Introduction to Computer Vision - P2
Lesson 2 - Introduction to Computer Vision

Alasdair Newson

alasdair.newson@telecom-paris.fr

November 25, 2021

Summary

Introduction

- Today, introduction to **computer vision**. What is computer vision ?
- Problems, methods and algorithms which pertain to automatic (computer), understanding of high-level characteristics of images and videos
  - Contours, corners and other features
  - Analysing motion
  - Identifying objects
- Goal : the computer can understand and analyse its surroundings

Summary

Segmentation

- **Segmentation** is one of the most common processing or pre-processing steps in computer vision
- Important for semantic understanding of a scene
  - Robotics : where am I, what is in front of me, what obstacles are there ?
  - Medical imaging : different parts of the body
- There are many approaches to segmentation :
  1. Thesholding
  2. Region growing
  3. Active contours (differential geometry)
  4. Motion-based
- Here we look at **thresholding, region growing and active contours** based methods

Segmentation

- Simplest approach : **thresholding**
  - Commonly known, simple, algorithm, Otsu's method*

Otsu's segmentation algorithm

- Let I represent an image. Otsu's algorithm searches for the threshold τ such that the intra-class variance is minimised :

$$\arg\min_{\tau} \omega_0(\tau)\sigma_0(\tau) + \omega_1(\tau)\sigma_1(\tau) \tag{1}$$

- $\sigma_0(t)$ and $\sigma_1(t)$: variances of the pixels in the first and second regions
- $\omega_0(t)$ and $\omega_1(t)$ are weights of the first and second regions

$$\omega_0(\tau) = \sum_{p\in\Omega} \mathbb{1}_\tau(I(p)), \qquad \omega_1(\tau) = \sum_{p\in\Omega} 1 - \mathbb{1}_\tau(I(p))$$

- Where $\mathbb{1}_\tau(I(p)) = \begin{cases} 1 \text{ if } I(p) <= \tau \\ 0 \text{ otherwise} \end{cases}$

* *A threshold selection method from gray-level histograms, Nobuyuki Otsu, 1979, IEEE Trans. Sys. Man. Cyber*

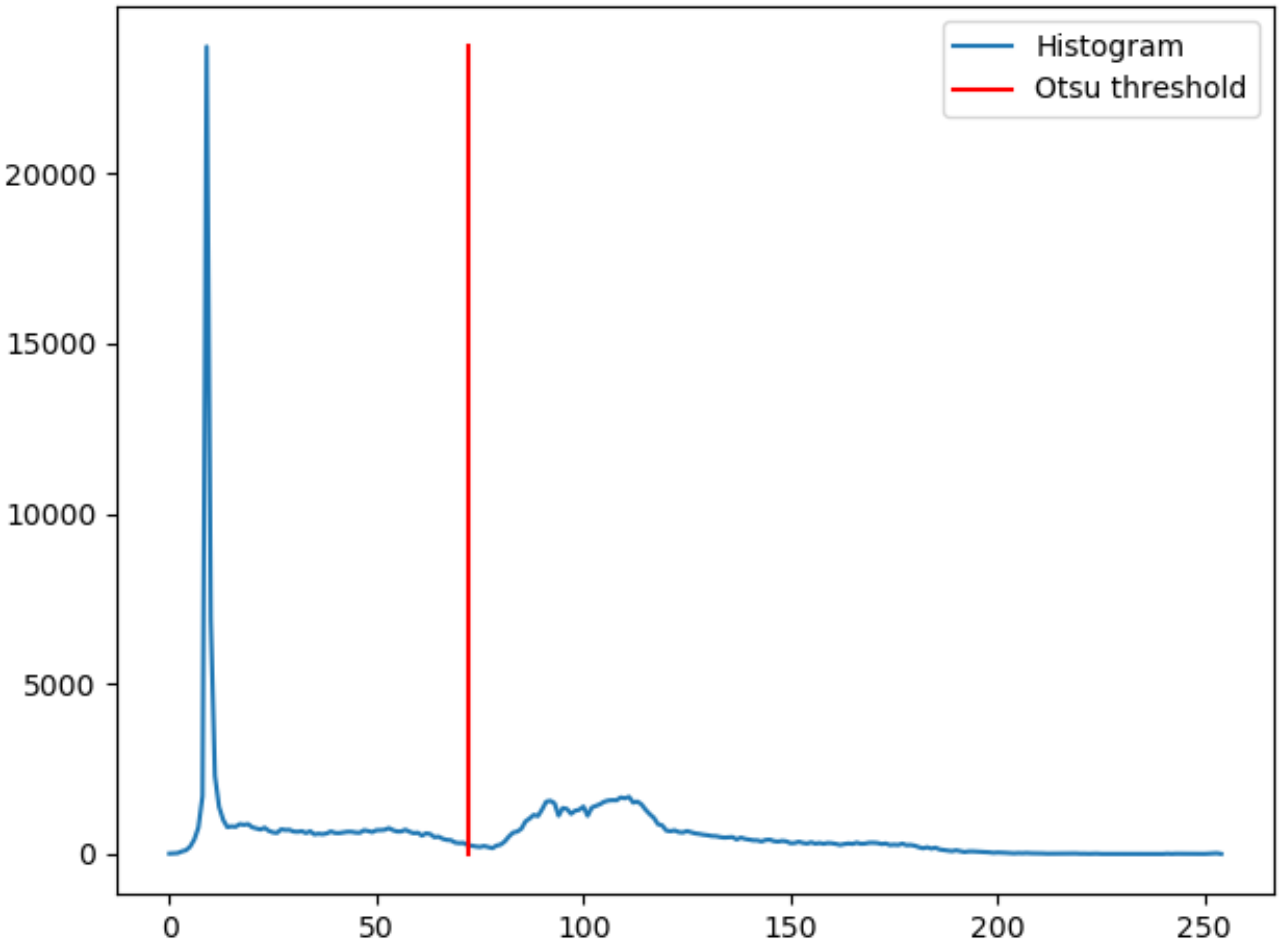
Segmentation

- Example of Otsu segmentation

Input image

Segmentation

- Example of Otsu segmentation



Otsu histogram threshold

Segmentation

- Example of Otsu segmentation

Otsu segmentation

Segmentation

- Some disadvantages of Otsu's method :
  - Global threshold, segmented objects may not be connected
  - Entirely based on pixel values, no notion of geometric coherence
- Another intuitive idea is to use **region growing methods**

Region growing

- We start by initially choosing some **seed points**
  - This can be a user input or random
- The algorithm then attempts to include more and more pixels in a neighbourhood Ψ
- Pixels are included based on a decision relative to the pixel intensity
- Simplest criterion : absolute difference between new pixel value $I(x, y)$ and average of the region, μ

