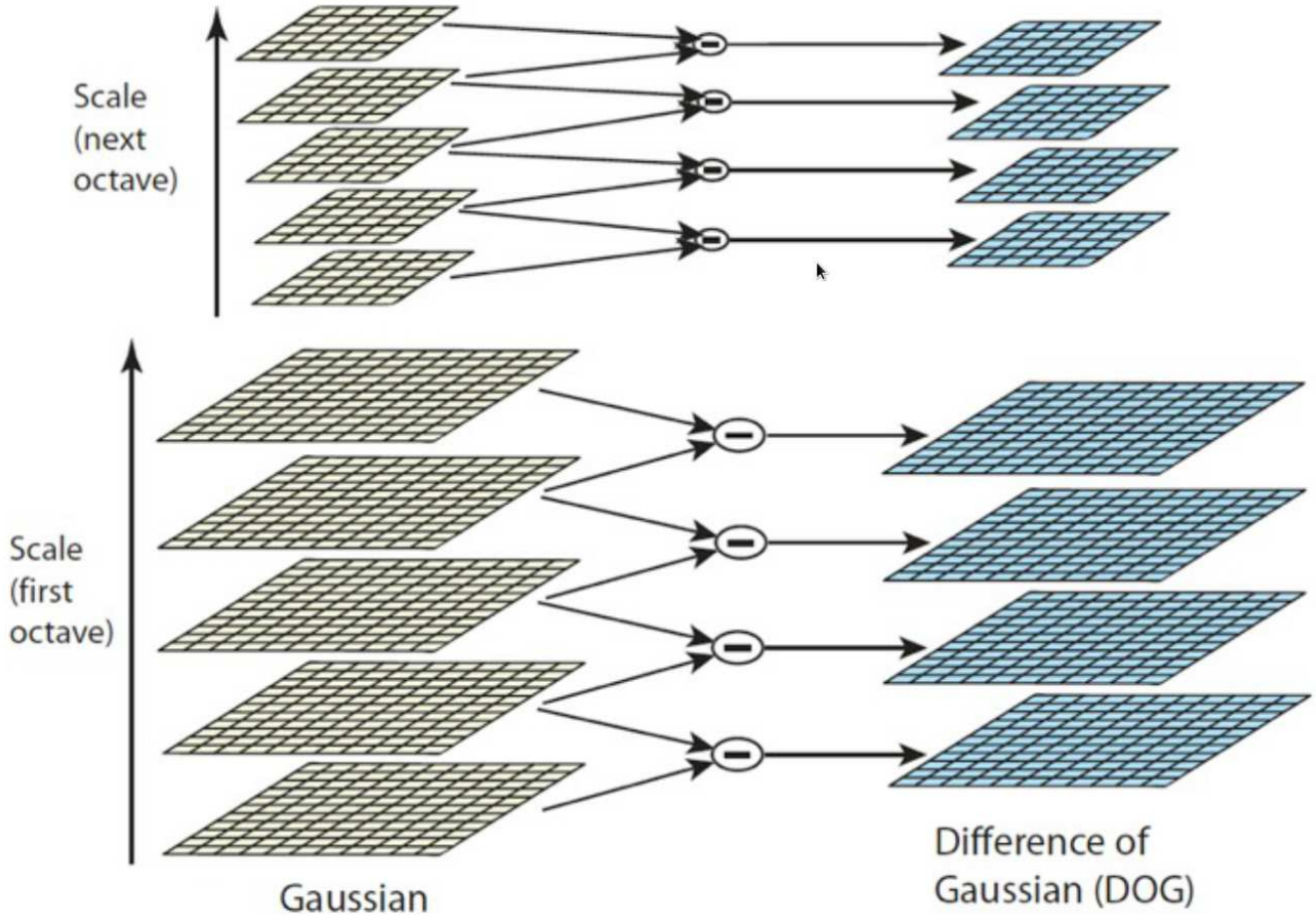




The SIFT detector

1. Construction of the scale space
2. Computing the DoGs
3. Computing the characteristic scale
4. Sub-pixel localization
5. Eliminating contour responses
6. Computing the orientation
7. Computing the descriptor

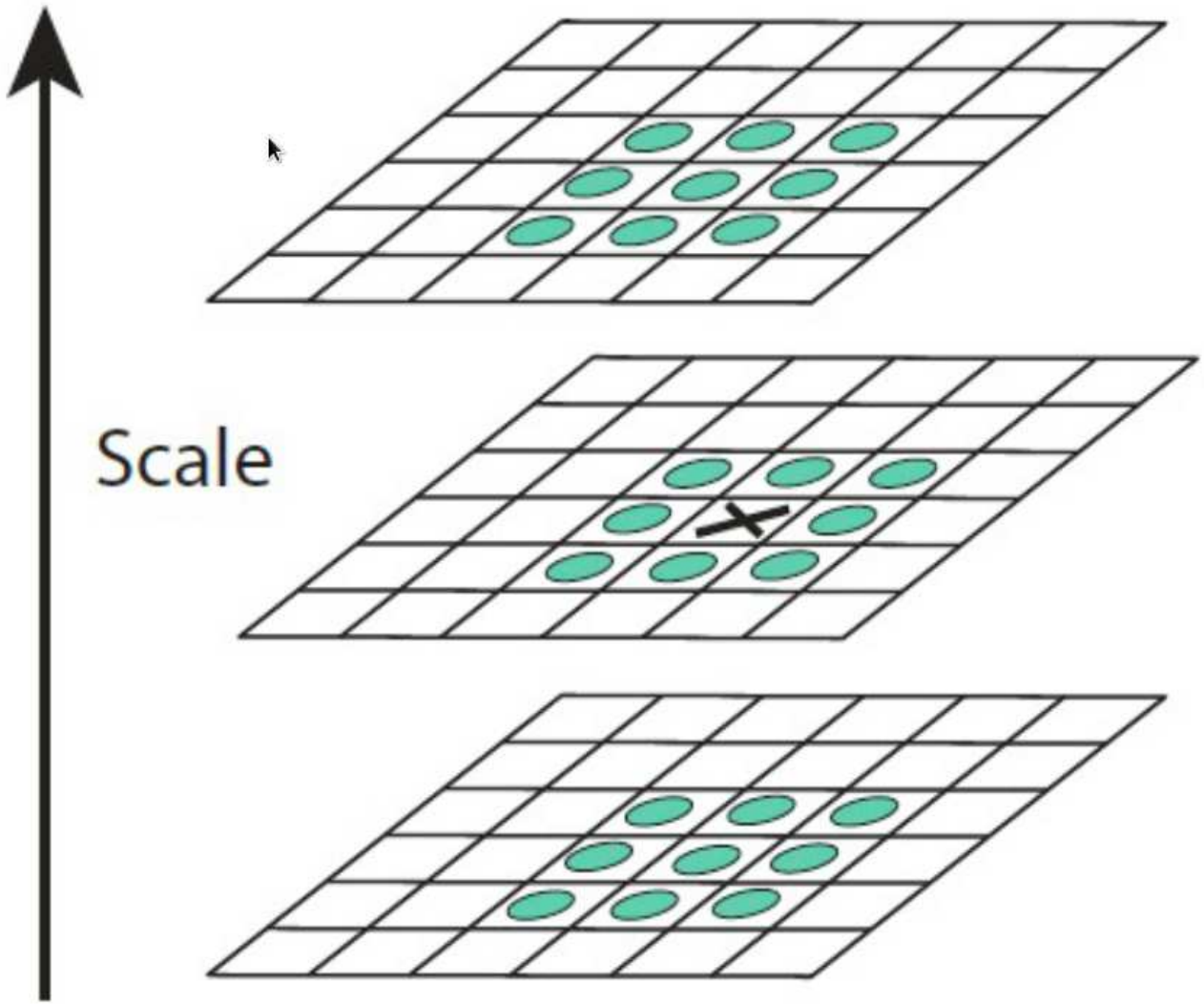
Computing the DoGs



The SIFT detector

1. Construction of the scale space
2. Computing the DoGs
3. Computing the characteristic scale
4. Sub-pixel localization
5. Eliminating contour responses
6. Computing the orientation
7. Computing the descriptor

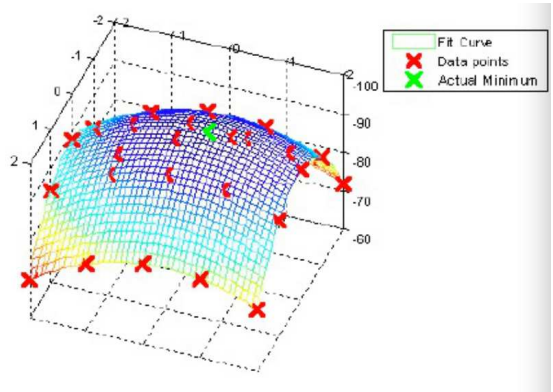
Identifying the extrema



The SIFT detector

1. Construction of the scale space
2. Computing the DoGs
3. Computing the characteristic scale
4. Sub-pixel localization
5. Eliminating contour responses
6. Computing the orientation
7. Computing the descriptor

Sub-pixel localization



Interpolation of discrete values of $D(x, y, \sigma)$. Use of the Taylor series second order development :

$$D(\mathrm{x}) = D + \frac{\partial D}{\partial \mathrm{x}}^T \mathrm{x} + \frac{1}{2}\mathrm{x}^T \frac{\partial^2 D}{\partial \mathrm{x}^2}\mathrm{x}$$

Solution :

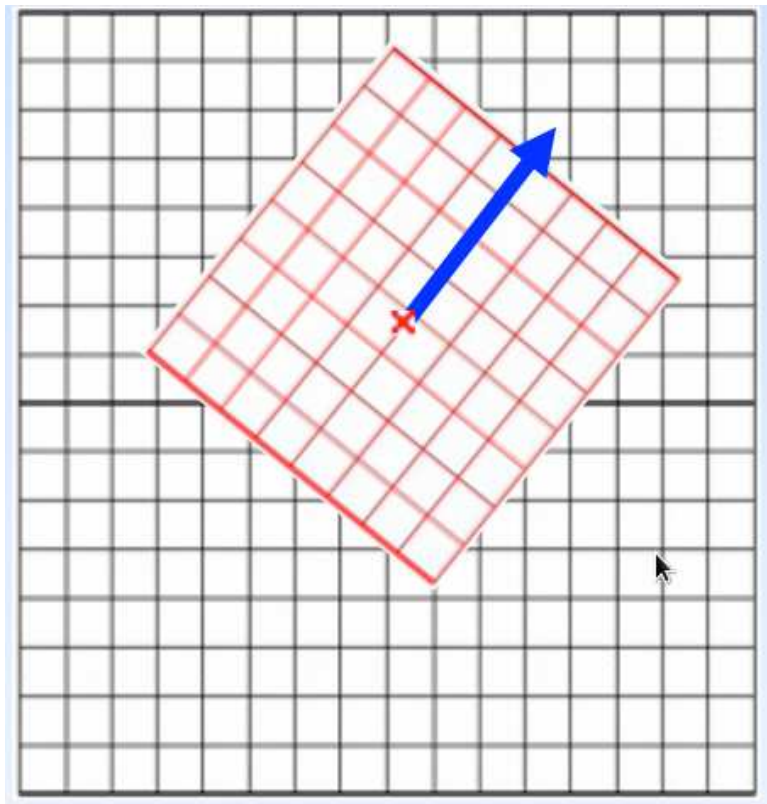
$$\hat{\mathrm{x}} = -\frac{\partial^2 D}{\partial \mathrm{x}^2}^{-1} \frac{\partial D}{\partial \mathrm{x}}$$

The SIFT detector

1. Construction of the scale space
2. Computing the DoGs
3. Computing the characteristic scale
4. Sub-pixel localization
5. Eliminating contour responses
6. Computing the orientation
7. Computing the descriptor

Computing the orientation

1. Compute local gradients at the characteristic scale
2. Compute local gradient histogram
3. The canonic orientation is the maximal direction
4. Each corner is characterized by : location, scale, orientation
5. Local coordinate system for building up the descriptor

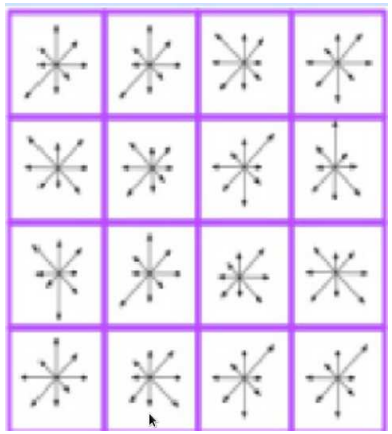


The SIFT detector

1. Construction of the scale space
2. Computing the DoGs
3. Computing the characteristic scale
4. Sub-pixel localization
5. Eliminating contour responses
6. Computing the orientation
7. Computing the descriptor