Region growing

Algorithm 1: Region growing algorithm

Result: R (segmentation region R - a set of pixels)
Define patch neighbourhood Ψ
Initialise region R with an initial seed R_0
while R *is still growing* **do**
  Re-calculate μ
  for $(x, y) \in R$ **do**
    for $(i, j) \in \Psi_{x,y}$ *such that* $(i, j) \notin R$ *(only test new pixels)* **do**
      if $|I(i, j) - \mu| < \tau$ **then**
        Add (i, j) to R
      end
    end
  end
end

Segmentation

Initial seed

Segmentation

Segmentation

Segmentation

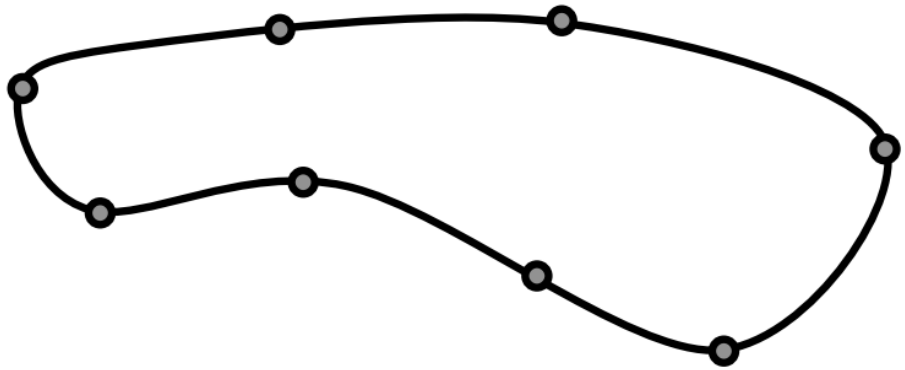
Segmentation

Segmentation

- Advantage of region growing : notion of geometry/topology, region must be connected
- Drawbacks of region growing :
  - Not robust to outliers (impulse noise for example)
  - No notion of smoothness in the segment boundary
- For this, we turn to geometry-based approaches. The best known is **active contours** (also known as the snakes model)

Segmentation

- Active contours* is used to segment specific objects
  - Start by choosing an initial segmentation
  - Iteratively refine the segmentation by minimising an energy
- The segment is determined by a closed **curve** v, parameterised by a set of points $\mathbf{a}_i$, $i = 0, \ldots, N-1$
  - Everything inside the line is one segment, everything outside is the other segment



* *M. Kass, A. Witkin and D. Terzopoulos,* ***Snakes : Active Contours Models****, 1988, IJCV*

Active contours model

$$E(v) = \int_0^1 E_{int}(v(s)) + E_{image}(v(s))ds \tag{2}$$

- E_{int} is the **internal energy**, requiring that the curve be as smooth as possible
- E_{image} is the **image energy**, requiring that the curve contract around edges

- Note, in the original approach, there is also an external energy E_{con}, which we do not go into here (used for manually guiding the snake)

Segmentation

- E_{int} measures the bending and curving of the line (we wish to have a smooth segmentation)

$$E_{int}(v(s)) = \frac{1}{2}\left(\alpha(s)|v'(s)|^2\right) + \frac{1}{2}\left(\alpha(s)|v''(s)|^2\right) \tag{3}$$

- Where v' and v'' correspond to $\frac{dv}{ds}$ and $\frac{d^2v}{ds^2}$ respectively
- $\alpha(s)$ and $\beta(s)$ are user-defined weights

- E_{image} makes sure that we stop at edges, and controls whether $v(s)$ is attracted to dark or bright contours

$$E_{image}(v(s)) = w_{line}I(v(s)) - w_{edge}\|\nabla I(v(s))\|_2^2 \tag{4}$$

- w_{line} and w_{edge} are paramters (w_{line} controls dark/light contours)

Segmentation

- This approach uses tools from differential geometry and functional analysis
- Quite a technical subject, so we do not go into great detail
- The main question : how do we minimise the energy in Equation (2) ?
- This requires the **calculus of variations**
  - Also needed in optical flow (motion estimation), so we present it now

Variational methods

- Suppose we wish to find a continuous function f which minimises some **functional** (an energy) J
- A functional is a mapping from a function space (the images) to $\mathbb{R}$
  - Basically, evaluates how "good" a function is

$$\hat{f} = \arg\min_{f} J(t, f, f').$$

- Functionals are usually integrals over the function domain. In our case, this will be an integral over the image domain Ω

$$J(f) = \int_{\Omega} E(t, f, f')dt$$

- E is a function which depends on the problem. It is usually a function which determines how "good" the solution f is

Variational methods

- Variational methods aim to minimise energies such as the previous one. They employ the theory of **calculus of variations**
- This is quite a technical subject, so we only present a brief overview here

Variational methods

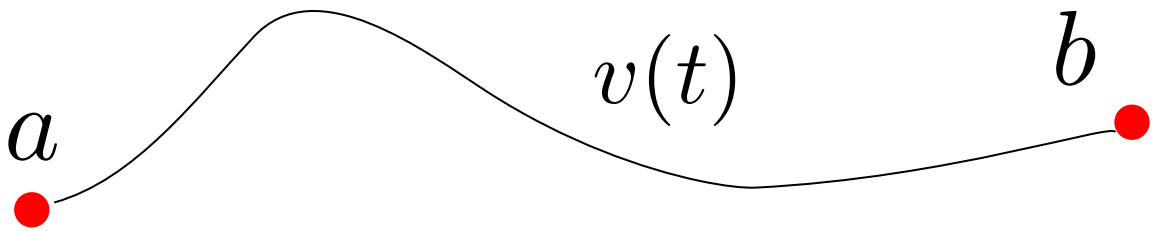
- Let us take a simple case where f is a function of one variable x
- Main idea is to add a small perturbation $\varepsilon\eta$, where η is a smooth function, and differentiate wrt ε : $\frac{dJ(f+\varepsilon\eta)}{d\varepsilon}$
  - Very similar to normal calculus, except we wish to find a minimising **function**, not a minimising number or vector
- Minimising in this manner leads to solving the **Euler-Lagrange equations**

$$E_f(t, f(t), f'(t)) - \frac{d}{dt} E_{f'}(t, f(t), f'(t)) = 0, \tag{5}$$

- Where E_f and $E_{f'}$ are the derivatives of E wrt the second and third arguments, respectively
- This can be generalised to higher derivatives and to more than one variable

Variational methods

- Example of energy functional : path length of a curve v between two points a and b



$$J(v) = \int_a^b \sqrt{1 + v'(t)^2} dt$$

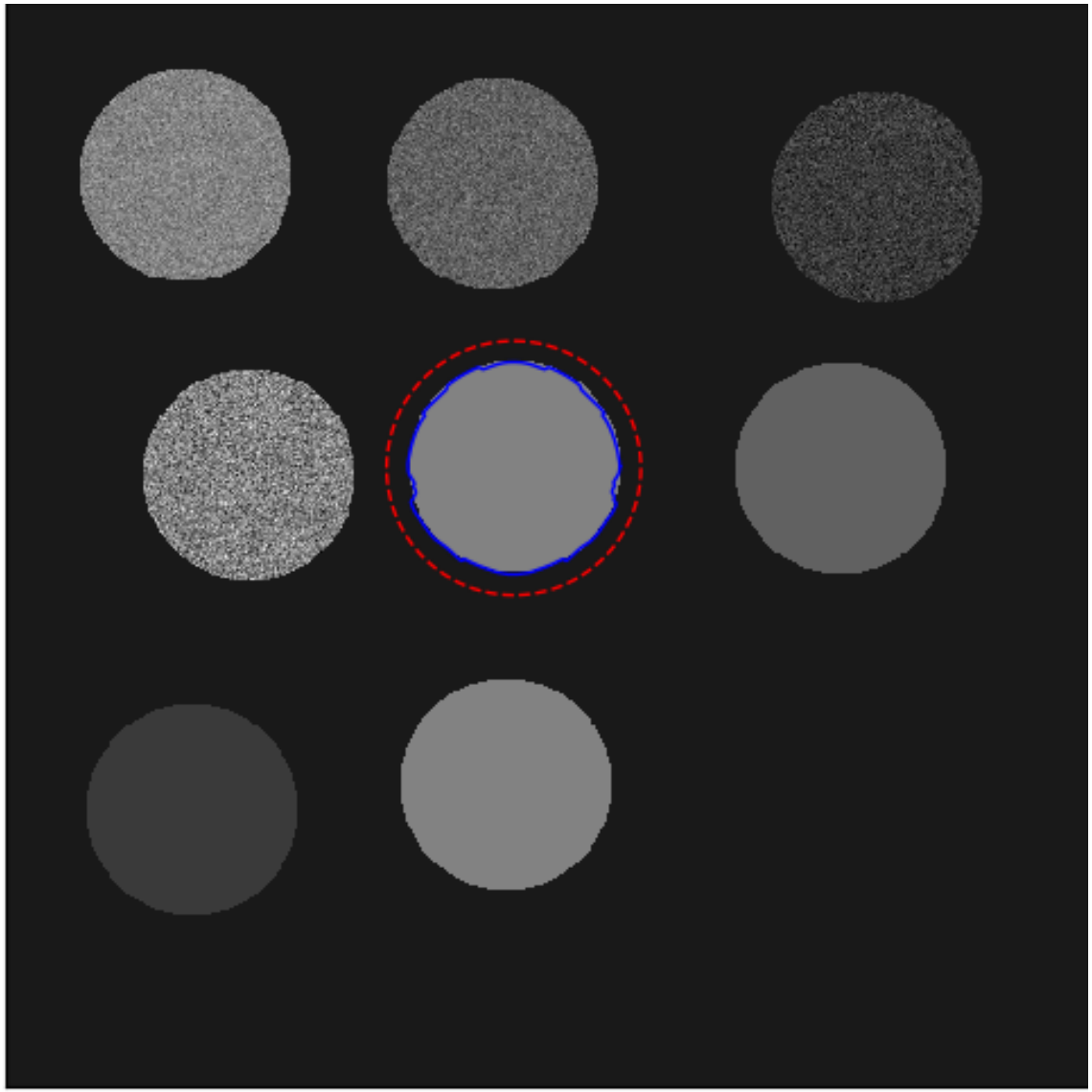
- The solution to the Euler-Lagrange equation of this functional is a straight line $v(t) = \mathbf{c}t + \mathbf{d}$

Segmentation

- Back to the active contours model
- In the original approach*, the authors either discretise the original functional or discretise the Euler-Lagrange eqaution
- An implicit numerical scheme is proposed for solving the resulting equations
  - These are quite involved, so we do not go into them here. Basically : numerical scheme for solving partial differential equations
- Let us look at some results

* *M. Kass, A. Witkin and D. Terzopoulos,* ***Snakes : Active Contours Models****, 1988, IJCV*

Segmentation



Segmentation

Segmentation - summary

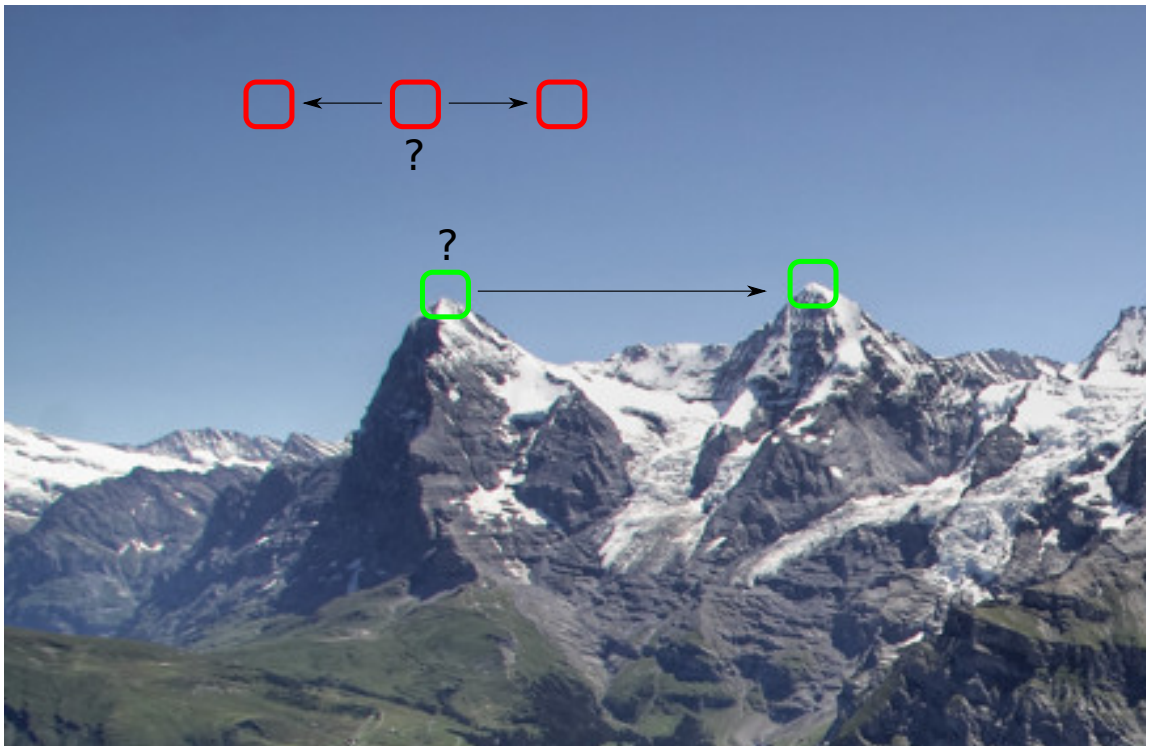
- There are quite a few variants on the idea of active contours (Chan and Vese*,
  - Advantages : rigorous formulation, minimisation of functional, geometrically meaningful
  - Disadvantages : quite technical to understand/implement, parameters of algorithm can be difficult to choose
- In general active contour-type algorithms are used when a specific object needs to be detected (eg medical imaging)
- More recently deep learning has become the state of the art (as in many domains)