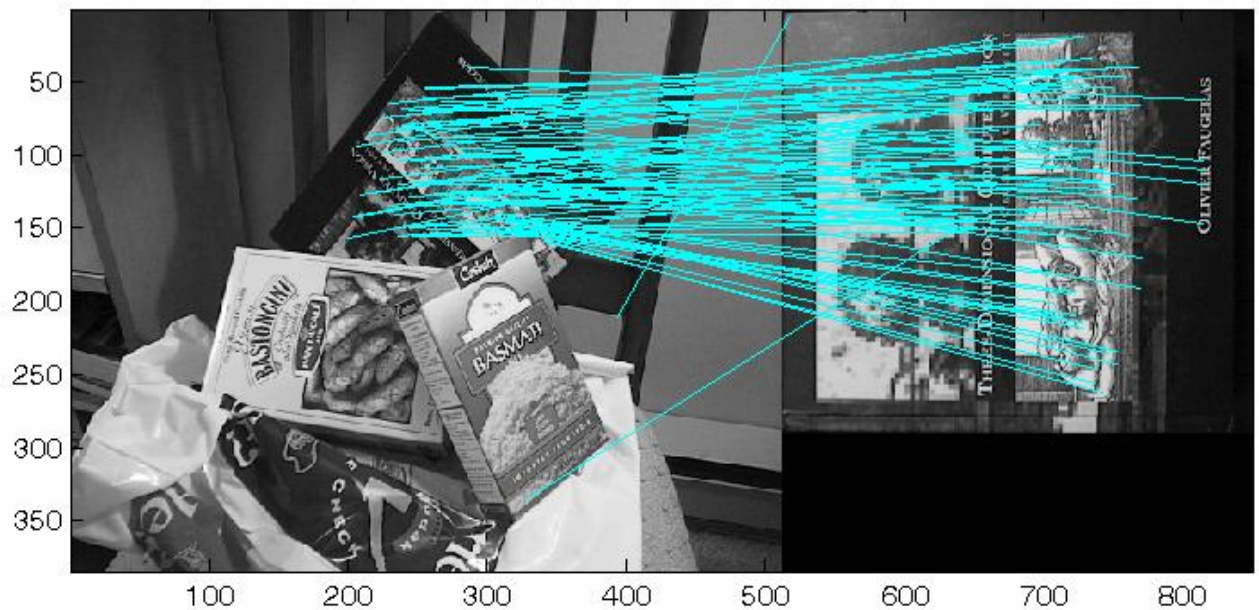
Computing the descriptor

1. Local gradient orientations in 16 neghboring regions
2. Coordinate system defined by the corner
3. 4*4*8 orientations = 128 (descriptor dimension)



Conclusions about SIFT

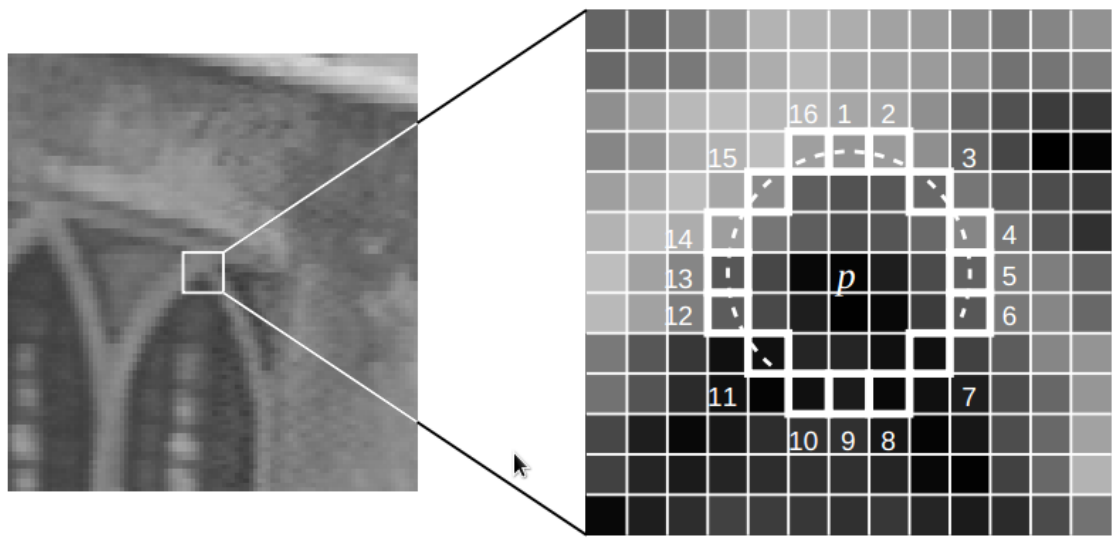
- Scale invariant
- Rotation invariant
- Illumination invariant
- Perspective invariant
- Costly



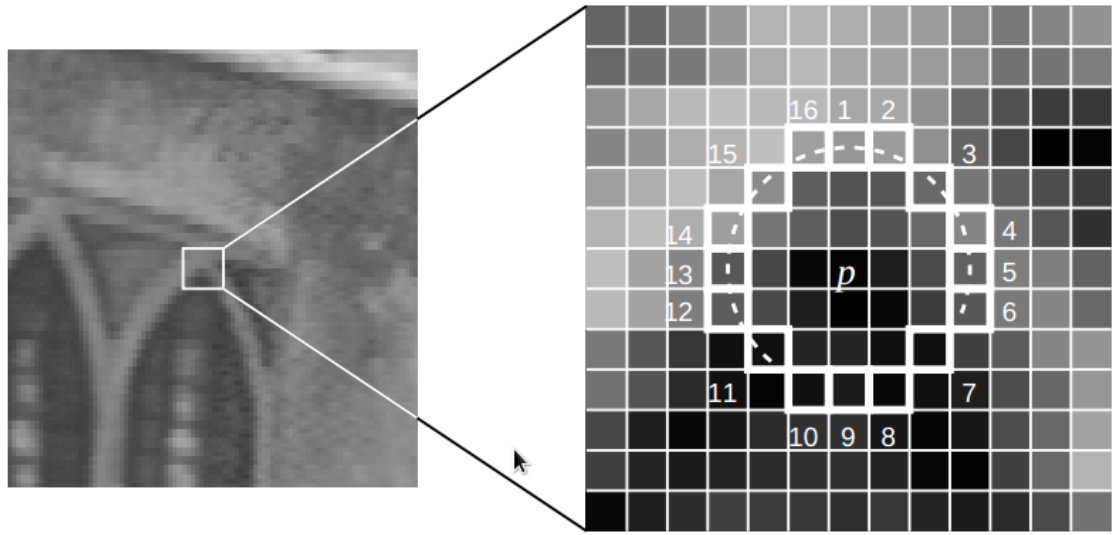
The FAST detector

Features from Accelerated Segment Test

- extremely fast
- no complex operations (convolution, gradient computation etc.)
- not too robust
- no descriptor



The FAST detector - strategy



$$S_{p\to x} = \begin{cases} d, & I_{p\to x} \leq I_p - t \\ s, & I_p - t < I_{p\to x} < I_p + t \\ b, & I_p + t \leq I_{p\to x} \end{cases}$$

The FAST detector

Question 1

Sketch a naive implementation in order to test whether a pixel is a FAST corner or not.

The FAST detector

Question 2

How many possible configurations are in total ?
How many coin configurations $c \in Q$ are there ?
What does the following function :

$$H(Q) = (c + \bar{c}) \log(c + \bar{c}) - c \log c - \bar{c} \log \bar{c}$$

represent ?

The FAST detector

Question 3

Given that the entropy gain is :

$$H_g = H(Q) - H(A) - H(B)$$

where $Q = A \cup B$, think of a trick in order to improve the test that you proposed for Question 1.

Corner association (matching)

How to do it ?

- matching needs to be fast and reliable

Corner association (matching)

How to do it ?

- matching needs to be fast and reliable
- if the detector provides a descriptor (i.e. SIFT), use it for matching

Corner association (matching)

How to do it ?

- matching needs to be fast and reliable
- if the detector provides a descriptor (i.e. SIFT), use it for matching
- otherwise, a simple solution is patch matching : a patch is extracted around the corner, and matched against a candidate in the destination image using a correlation, SSD or SAD function

Corner association (matching)

How to do it ?

- matching needs to be fast and reliable
- if the detector provides a descriptor (i.e. SIFT), use it for matching
- otherwise, a simple solution is patch matching : a patch is extracted around the corner, and matched against a candidate in the destination image using a correlation, SSD or SAD function
- other solutions exist (BRIEF, FREAK etc.)

Corner association (matching)

How to do it ?

- matching needs to be fast and reliable
- if the detector provides a descriptor (i.e. SIFT), use it for matching
- otherwise, a simple solution is patch matching : a patch is extracted around the corner, and matched against a candidate in the destination image using a correlation, SSD or SAD function
- other solutions exist (BRIEF, FREAK etc.)

Tricks used commonly in order to improve matching quality

- these tricks usually increase the computation time but remove false matches (and also some good matches sometimes)

Corner association (matching)

How to do it ?

- matching needs to be fast and reliable
- if the detector provides a descriptor (i.e. SIFT), use it for matching
- otherwise, a simple solution is patch matching : a patch is extracted around the corner, and matched against a candidate in the destination image using a correlation, SSD or SAD function
- other solutions exist (BRIEF, FREAK etc.)

Tricks used commonly in order to improve matching quality

- these tricks usually increase the computation time but remove false matches (and also some good matches sometimes)
- married matching : the best candidate has to pick up the initial corner as best candidate as well

Corner association (matching)

How to do it ?

- matching needs to be fast and reliable
- if the detector provides a descriptor (i.e. SIFT), use it for matching
- otherwise, a simple solution is patch matching : a patch is extracted around the corner, and matched against a candidate in the destination image using a correlation, SSD or SAD function
- other solutions exist (BRIEF, FREAK etc.)

Tricks used commonly in order to improve matching quality

- these tricks usually increase the computation time but remove false matches (and also some good matches sometimes)
- married matching : the best candidate has to pick up the initial corner as best candidate as well
- ranking : the second match must have a significantly larger distance/lower similarity than the best match, in order to avoid confusion between similarly looking corners

Detectors - conclusion

Overview

- FAST : not so robust, no descriptor provided - but runs in 1ms on a regular image ;

Detectors - conclusion

Overview

- FAST : not so robust, no descriptor provided - but runs in 1ms on a regular image ;
- Harris : slightly more robust, no descriptor provided - runs in 25-40ms on a regular image

Detectors - conclusion

Overview

- FAST : not so robust, no descriptor provided - but runs in 1ms on a regular image ;
- Harris : slightly more robust, no descriptor provided - runs in 25-40ms on a regular image
- SIFT : very robust, descriptor provided - runs in 2-5 seconds on a regular image

Detectors - conclusion

Overview

- FAST : not so robust, no descriptor provided - but runs in 1ms on a regular image ;
- Harris : slightly more robust, no descriptor provided - runs in 25-40ms on a regular image
- SIFT : very robust, descriptor provided - runs in 2-5 seconds on a regular image
- plenty other detectors which provide some advantage in terms of either computational time or some invariance : SURF, AGAST, ORB, HOOFR etc.

Detectors - conclusion

Overview

- FAST : not so robust, no descriptor provided - but runs in 1ms on a regular image ;
- Harris : slightly more robust, no descriptor provided - runs in 25-40ms on a regular image
- SIFT : very robust, descriptor provided - runs in 2-5 seconds on a regular image
- plenty other detectors which provide some advantage in terms of either computational time or some invariance : SURF, AGAST, ORB, HOOFR etc.

Which detector to choose ?

- the choice is application dependent

Detectors - conclusion

Overview

- FAST : not so robust, no descriptor provided - but runs in 1ms on a regular image ;
- Harris : slightly more robust, no descriptor provided - runs in 25-40ms on a regular image
- SIFT : very robust, descriptor provided - runs in 2-5 seconds on a regular image
- plenty other detectors which provide some advantage in terms of either computational time or some invariance : SURF, AGAST, ORB, HOOFR etc.

Which detector to choose ?

- the choice is application dependent
- FAST : great for real time robotic navigation

Detectors - conclusion

Overview

- FAST : not so robust, no descriptor provided - but runs in 1ms on a regular image ;
- Harris : slightly more robust, no descriptor provided - runs in 25-40ms on a regular image
- SIFT : very robust, descriptor provided - runs in 2-5 seconds on a regular image
- plenty other detectors which provide some advantage in terms of either computational time or some invariance : SURF, AGAST, ORB, HOOFR etc.

Which detector to choose ?

- the choice is application dependent
- FAST : great for real time robotic navigation
- SIFT : useful when quality is important

Detectors - conclusion

Overview

- FAST : not so robust, no descriptor provided - but runs in 1ms on a regular image ;
- Harris : slightly more robust, no descriptor provided - runs in 25-40ms on a regular image
- SIFT : very robust, descriptor provided - runs in 2-5 seconds on a regular image
- plenty other detectors which provide some advantage in terms of either computational time or some invariance : SURF, AGAST, ORB, HOOFR etc.

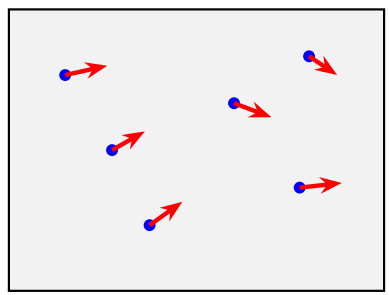
Which detector to choose ?

- the choice is application dependent
- FAST : great for real time robotic navigation
- SIFT : useful when quality is important
- most other descriptors provide a compromise between robustness and cost

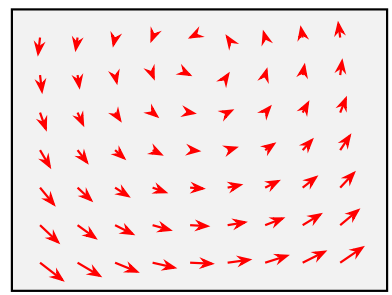
Beyond corners : dense matching

Sparse vs dense correspondences

- sparse : a few hundred corners, matched with descriptors
- dense : one displacement vector for *every* pixel



sparse : corners only

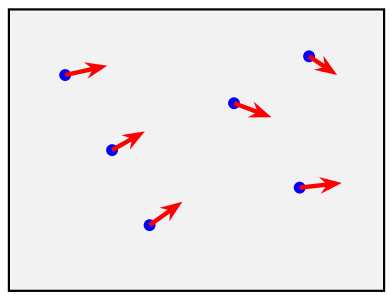


dense : every pixel

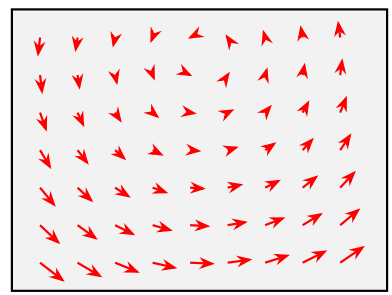
Beyond corners : dense matching

Sparse vs dense correspondences

- sparse : a few hundred corners, matched with descriptors
- dense : one displacement vector for *every* pixel



sparse : corners only



dense : every pixel

Typical applications

- optical flow : apparent motion between two video frames
- stereo : a disparity (hence a depth) for every pixel
- image registration : medical and satellite images, video stabilization

Dense matching : the problem

Displacement field

For every pixel x $= (x, y)$ of I_1, find u(x) $= \big(u(x, y), v(x, y)\big)$ such that

$$I_2(x + u,\ y + v) \approx I_1(x, y)$$

Dense matching : the problem

Displacement field

For every pixel x $= (x, y)$ of I_1, find u(x) $= \big(u(x, y), v(x, y)\big)$ such that

$$I_2(x + u,\ y + v) \approx I_1(x, y)$$

- brightness constancy : a scene point keeps its intensity when it moves

Dense matching : the problem

Displacement field

For every pixel $\mathrm{x} = (x, y)$ of I_1, find $\mathrm{u}(\mathrm{x}) = \big(u(x,y), v(x,y)\big)$ such that

$$I_2(x+u,\, y+v) \approx I_1(x,y)$$

- brightness constancy : a scene point keeps its intensity when it moves
- natural cost : $E(\mathrm{u}) = \sum_{\mathrm{x}} \big[I_2(\mathrm{x}+\mathrm{u}(\mathrm{x})) - I_1(\mathrm{x})\big]^2$

Dense matching : the problem

Displacement field

For every pixel $\mathrm{x} = (x, y)$ of I_1, find $\mathrm{u}(\mathrm{x}) = \big(u(x,y), v(x,y)\big)$ such that

$$I_2(x+u,\, y+v) \approx I_1(x,y)$$

- brightness constancy : a scene point keeps its intensity when it moves
- natural cost : $E(\mathrm{u}) = \sum_{\mathrm{x}} \big[I_2(\mathrm{x}+\mathrm{u}(\mathrm{x})) - I_1(\mathrm{x})\big]^2$
- same function as E in the Harris detector, but now we look for the shift that makes the patches the *most similar*

Dense matching : the problem

Displacement field

For every pixel $\mathrm{x} = (x, y)$ of I_1, find $\mathrm{u}(\mathrm{x}) = \big(u(x,y), v(x,y)\big)$ such that

$$I_2(x+u,\, y+v) \approx I_1(x,y)$$

- brightness constancy : a scene point keeps its intensity when it moves
- natural cost : $E(\mathrm{u}) = \sum_{\mathrm{x}} \big[I_2(\mathrm{x}+\mathrm{u}(\mathrm{x})) - I_1(\mathrm{x})\big]^2$
- same function as E in the Harris detector, but now we look for the shift that makes the patches the *most similar*
- 2 unknowns, 1 measurement per pixel : ill-posed, extra assumptions are needed

Dense matching : the problem

Displacement field

For every pixel $\mathrm{x} = (x, y)$ of I_1, find $\mathrm{u}(\mathrm{x}) = \big(u(x,y), v(x,y)\big)$ such that

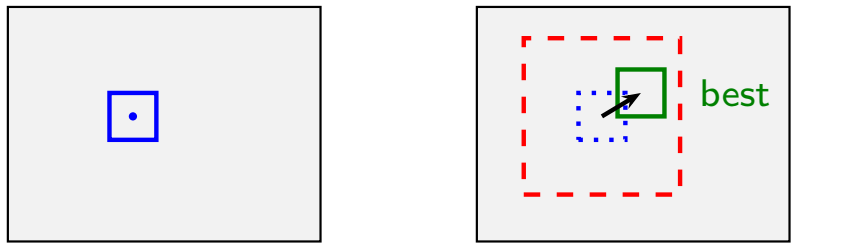
$$I_2(x+u,\, y+v) \approx I_1(x,y)$$

- brightness constancy : a scene point keeps its intensity when it moves
- natural cost : $E(\mathrm{u}) = \sum_{\mathrm{x}} \big[I_2(\mathrm{x}+\mathrm{u}(\mathrm{x})) - I_1(\mathrm{x})\big]^2$
- same function as E in the Harris detector, but now we look for the shift that makes the patches the *most similar*
- 2 unknowns, 1 measurement per pixel : ill-posed, extra assumptions are needed

Three families of assumptions

- local : u constant in a small patch $\rightarrow$ block matching, Lucas–Kanade
- global : u varies smoothly over the image $\rightarrow$ Horn–Schunck
- parametric : one transform for the whole image $\rightarrow$ registration

The simplest dense method : block matching



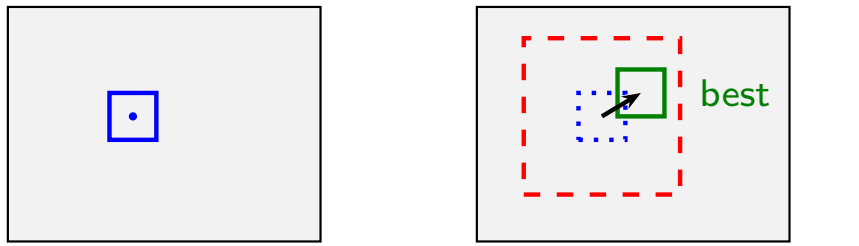
I_1 : patch $K \times K$ I_2 : search window $W \times W$

For every pixel x of I_1

compare its $K \times K$ patch $P(x)$ with all the candidates of the search window, keep the best one (*winner takes all*) :

$$u(x) = \arg \min_{d \in \text{window}} \sum_{p \in P(x)} \left[I_2(p+d) - I_1(p) \right]^2$$

The simplest dense method : block matching



I_1 : patch $K \times K$ I_2 : search window $W \times W$

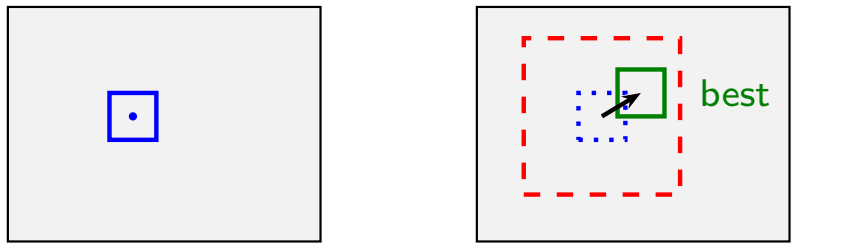
For every pixel x of I_1

compare its $K \times K$ patch $P(x)$ with all the candidates of the search window, keep the best one (*winner takes all*) :

$$u(x) = \arg \min_{d \in \text{window}} \sum_{p \in P(x)} \left[I_2(p+d) - I_1(p) \right]^2$$

- same scores as for matching corners : SSD, SAD, NCC (robust to illumination changes)

The simplest dense method : block matching



I_1 : patch $K \times K$ I_2 : search window $W \times W$

For every pixel x of I_1

compare its $K \times K$ patch $P(\mathrm{x})$ with all the candidates of the search window, keep the best one (*winner takes all*) :

$$\mathrm{u}(\mathrm{x}) = \arg \min_{\mathrm{d} \in \mathrm{window}} \sum_{\mathrm{p} \in P(\mathrm{x})} \left[I_2(\mathrm{p}+\mathrm{d}) - I_1(\mathrm{p}) \right]^2$$

- same scores as for matching corners : SSD, SAD, NCC (robust to illumination changes)
- stereo with rectified cameras : 1D search along the same row $\rightarrow$ disparity d, depth $Z = fB/d$ (f : focal length, B : baseline)

Block matching : questions

Question 1

What is the computational cost of block matching for an $N \times N$ image, a $K \times K$ patch and a $W \times W$ search window ? Give an order of magnitude for $N = 1000$, $K = 9$, $W = 64$.

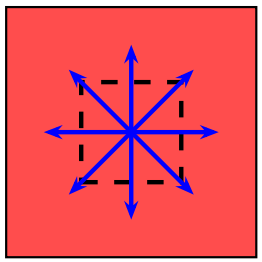
Question 2

For a fixed displacement d, neighboring patches share most of their squared differences. How can you get rid of the K^2 factor ? (hint : box filter / integral image)

Question 3

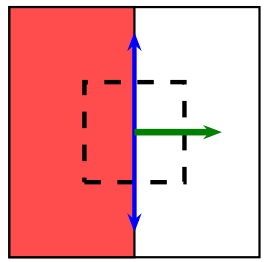
For which pixels do you expect the estimated displacement to be reliable ?

When does local matching fail ?



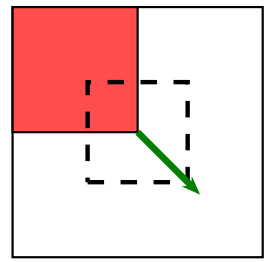
flat :

any shift fits



edge : shift along

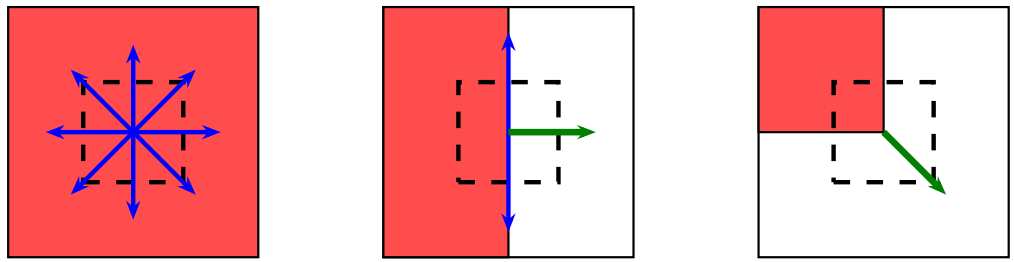
the edge unknown



corner :

unique match

When does local matching fail ?