* *T. F. Chan and L. A. Vese,* ***Active Contours Without Edges****, 2001, IEEE TIP* † *V. Caselles, R. Kimmel and G. Sapiro,* ***Geodesic Active Contours****, 1997, IJCV*

Summary

Feature detection

- Often, we wish to detect **features** in images. What are features ?
  - Lines, corners, and other points of interest
- For tasks such as tracking, we need unambiguous, identifiable points
- Textureless points are very difficult to follow in an image

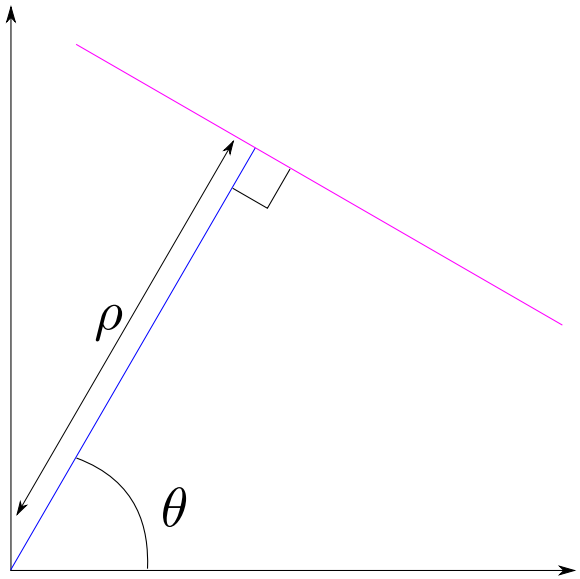


Line detection

- **Lines and segments** often need to be detected in images
  - Vanishing point detection, structure, object detection
- How do we find lines ? Two steps :
  - Edge detection (sobel filters or otherwise)
  - A way to group strong collections of edges : **Hough transform**

Line detection

- The Hough transform is an operation on digital images whose goal is to **detect parametric shapes** (lines, circles, ellipses)
- Commonly used for detecting lines, but can be used for circles etc
  - In this lesson, we detect lines
- First question : how do we parametrise lines ?
  - One approach : $y = ax + b$. Unfortunately, this does not work for vertical lines (infinite slope)
  - Better to use polar coordinates : $\rho = x\cos(\theta) + y\sin(\theta)$



Line detection

- The Hough transform calculates an **accumulation matrix $\mathbf{A}$**
  - Discretise line parameters :
    $(\rho_i, \theta_j),\ i = 0, \ldots, m_\rho - 1\ ,\ j = 0, \ldots, n_\theta - 1$
  - For each discretised line, count how many non-zero pixels lie along this line

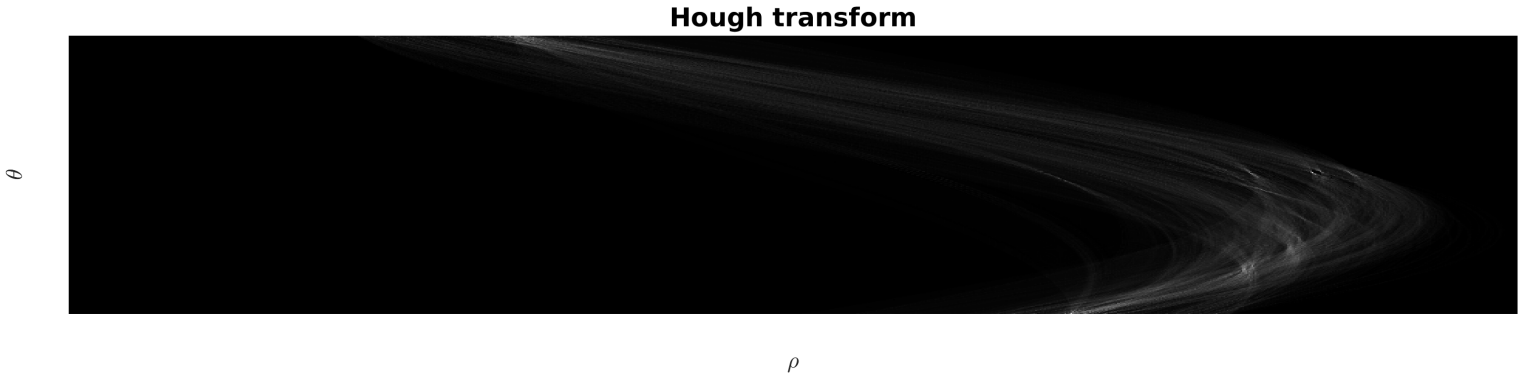
---

Algorithm 2: Hough accumulation matrix

Result: $\mathbf{A}$

$\mathbf{A} = \text{zeros}(m_\rho, n_\theta)$ **for**
$(i, j) \in (0, \ldots, m_\rho - 1) \times (0, \ldots, n_\theta - 1)$ **do**
  for $(x, y) \in (0, \ldots, m - 1) \times (0, \ldots, n - 1)$ **do**
    if $(x, y) \in \mathit{Line}(\rho_i, \theta_j)$ *and* $I(x, y) = 1$ **then**
      $\mathbf{A}_{i,j} = \mathbf{A}_{i,j} + 1$
    end
  end
end

Line detection



Line detection

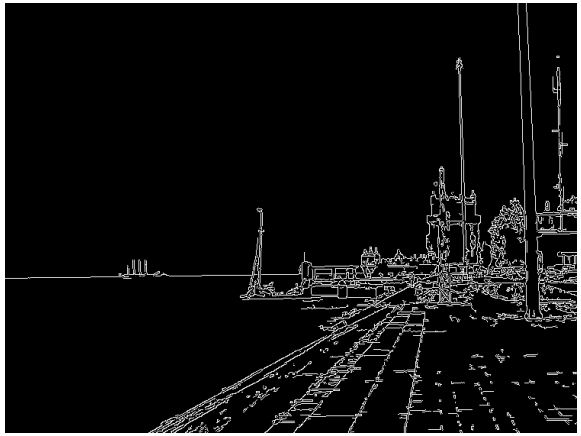
Hough transform

1. Use a pixel-wise edge detector to detect potential lines
2. Calculate the Hough transform of the edge image
3. Threshold Hough transform to find lines