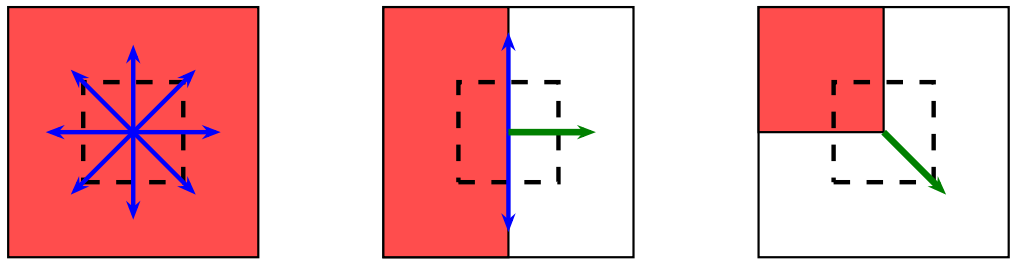
flat : any shift fits

edge : shift along the edge unknown

corner : unique match

- homogeneous regions : all candidate patches look alike

When does local matching fail ?



flat :

any shift fits

edge : shift along

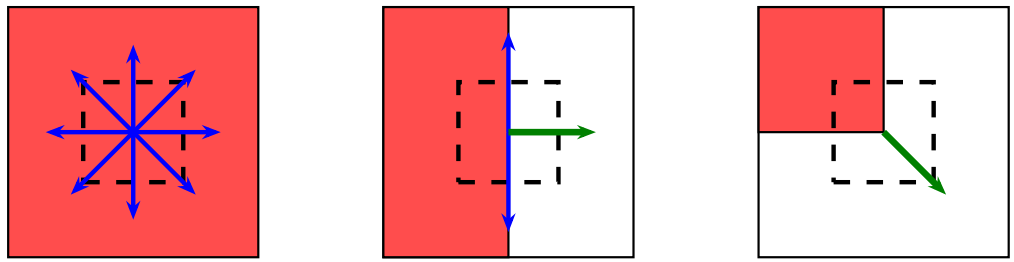
the edge unknown

corner :

unique match

- homogeneous regions : all candidate patches look alike
- edges : only the displacement perpendicular to the edge can be measured : the aperture problem

When does local matching fail ?



flat :

any shift fits

edge : shift along

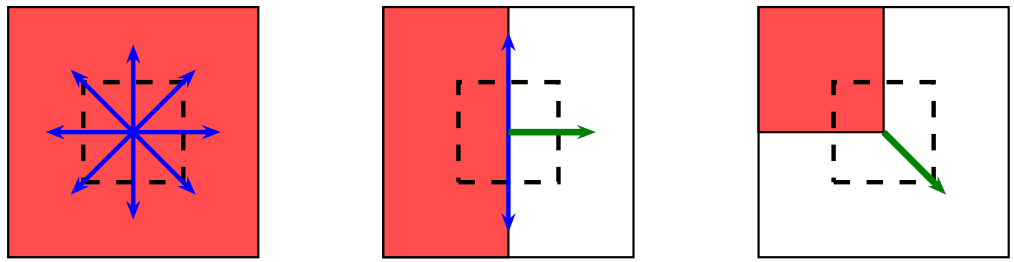
the edge unknown

corner :

unique match

- homogeneous regions : all candidate patches look alike
- edges : only the displacement perpendicular to the edge can be measured : the aperture problem
- also : repetitive textures, occlusions, illumination changes

When does local matching fail ?



flat : any shift fits

edge : shift along the edge unknown

corner : unique match

- homogeneous regions : all candidate patches look alike
- edges : only the displacement perpendicular to the edge can be measured : the aperture problem
- also : repetitive textures, occlusions, illumination changes

Remember the structure tensor

A patch can be matched reliably only if it changes strongly for every shift : λ_1 and λ_2 both large, i.e. near a corner !

Optical flow : the differential approach

Video $I(x, y, t)$, small displacement (u, v) between times t and $t + 1$:

$$I(x + u,\ y + v,\ t + 1) = I(x, y, t)$$

Optical flow : the differential approach

Video $I(x, y, t)$, small displacement (u, v) between times t and $t + 1$:

$$I(x + u,\ y + v,\ t + 1) = I(x, y, t)$$

First order Taylor development (as for the Harris detector)

$$I(x + u,\ y + v,\ t + 1) \approx I(x, y, t) + I_x u + I_y v + I_t$$

Optical flow : the differential approach

Video $I(x, y, t)$, small displacement (u, v) between times t and $t + 1$:

$$I(x + u,\ y + v,\ t + 1) = I(x, y, t)$$

First order Taylor development (as for the Harris detector)

$$I(x + u,\ y + v,\ t + 1) \approx I(x, y, t) + I_x u + I_y v + I_t$$

Optical flow constraint

$$I_x u + I_y v + I_t = 0 \qquad \Leftrightarrow \qquad \nabla I^T \mathrm{u} = -I_t$$

Optical flow : the differential approach

Video $I(x, y, t)$, small displacement (u, v) between times t and $t + 1$:

$$I(x + u,\ y + v,\ t + 1) = I(x, y, t)$$

First order Taylor development (as for the Harris detector)

$$I(x + u,\ y + v,\ t + 1) \approx I(x, y, t) + I_x u + I_y v + I_t$$

Optical flow constraint

$$I_x u + I_y v + I_t = 0 \qquad \Leftrightarrow \qquad \nabla I^T \mathrm{u} = -I_t$$

- I_x, I_y : spatial gradients ; $I_t \approx I(x, y, t + 1) - I(x, y, t)$: temporal derivative

Optical flow : the differential approach

Video $I(x, y, t)$, small displacement (u, v) between times t and $t + 1$:

$$I(x + u,\ y + v,\ t + 1) = I(x, y, t)$$

First order Taylor development (as for the Harris detector)

$$I(x + u,\ y + v,\ t + 1) \approx I(x, y, t) + I_x u + I_y v + I_t$$

Optical flow constraint

$$I_x u + I_y v + I_t = 0 \qquad \Leftrightarrow \qquad \nabla I^T \mathrm{u} = -I_t$$

- I_x, I_y : spatial gradients ; $I_t \approx I(x, y, t + 1) - I(x, y, t)$: temporal derivative
- one equation, two unknowns : only the component of u along ∇I can be recovered (the aperture problem again)

Optical flow : the differential approach

Video $I(x, y, t)$, small displacement (u, v) between times t and $t + 1$:

$$I(x + u,\ y + v,\ t + 1) = I(x, y, t)$$

First order Taylor development (as for the Harris detector)

$$I(x + u,\ y + v,\ t + 1) \approx I(x, y, t) + I_x u + I_y v + I_t$$

Optical flow constraint

$$I_x u + I_y v + I_t = 0 \qquad \Leftrightarrow \qquad \nabla I^T \mathrm{u} = -I_t$$

- I_x, I_y : spatial gradients ; $I_t \approx I(x, y, t + 1) - I(x, y, t)$: temporal derivative
- one equation, two unknowns : only the component of u along ∇I can be recovered (the aperture problem again)
- valid only for small displacements (about one pixel)

Lucas–Kanade : constant flow in a window

Assumption : u is constant in a small window around x $\rightarrow$ least squares :

$$E(u, v) = \sum_{\text{window}} \left[I_x u + I_y v + I_t \right]^2$$

Lucas–Kanade : constant flow in a window

Assumption : u is constant in a small window around x $\rightarrow$ least squares :

$$E(u,v) = \sum_{\text{window}} \left[I_x u + I_y v + I_t \right]^2$$

Setting $\frac{\partial E}{\partial u} = \frac{\partial E}{\partial v} = 0$:

$$\underbrace{\begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}}_{M = \text{the structure tensor}} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

Lucas–Kanade : constant flow in a window

Assumption : u is constant in a small window around x $\rightarrow$ least squares :

$$E(u,v) = \sum_{\text{window}} \left[I_x u + I_y v + I_t \right]^2$$

Setting $\frac{\partial E}{\partial u} = \frac{\partial E}{\partial v} = 0$:

$$\underbrace{\begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}}_{M = \text{the structure tensor}} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

- solvable and stable only if λ_1, λ_2 are both large : corners again

Lucas–Kanade : constant flow in a window

Assumption : u is constant in a small window around x $\rightarrow$ least squares :

$$E(u,v) = \sum_{\text{window}} \left[I_x u + I_y v + I_t \right]^2$$

Setting $\frac{\partial E}{\partial u} = \frac{\partial E}{\partial v} = 0$:

$$\underbrace{\begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}}_{M = \text{the structure tensor}} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

- solvable and stable only if λ_1, λ_2 are both large : corners again
- flat regions and edges : M is (nearly) singular, no reliable flow

Lucas–Kanade : constant flow in a window

Assumption : u is constant in a small window around x $\rightarrow$ least squares :

$$E(u,v) = \sum_{\text{window}} \left[I_x u + I_y v + I_t \right]^2$$

Setting $\frac{\partial E}{\partial u} = \frac{\partial E}{\partial v} = 0$:

$$\underbrace{\begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}}_{M = \text{the structure tensor}} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

- solvable and stable only if λ_1, λ_2 are both large : corners again
- flat regions and edges : M is (nearly) singular, no reliable flow
- Lucas–Kanade computed at detected corners = the KLT tracker, widely used for tracking and visual odometry

Horn–Schunck : global smoothness

Idea : neighboring pixels usually move together $\rightarrow$ penalize the variations of the flow and estimate all the vectors jointly :

$$E(u,v) = \sum_{x} \underbrace{\left[I_x u + I_y v + I_t\right]^2}_{\text{data term}} + \lambda \underbrace{\left(\|\nabla u\|^2 + \|\nabla v\|^2\right)}_{\text{smoothness term}}$$

Horn–Schunck : global smoothness

Idea : neighboring pixels usually move together $\rightarrow$ penalize the variations of the flow and estimate all the vectors jointly :

$$E(u,v) = \sum_{x} \underbrace{\left[I_x u + I_y v + I_t\right]^2}_{\text{data term}} + \lambda \underbrace{\left(\|\nabla u\|^2 + \|\nabla v\|^2\right)}_{\text{smoothness term}}$$

- the flow propagates from textured areas into homogeneous ones : a truly dense result

Horn–Schunck : global smoothness

Idea : neighboring pixels usually move together $\rightarrow$ penalize the variations of the flow and estimate all the vectors jointly :

$$E(u,v) = \sum_{x} \underbrace{\left[I_x u + I_y v + I_t\right]^2}_{\text{data term}} + \lambda \underbrace{\left(\|\nabla u\|^2 + \|\nabla v\|^2\right)}_{\text{smoothness term}}$$

- the flow propagates from textured areas into homogeneous ones : a truly dense result
- λ balances the two terms : a large λ gives a smooth flow, but blurred motion boundaries

Horn–Schunck : global smoothness

Idea : neighboring pixels usually move together $\rightarrow$ penalize the variations of the flow and estimate all the vectors jointly :

$$E(u,v) = \sum_{x} \underbrace{\left[I_x u + I_y v + I_t\right]^2}_{\text{data term}} + \lambda \underbrace{\left(\|\nabla u\|^2 + \|\nabla v\|^2\right)}_{\text{smoothness term}}$$

- the flow propagates from textured areas into homogeneous ones : a truly dense result
- λ balances the two terms : a large λ gives a smooth flow, but blurred motion boundaries
- solved iteratively : each vector is pulled towards the average of its neighbors, and towards its own constraint line

Horn–Schunck : global smoothness

Idea : neighboring pixels usually move together $\rightarrow$ penalize the variations of the flow and estimate all the vectors jointly :

$$E(u,v) = \sum_{x} \underbrace{\left[I_x u + I_y v + I_t\right]^2}_{\text{data term}} + \lambda \underbrace{\left(\|\nabla u\|^2 + \|\nabla v\|^2\right)}_{\text{smoothness term}}$$