Line detection

Example of Hough transform line detection

Input image



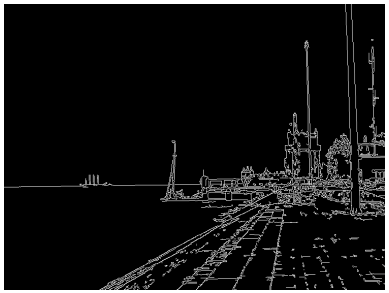
Canny edge detection

Detected lines

Line detection

- Our final task is to group the line points into **segments**
- This is often done using an heuristic algorithm, specifying :
  - A minimum segment length
  - A maximum length of holes in a segment

Input image



Canny edge detection

Detected lines

Detected segments

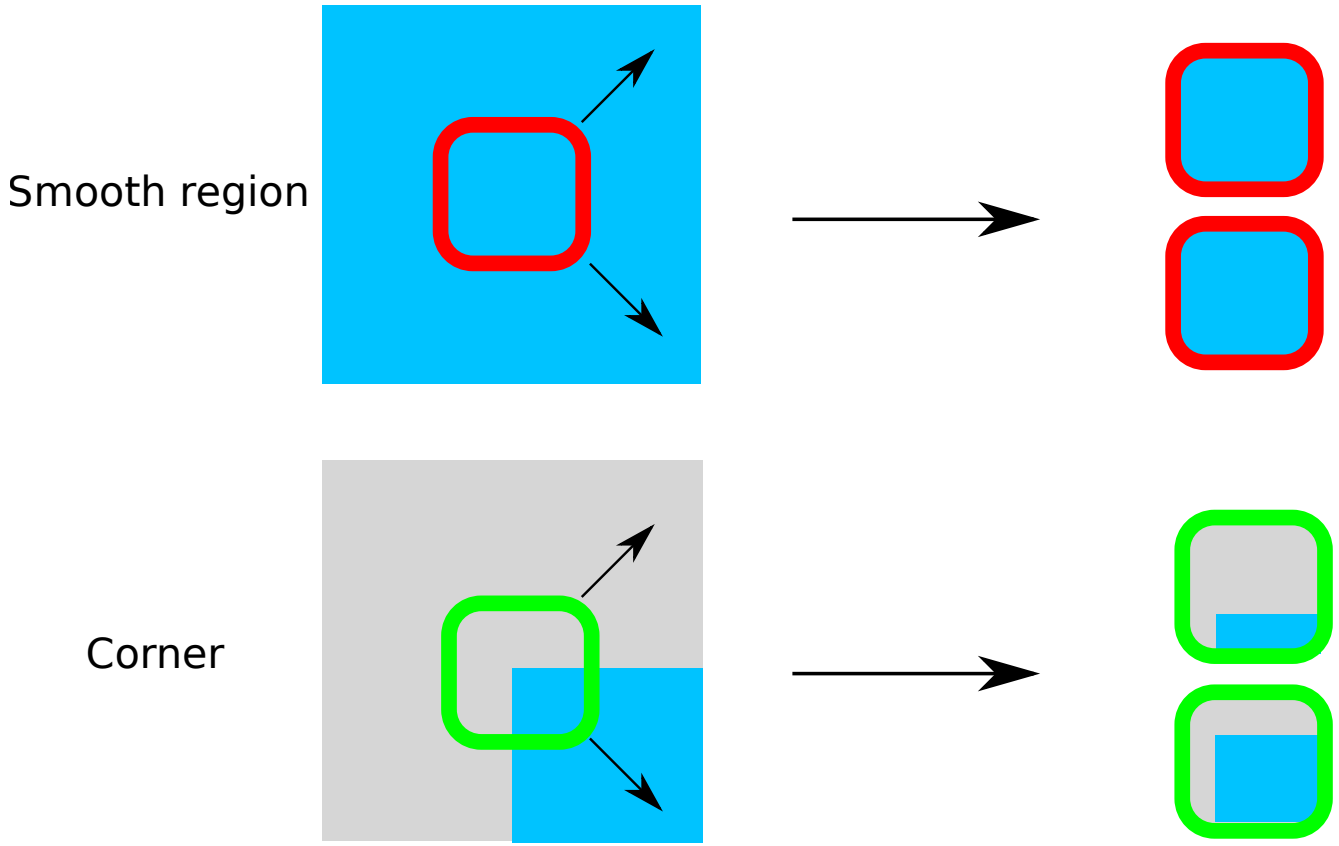
Feature detection - corners

- Next, we look at identifying **corners** in an image
  - Corners are unambiguous
- How do we identify such corners ? One approach : Harris corner detector
- Let us first define a **patch neigbourhood** $\Psi = \{-a, \ldots, a\}^2$, with $a \in \mathbb{N}$
- We define Ψ_p as a patch centred on pixel p : $\Psi_p = \Psi + p$
  - In practice, a patch is a square of pixel positions
  - Here, we use odd size patches : square of odd size $P = a * 2 + 1$, $a \in \mathbb{N}$
- We define the patch distance, between two patches Ψ_p and Ψ_q, as the sum of squared differences of the pixel values

$$\|\Psi_p - \Psi_q\|_2^2 = \sum_{t \in \Psi} (I(p+t) - I(q+t))^2 \tag{6}$$

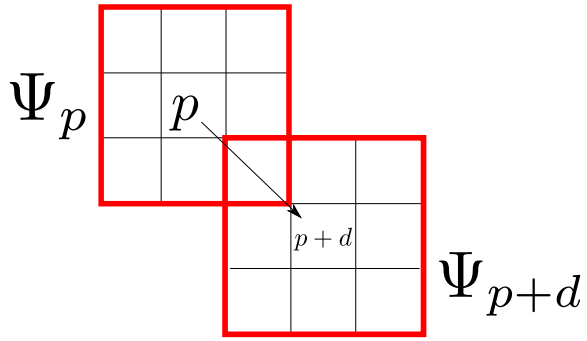
Feature detection - corners

- Main idea of the Harris corner detector : a small shift of the patch in **any direction** should change the patch greatly



Feature detection - corners

- Consider a small shift $d = (d_x, d_y)$ from p



- Using a Taylor series development*, the sum of squared differences of the patches is

$$\|\Psi_p - \Psi_{p+d}\|_2^2 = \sum_{q \in \Psi_p} (I(q_x, q_y) - I(q_x + d_x, q_y + d_x))^2 \quad (7)$$

$$= \sum_{q \in \Psi_p} (I(q_x, q_y) - (I(q_x, q_y) + I_x(q_x, q_y)d_x + I_y(q_x, q_y)d_y))^2 \quad (8)$$

* *Why do we do this ? Because it would be slow to check all small displacements around p*

Feature detection - corners

$$\|\Psi_p - \Psi_{p+d}\|_2^2 = \sum_{q \in \Psi_p} (I_x(q_x, q_y)d_x + I_y(q_x, q_y)d_y))^2 \tag{9}$$

$$= \sum_{q \in \Psi_p} I_x^2(q_x, q_y)d_x^2 + 2I_x(q_x, q_y)I_y(q_x, q_y)d_x d_y + I_y^2(q_x, q_y)d_y^2 \tag{10}$$

Feature detection - corners

- We can re-write this in matrix form, for any displacement d :

$$\|\Psi_{p+d} - \Psi_p\|_2^2 = d^\intercal \; A \; d \tag{11}$$

- Where : $A = \begin{bmatrix} \sum_{q \in \Psi_p} I_x(q)^2 & \sum_{q \in \Psi_p} I_x(q) I_y(q) \\ \sum_{q \in \Psi_p} I_x(q) I_y(q) & \sum_{q \in \Psi_p} I_y(q)^2 \end{bmatrix}$
- The matrix $A \in \mathbb{R}_{2,2}$ represents the auto-correlation of the image close to the point p :
  - How much does the image change in any direction d around p ?
- We can find the directions of greatest and smallest change by taking the eigendecomposition of A

Feature detection - corners

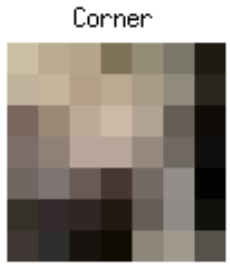
- Let λ_1 and λ_0 be the biggest and smallest eigenvalues of A respectively. We have:

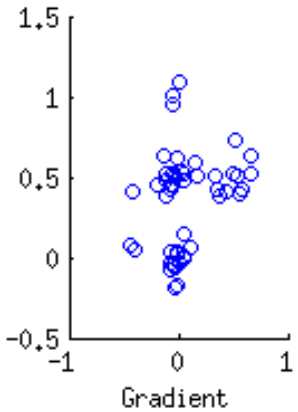
$$\lambda_2 d^T d \leq d^T A d \leq \lambda_1 d^T d \tag{12}$$

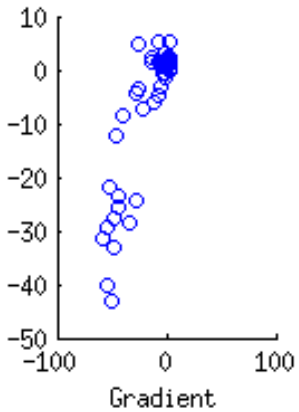
- Therefore, if both λ_1 and λ_2 are high, then **the patch difference is high, whatever the direction** d
  - Note, λ_1 and λ_2 are positive because $d^T A d > 0$ (A is positive definite)

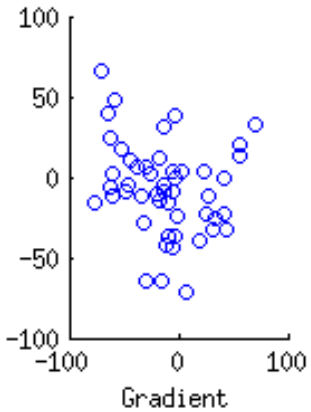
Feature detection - corners

- There are three configurations which correspond to different situations
  1. Two small eigenvalues : flat (constant) region
  2. One small, one large eigenvalue : edge
  3. Two large eigenvalues : corner









Feature detection - corners

- How do we measure this ? One way, proposed by Harris et al, is the following

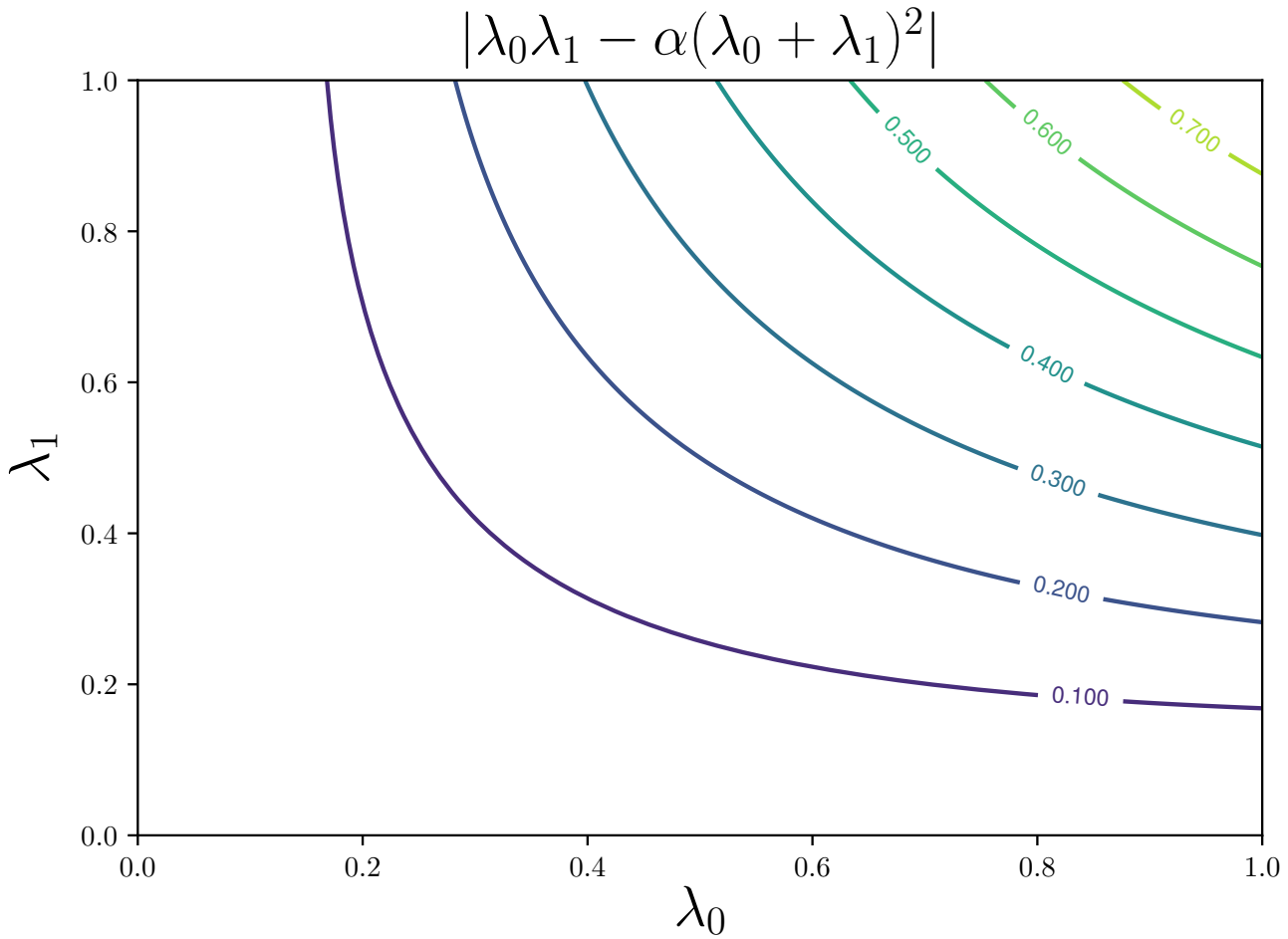
Harris corner detector

- Calculate I_x and I_y, the gradients of the image
- Determine the eigenvalues λ_0, λ_1 of the matrix A
- Detect all points p such that :

$$|\lambda_0\lambda_1 - \alpha(\lambda_0 + \lambda_1)^2| = |\mathsf{det}(A) - \alpha\mathsf{trace}(A)^2| > k \tag{13}$$