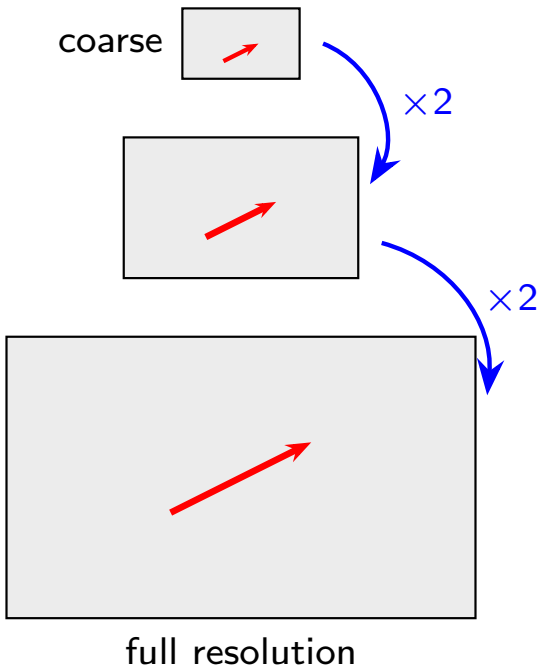
- the flow propagates from textured areas into homogeneous ones : a truly dense result
- λ balances the two terms : a large λ gives a smooth flow, but blurred motion boundaries
- solved iteratively : each vector is pulled towards the average of its neighbors, and towards its own constraint line

A general template

data term + regularization : this is the structure of most dense methods (stereo, registration, . . .) and, as we will see, of some neural network losses

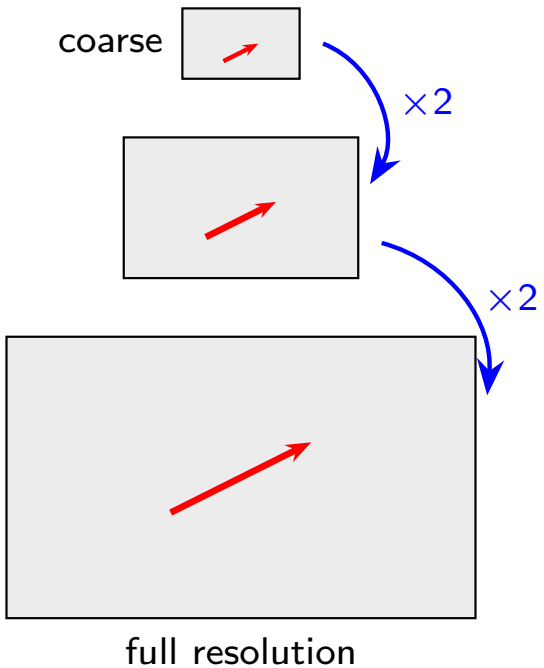
Large displacements : coarse to fine

The Taylor approximation (and the cost of the search window) limits us to small displacements.



Large displacements : coarse to fine

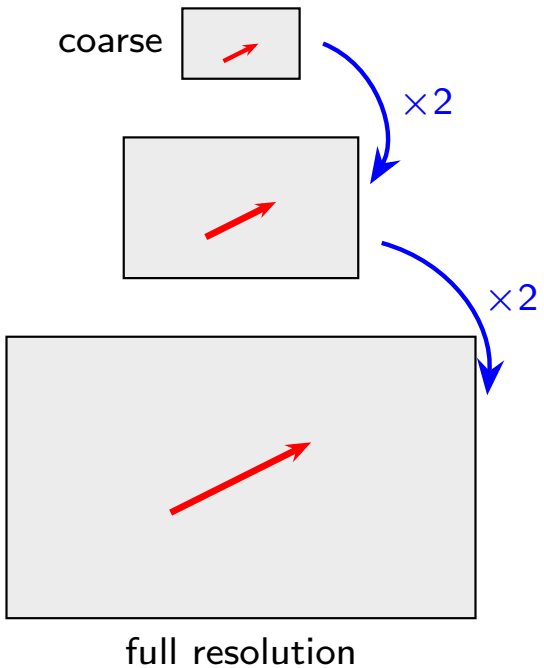
The Taylor approximation (and the cost of the search window) limits us to small displacements.



1. build Gaussian pyramids of I_1 and I_2 (as for SIFT)

Large displacements : coarse to fine

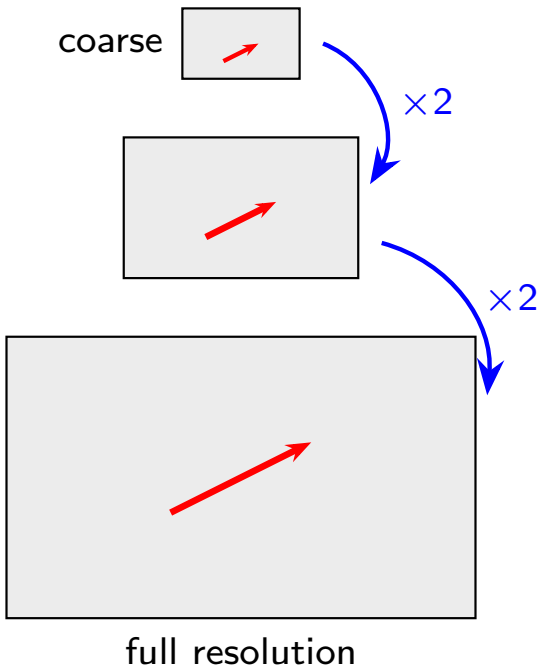
The Taylor approximation (and the cost of the search window) limits us to small displacements.



1. build Gaussian pyramids of I_1 and I_2 (as for SIFT)
2. estimate the flow at the coarsest level : a 16 pixel motion is only 2 pixels after 3 halvings

Large displacements : coarse to fine

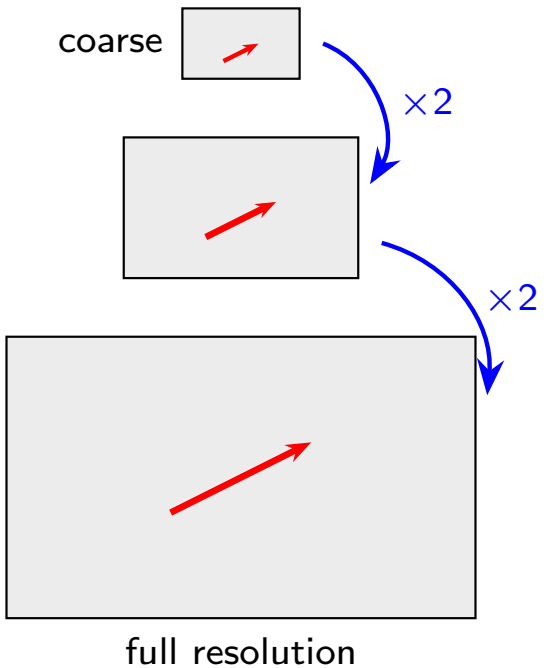
The Taylor approximation (and the cost of the search window) limits us to small displacements.



1. build Gaussian pyramids of I_1 and I_2 (as for SIFT)
2. estimate the flow at the coarsest level : a 16 pixel motion is only 2 pixels after 3 halvings
3. upsample the flow ($\times 2$ in size and in magnitude), warp I_2 towards I_1, estimate the small residual flow

Large displacements : coarse to fine

The Taylor approximation (and the cost of the search window) limits us to small displacements.



1. build Gaussian pyramids of I_1 and I_2 (as for SIFT)
2. estimate the flow at the coarsest level : a 16 pixel motion is only 2 pixels after 3 halvings
3. upsample the flow ($\times 2$ in size and in magnitude), warp I_2 towards I_1, estimate the small residual flow
4. repeat down to full resolution

Image registration : one global transform

If all the pixels follow one model (affine, homography... : see *The core of the problem*), estimate its parameters p directly from the intensities :

$$\hat{\mathrm{p}} = \arg\min_{\mathrm{p}} \sum_{\mathrm{x}} \left[I_2(W(\mathrm{x};\mathrm{p})) - I_1(\mathrm{x}) \right]^2$$

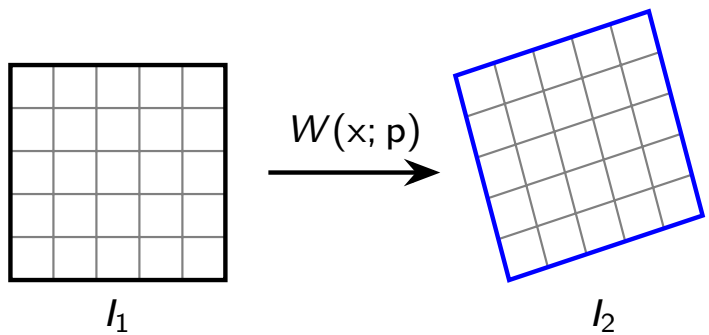
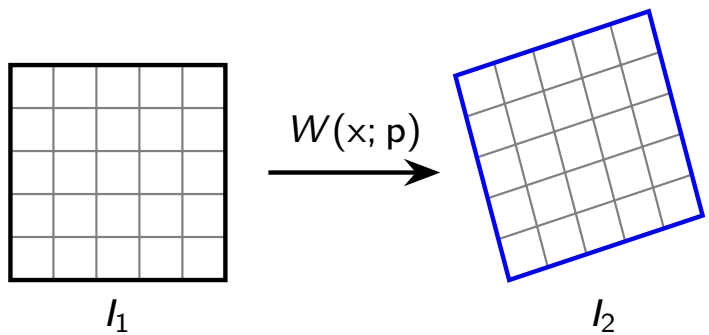


Image registration : one global transform

If all the pixels follow one model (affine, homography... : see *The core of the problem*), estimate its parameters p directly from the intensities :

$$\hat{\mathrm{p}} = \arg\min_{\mathrm{p}} \sum_{\mathrm{x}} \left[I_2(W(\mathrm{x};\mathrm{p})) - I_1(\mathrm{x}) \right]^2$$

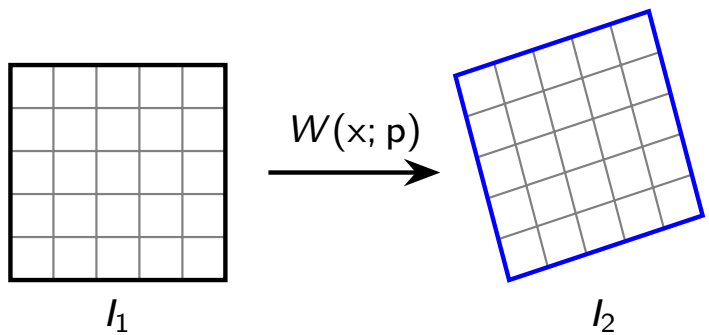


- e.g. affine warp $W(\mathrm{x};\mathrm{p}) = A\mathrm{x} + \mathrm{t}$: 6 unknowns, every pixel contributes $\rightarrow$ accurate, robust to noise

Image registration : one global transform

If all the pixels follow one model (affine, homography... : see *The core of the problem*), estimate its parameters p directly from the intensities :

$$\hat{\mathrm{p}} = \arg\min_{\mathrm{p}} \sum_{\mathrm{x}} \left[I_2(W(\mathrm{x};\mathrm{p})) - I_1(\mathrm{x}) \right]^2$$

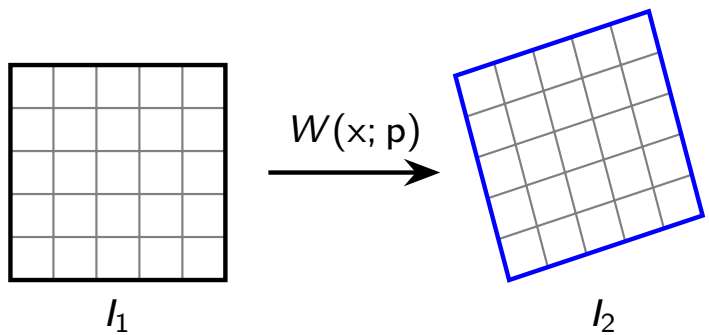


- e.g. affine warp $W(\mathrm{x};\mathrm{p}) = A\mathrm{x} + \mathrm{t}$: 6 unknowns, every pixel contributes $\rightarrow$ accurate, robust to noise
- solved iteratively (Taylor development, as in Lucas–Kanade), coarse to fine

Image registration : one global transform

If all the pixels follow one model (affine, homography... : see *The core of the problem*), estimate its parameters p directly from the intensities :

$$\hat{\mathrm{p}} = \arg\min_{\mathrm{p}} \sum_{\mathrm{x}} \left[I_2(W(\mathrm{x};\mathrm{p})) - I_1(\mathrm{x}) \right]^2$$