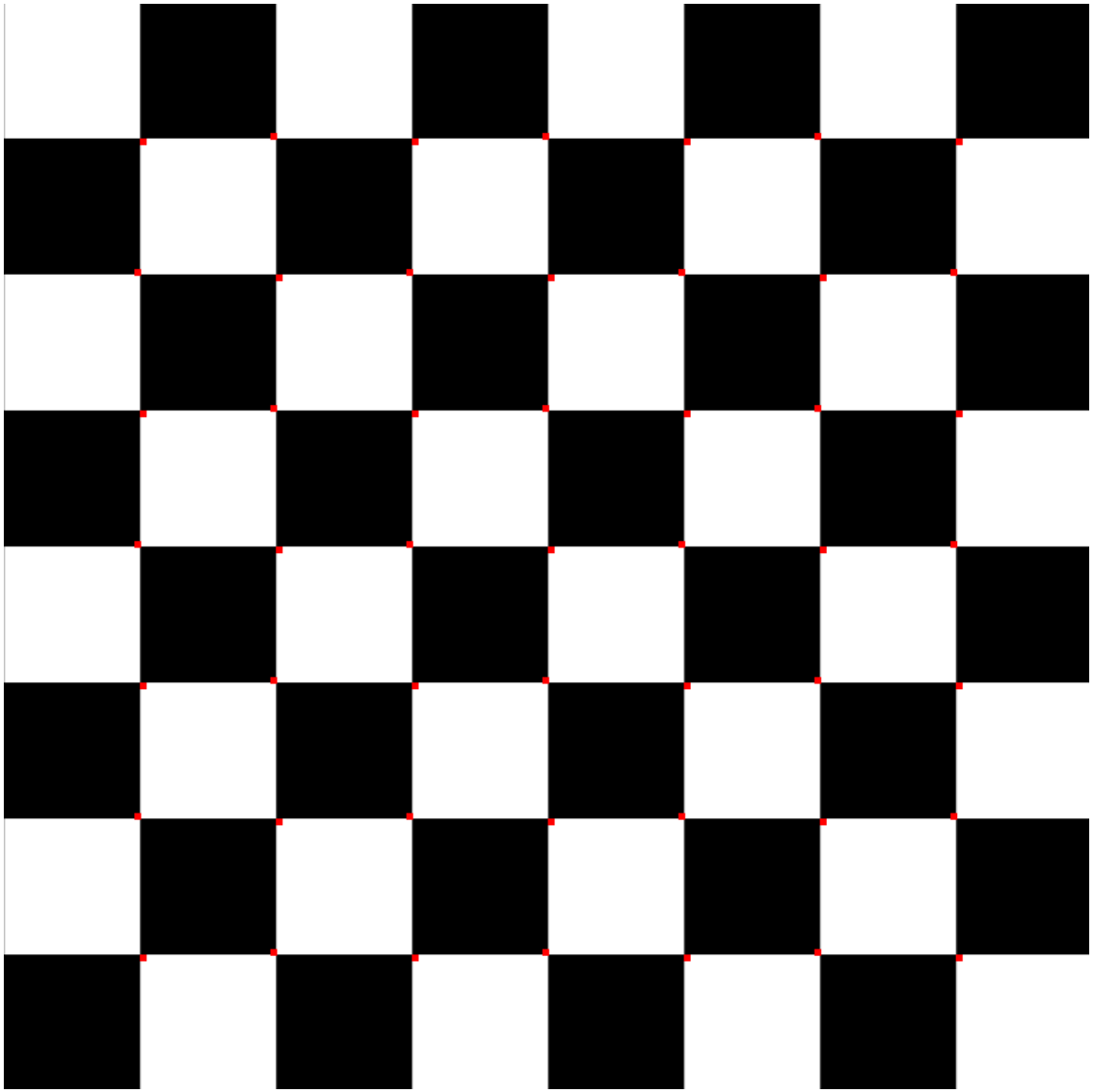
- What does this criterion look like ?