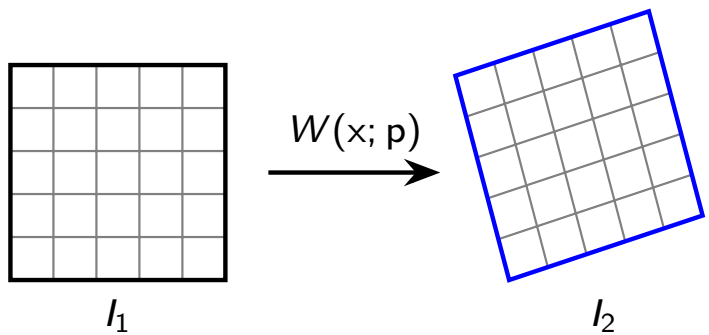


- e.g. affine warp $W(\mathrm{x};\mathrm{p}) = A\mathrm{x} + \mathrm{t}$: 6 unknowns, every pixel contributes $\rightarrow$ accurate, robust to noise
- solved iteratively (Taylor development, as in Lucas–Kanade), coarse to fine
- other scores : NCC (illumination changes), mutual information (different modalities, e.g. MRI vs CT)

Image registration : one global transform

If all the pixels follow one model (affine, homography... : see *The core of the problem*), estimate its parameters p directly from the intensities :

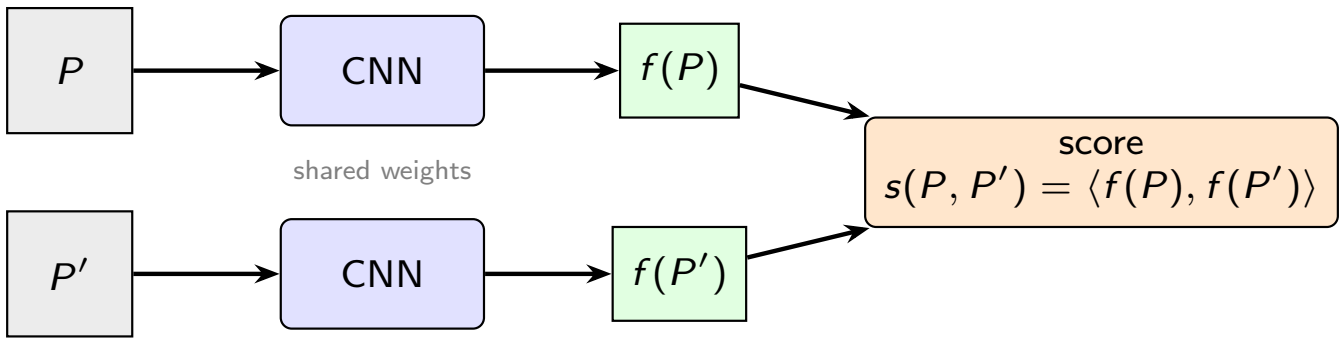
$$\hat{\mathrm{p}} = \arg\min_{\mathrm{p}} \sum_{\mathrm{x}} \left[I_2(W(\mathrm{x};\mathrm{p})) - I_1(\mathrm{x}) \right]^2$$



- e.g. affine warp $W(\mathrm{x};\mathrm{p}) = A\mathrm{x} + \mathrm{t}$: 6 unknowns, every pixel contributes $\rightarrow$ accurate, robust to noise
- solved iteratively (Taylor development, as in Lucas–Kanade), coarse to fine
- other scores : NCC (illumination changes), mutual information (different modalities, e.g. MRI vs CT)
- a *direct* alternative to : corners $\rightarrow$ matching $\rightarrow$ robust estimation

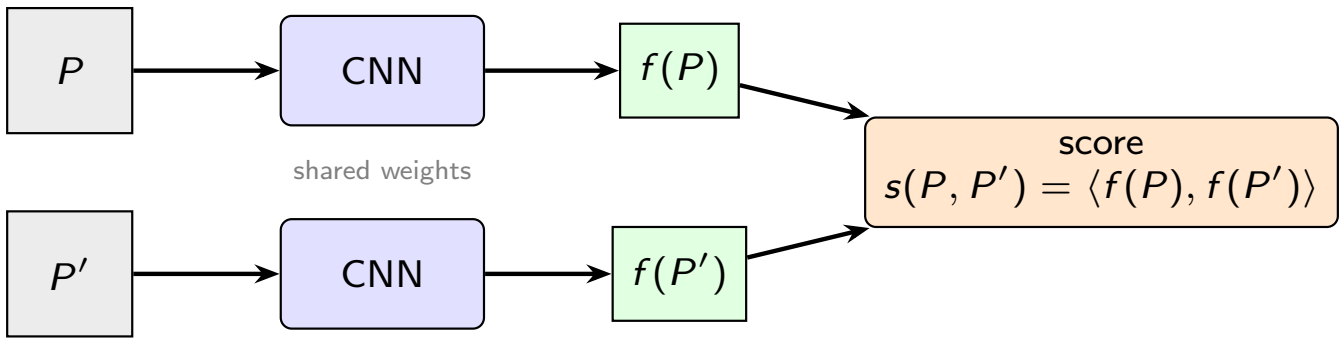
Learning to match (1) : a learned matching score

Keep block matching, but replace SSD/NCC by a score learned from examples (e.g. MC-CNN for stereo, 2015)



Learning to match (1) : a learned matching score

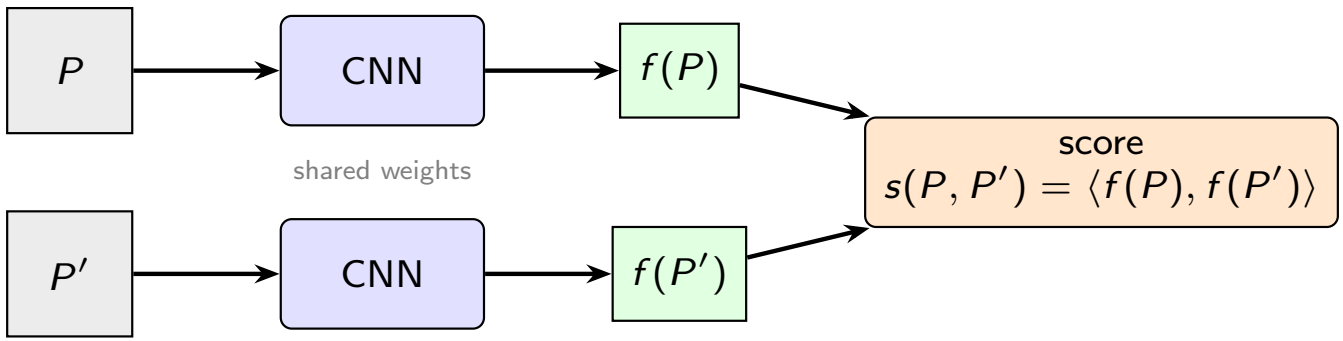
Keep block matching, but replace SSD/NCC by a score learned from examples (e.g. MC-CNN for stereo, 2015)



- input : two small patches P and P' (e.g. 11×11), one from each image

Learning to match (1) : a learned matching score

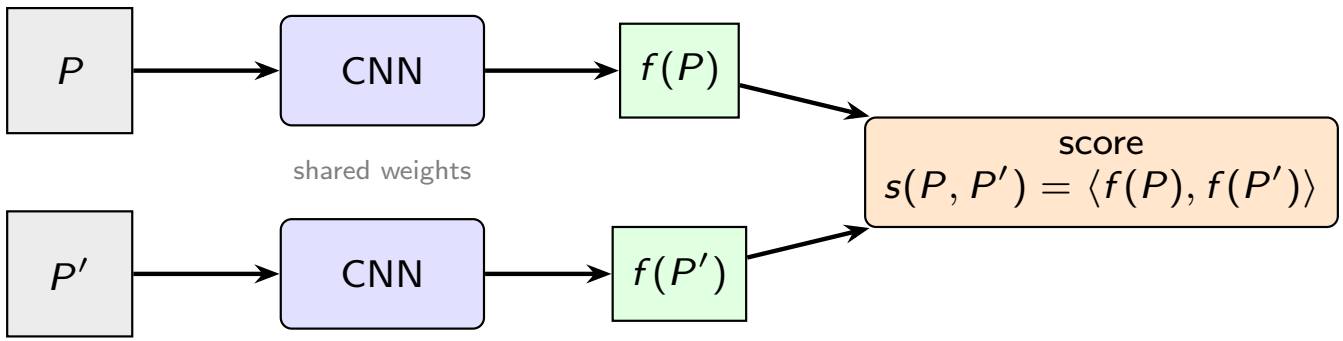
Keep block matching, but replace SSD/NCC by a score learned from examples (e.g. MC-CNN for stereo, 2015)



- input : two small patches P and P' (e.g. 11×11), one from each image
- output : a similarity score $s(P, P')$, used in block matching in place of $-$SSD

Learning to match (1) : a learned matching score

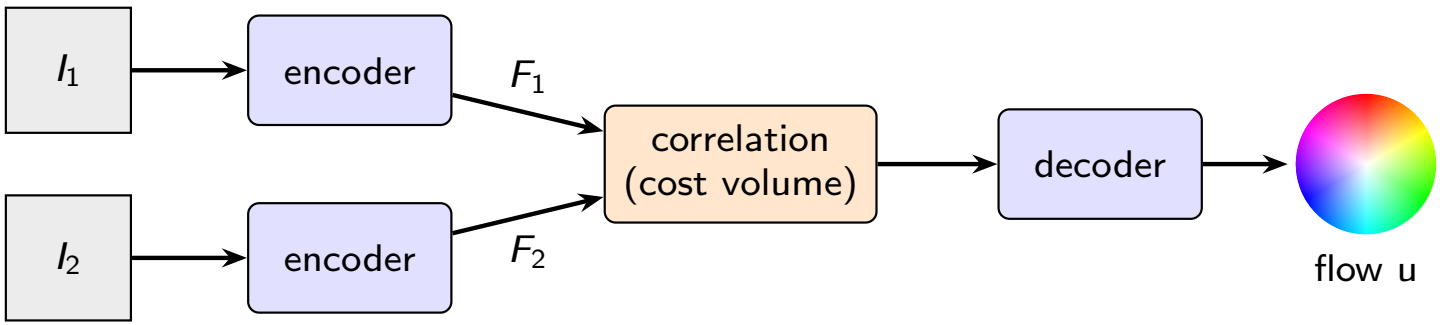
Keep block matching, but replace SSD/NCC by a score learned from examples (e.g. MC-CNN for stereo, 2015)



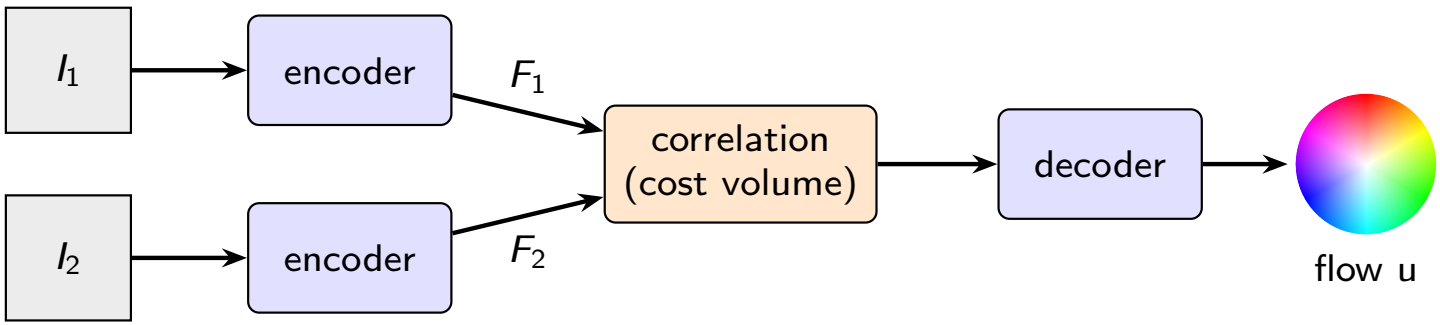
- input : two small patches P and P' (e.g. 11×11), one from each image
- output : a similarity score $s(P, P')$, used in block matching in place of $-$SSD
- loss : the correct match P^+ must beat a wrong one P^- (both known from ground truth disparities) by a margin m :

$$\mathcal{L} = \max\left(0,\ m + s(P, P^-) - s(P, P^+)\right)$$

Learning to match (2) : end-to-end dense flow

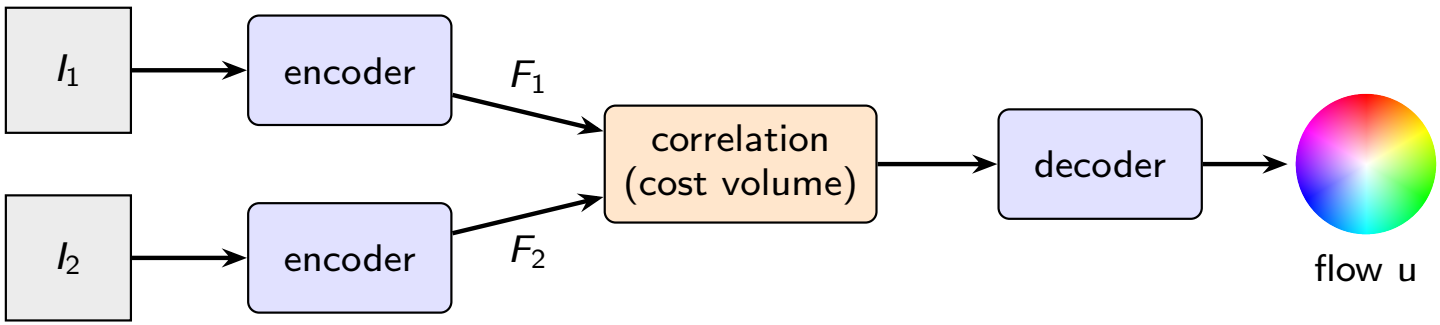


Learning to match (2) : end-to-end dense flow



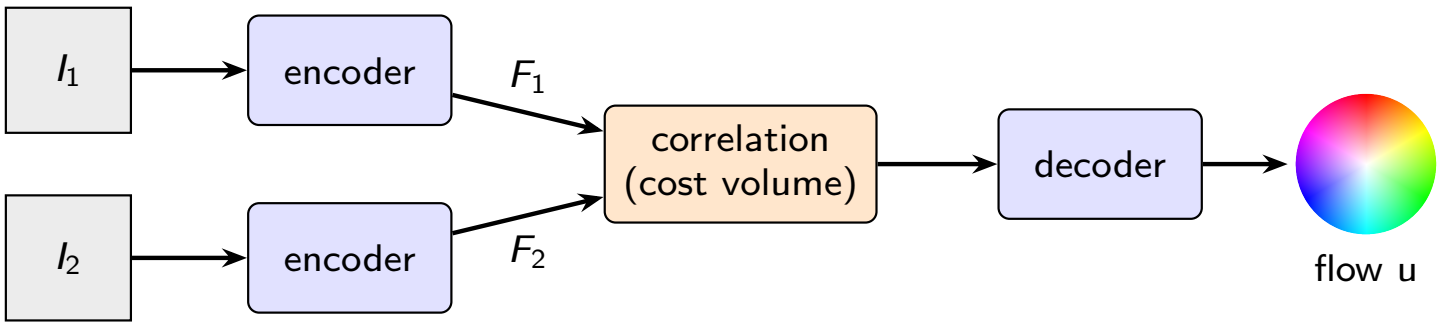
- input : the two images ($H \times W \times 3$ each)

Learning to match (2) : end-to-end dense flow



- input : the two images ($H \times W \times 3$ each)
- output : the dense flow, $H \times W \times 2$ (displayed with a color code : hue = direction, saturation = magnitude)

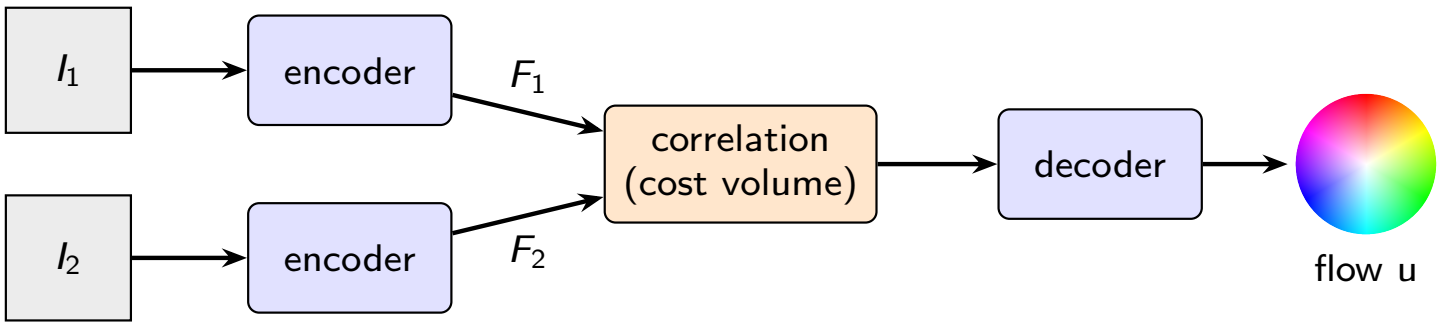
Learning to match (2) : end-to-end dense flow



- input : the two images ($H \times W \times 3$ each)
- output : the dense flow, $H \times W \times 2$ (displayed with a color code : hue = direction, saturation = magnitude)
- loss : average endpoint error (EPE) with respect to the ground truth flow

$$\mathcal{L}_{EPE} = \frac{1}{HW} \sum_{x} \left\| \mathrm{u}_{pred}(x) - \mathrm{u}_{gt}(x) \right\|_2$$

Learning to match (2) : end-to-end dense flow

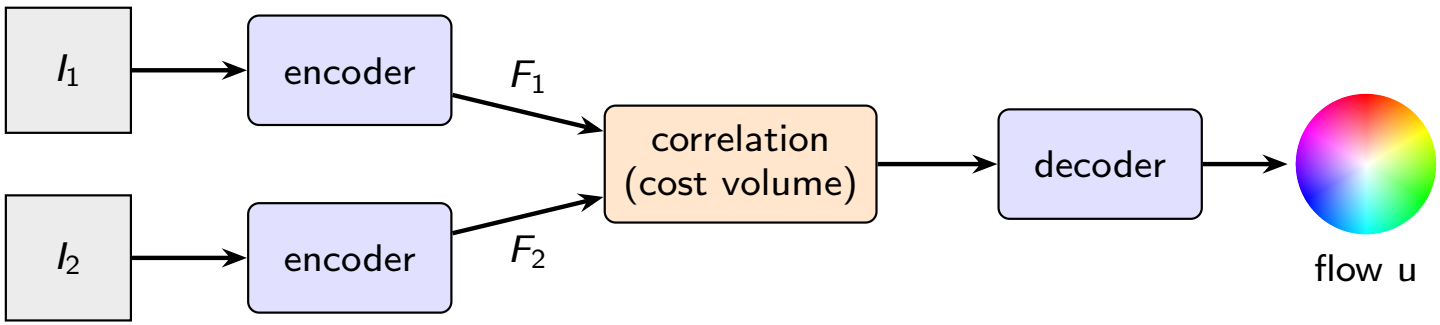


- input : the two images ($H \times W \times 3$ each)
- output : the dense flow, $H \times W \times 2$ (displayed with a color code : hue = direction, saturation = magnitude)
- loss : average endpoint error (EPE) with respect to the ground truth flow

$$\mathcal{L}_{EPE} = \frac{1}{HW} \sum_{\mathrm{x}} \left\| \mathrm{u}_{pred}(\mathrm{x}) - \mathrm{u}_{gt}(\mathrm{x}) \right\|_2$$

- correlation layer $C(\mathrm{x}, \mathrm{d}) = \langle F_1(\mathrm{x}), F_2(\mathrm{x} + \mathrm{d}) \rangle$: block matching on learned features !

Learning to match (2) : end-to-end dense flow

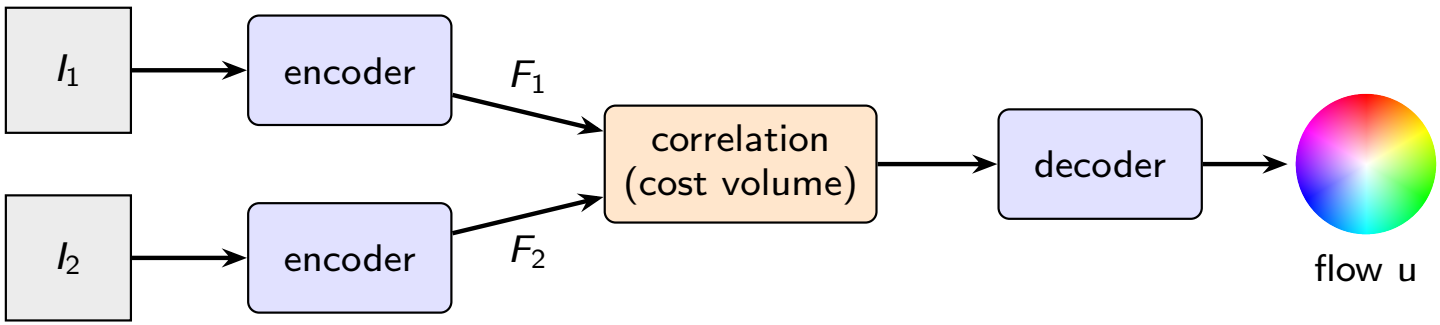


- input : the two images ($H \times W \times 3$ each)
- output : the dense flow, $H \times W \times 2$ (displayed with a color code : hue = direction, saturation = magnitude)
- loss : average endpoint error (EPE) with respect to the ground truth flow

$$\mathcal{L}_{EPE} = \frac{1}{HW} \sum_{x} \left\| u_{pred}(x) - u_{gt}(x) \right\|_2$$

- correlation layer $C(x, d) = \langle F_1(x), F_2(x + d) \rangle$: block matching on learned features !
- real ground truth flow is almost impossible to measure $\rightarrow$ synthetic training scenes (Flying Chairs, Sintel)

Learning to match (2) : end-to-end dense flow



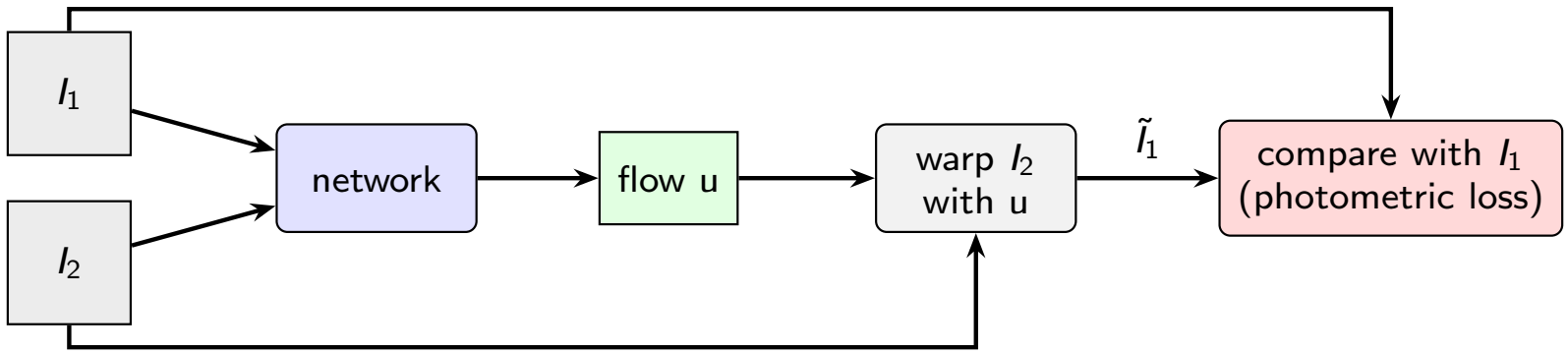
- input : the two images ($H \times W \times 3$ each)
- output : the dense flow, $H \times W \times 2$ (displayed with a color code : hue = direction, saturation = magnitude)
- loss : average endpoint error (EPE) with respect to the ground truth flow

$$\mathcal{L}_{EPE} = \frac{1}{HW} \sum_{x} \left\| u_{pred}(x) - u_{gt}(x) \right\|_2$$

- correlation layer $C(x, d) = \langle F_1(x), F_2(x + d) \rangle$: block matching on learned features !
- real ground truth flow is almost impossible to measure $\rightarrow$ synthetic training scenes (Flying Chairs, Sintel)
- FlowNet (2015), PWC-Net (2018, coarse to fine with warping), RAFT (2020)

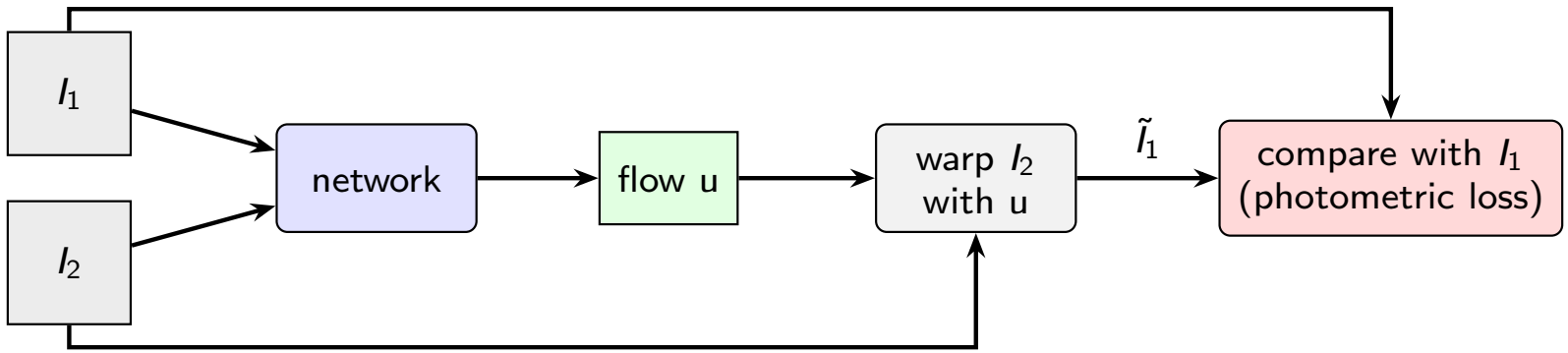
Learning to match (3) : no ground truth needed

No ground truth flow for real videos ? Use the classical energy as the loss !



Learning to match (3) : no ground truth needed

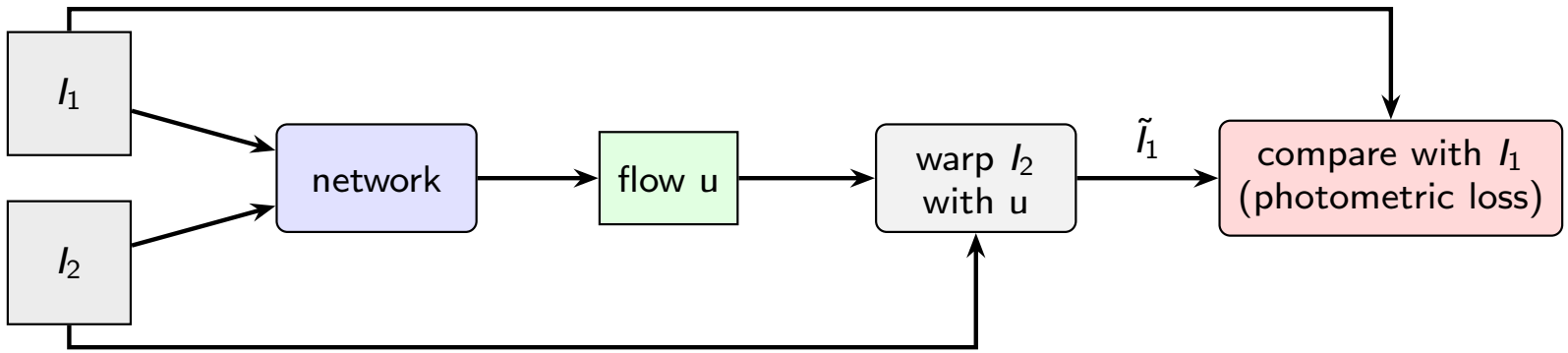
No ground truth flow for real videos ? Use the classical energy as the loss !



- input : two images output : flow u (same kind of network)

Learning to match (3) : no ground truth needed

No ground truth flow for real videos ? Use the classical energy as the loss !

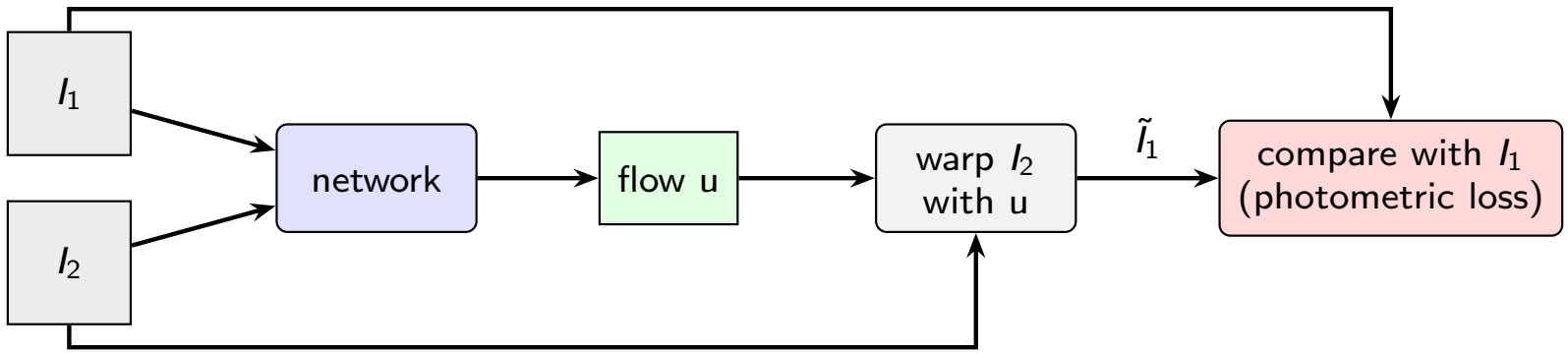


- input : two images output : flow u (same kind of network)
- loss : photometric term + smoothness term, i.e. Horn–Schunck used as a loss :

$$\mathcal{L} = \sum_x \rho\big(I_2(x + u(x)) - I_1(x)\big) + \lambda \sum_x \|\nabla u(x)\|$$

Learning to match (3) : no ground truth needed

No ground truth flow for real videos ? Use the classical energy as the loss !



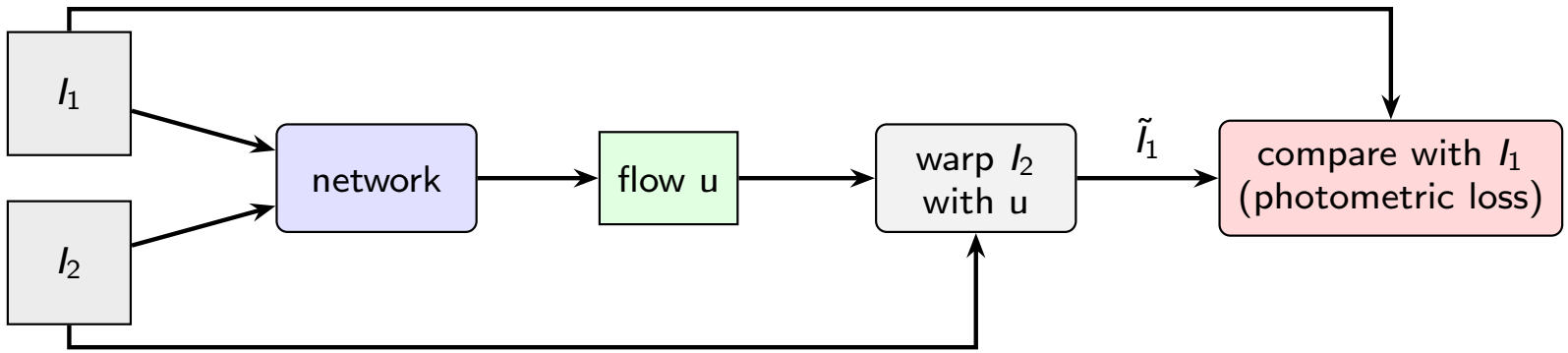
- input : two images output : flow u (same kind of network)
- loss : photometric term + smoothness term, i.e. Horn–Schunck used as a loss :

$$\mathcal{L} = \sum_x \rho\big(I_2(x + u(x)) - I_1(x)\big) + \lambda \sum_x \|\nabla u(x)\|$$

- warping (bilinear interpolation) is differentiable $\rightarrow$ backpropagation works

Learning to match (3) : no ground truth needed

No ground truth flow for real videos ? Use the classical energy as the loss !



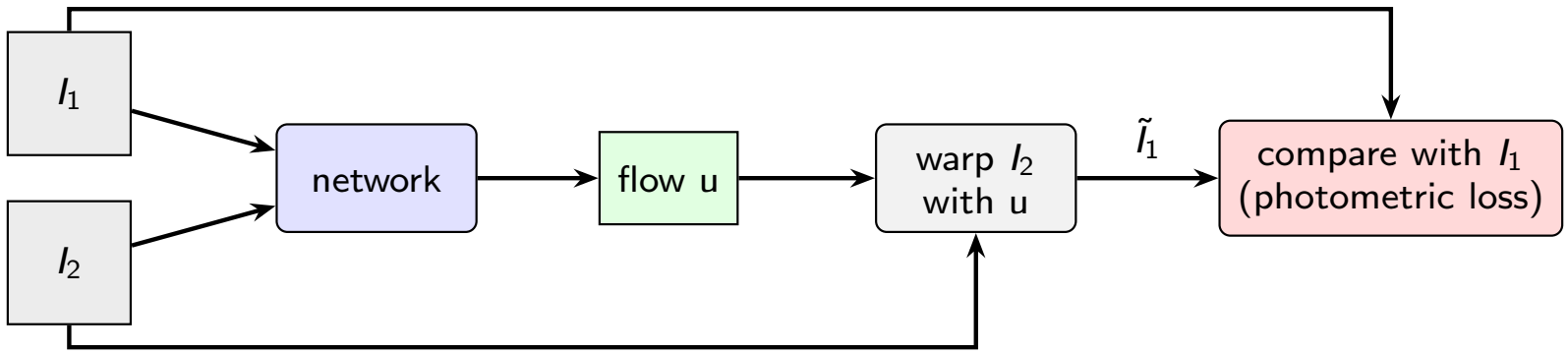
- input : two images output : flow u (same kind of network)
- loss : photometric term + smoothness term, i.e. Horn–Schunck used as a loss :

$$\mathcal{L} = \sum_{x} \rho\big(I_2(x + u(x)) - I_1(x)\big) + \lambda \sum_{x} \|\nabla u(x)\|$$

- warping (bilinear interpolation) is differentiable $\rightarrow$ backpropagation works
- ρ : robust penalty, e.g. $|\cdot|$; occluded pixels have no match and are masked out

Learning to match (3) : no ground truth needed

No ground truth flow for real videos ? Use the classical energy as the loss !



- input : two images output : flow u (same kind of network)
- loss : photometric term + smoothness term, i.e. Horn–Schunck used as a loss :

$$\mathcal{L} = \sum_{x} \rho\big(I_2(x + u(x)) - I_1(x)\big) + \lambda \sum_{x} \|\nabla u(x)\|$$

- warping (bilinear interpolation) is differentiable $\rightarrow$ backpropagation works
- ρ : robust penalty, e.g. $|\cdot|$; occluded pixels have no match and are masked out
- same idea for depth (Monodepth, 2017) : predict the disparity from one image, warp the other view of a stereo pair, compare

Dense matching - conclusion

	sparse (corners)	dense, classical	dense, learned
output	a few hundred matches	one vector per pixel	one vector per pixel
no texture	no corners there	filled in by the regularization	filled in by learned priors
large motion	fine (descriptors)	coarse to fine	fine
cost	low	high	high (GPU) + training data
typical use	pose, SLAM, panoramas	flow, stereo, registration	state of the art flow and stereo

In practice (OpenCV)

- `calcOpticalFlowPyrLK` (KLT), `calcOpticalFlowFarneback` (dense flow)
- `StereoBM`, `StereoSGBM` (block matching for stereo)
- `findTransformECC` (direct registration)

Dense matching - conclusion

	sparse (corners)	dense, classical	dense, learned
output	a few hundred matches	one vector per pixel	one vector per pixel
no texture	no corners there	filled in by the regularization	filled in by learned priors
large motion	fine (descriptors)	coarse to fine	fine
cost	low	high	high (GPU) + training data
typical use	pose, SLAM, panoramas	flow, stereo, registration	state of the art flow and stereo

In practice (OpenCV)

- `calcOpticalFlowPyrLK` (KLT), `calcOpticalFlowFarneback` (dense flow)
- `StereoBM`, `StereoSGBM` (block matching for stereo)
- `findTransformECC` (direct registration)

The sparse/dense border is fading : detector-free matchers (e.g. LoFTR, 2021) compute dense learned features, then keep the most confident matches.