Feature detection - corners

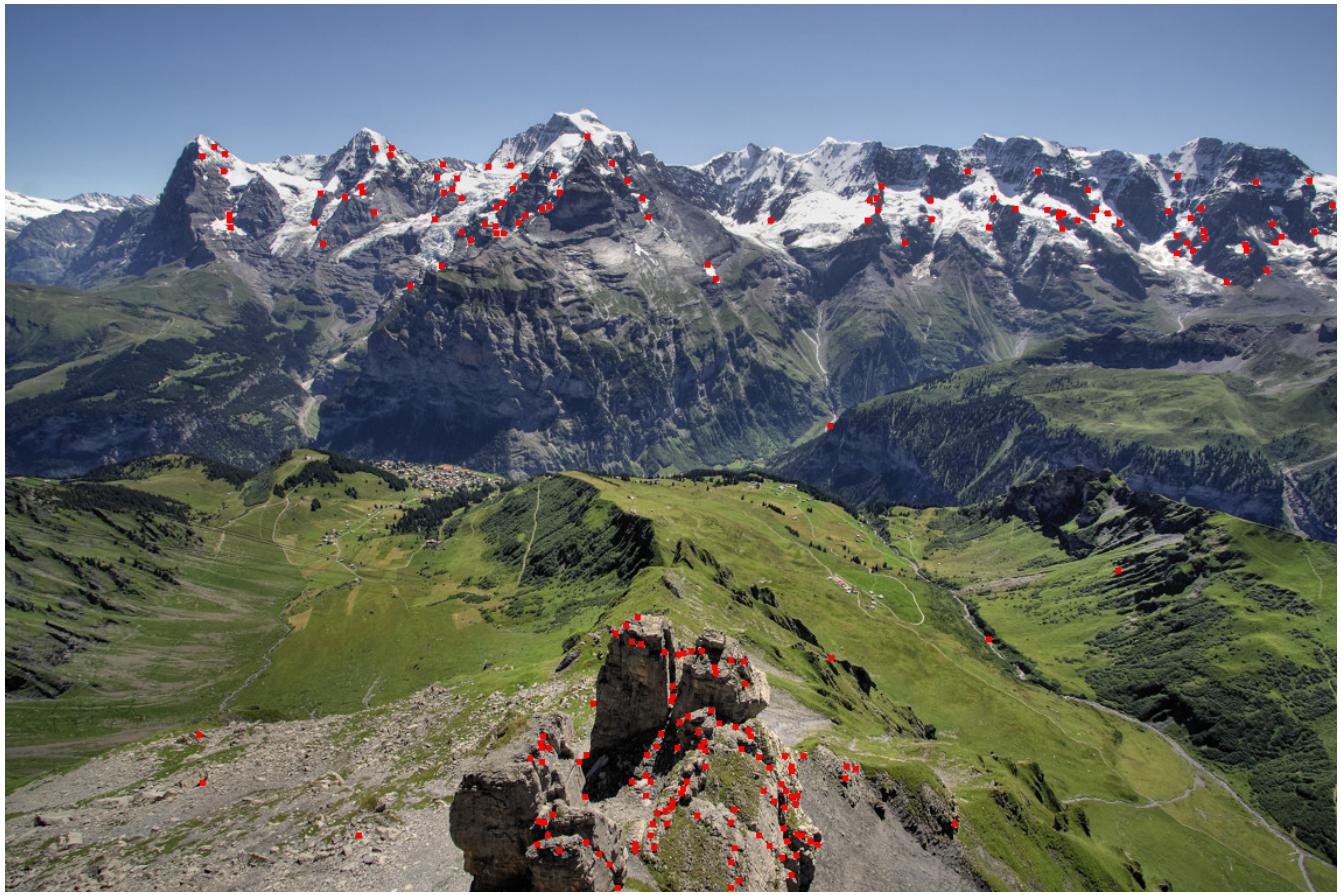


- To have a high value of the Harris criterion, both λ_0 and λ_1 have to be large

Feature detection - corners



Feature detection



Feature detection - corners

- More generally, the patch distance can give greater weight to positions closer to the centre of the patch
- Therefore, using w_i to denote these weights, A can be written :

$$\begin{bmatrix} \sum_{q\in\Psi_p} w_q I_x(q)^2 & \sum_{q\in\Psi_p} w_q I_x(q) I_y(q) \\ \sum_{q\in\Psi_p} w_q I_x(q) I_y(q) & \sum_{q\in\Psi_p} w_q I_y(q)^2 \end{bmatrix} \tag{14}$$

- w_i will often be a 2D centred Gaussian function
- Advantage of Harris detector : rotation invariant
  - The same corner rotated will be detected in the same manner

Feature detection - corners

- In many situations, we would like to impose two properties on our detection :
  - Scale invariance
  - Rotation invariance
- The Harris detector allows for rotation invariance, but not scale invariance
  - Large-scale corners may not be detected
- Ideally, we would like to detect corners, blobs and features at **several scales**
- This leads to the idea of **scale-space**

Feature detection - scale space

- Scale space is a theoretical framework for dealing with images at several scales
  - We will not go too far into this (very rich) theory
  - We present the main ideas here
- Scale space consists of creating a family of images I^t at several scales t

Image at increasing scales

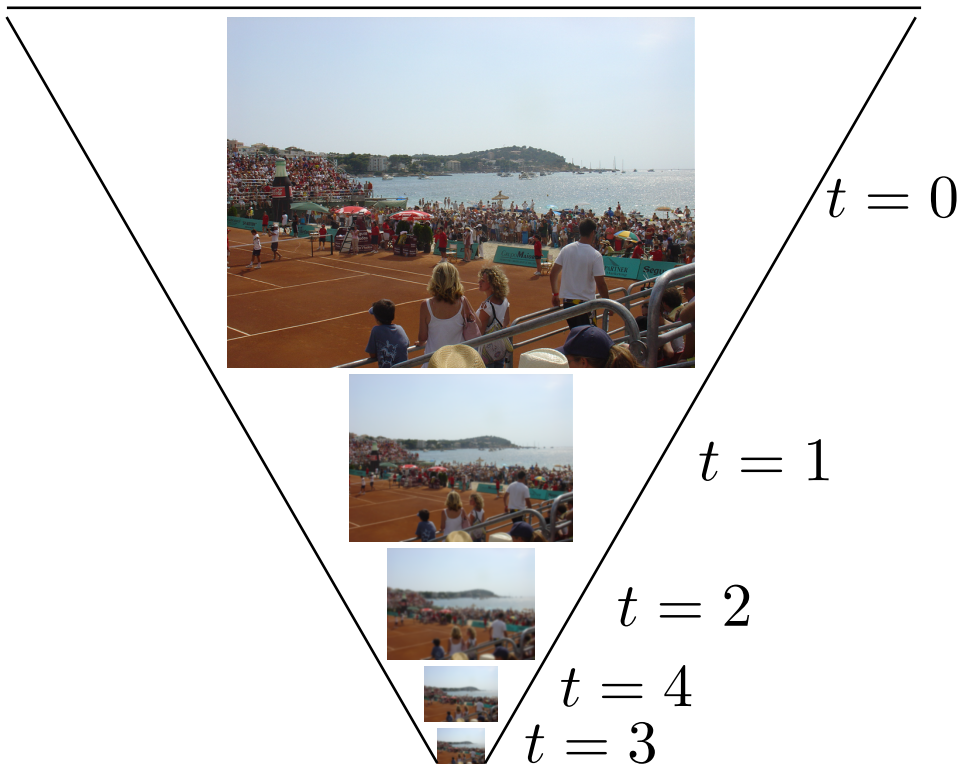
- This is done by with some transformation: $I^t = T_t(I)$

Feature detection - scale space

- There are several axioms which we would like to impose on the filter g_t for creating a scale space :
  - Linearity : $T_t(\alpha I_0 + I_1) = \alpha T_t(I_0) + T_t(I_1)$
  - Shift invariance : $T_t(I_{p+\tau}) = T_t(I)_{p+\tau}$
  - Iterative pyramid cascade : $T_{t_1+t_2}(I) = \alpha T_{t_2}(T_{t_1}(I))$
- Non-creation of local extrema
- Rotational and scale invariance
- It turns out that the isotropic Gaussian filter is the unique choice which verifies these properties
  - $g_t(x, y) = \frac{1}{\sqrt{2\pi t^2}} \exp \frac{-(x^2+y^2)}{2t^2}$

Feature detection

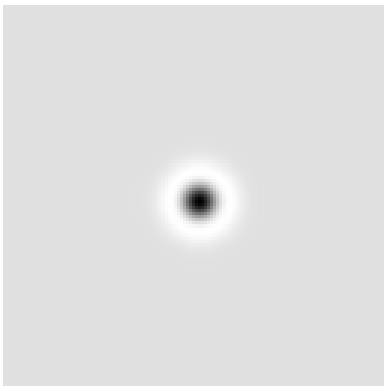
- An image pyramid is created in this manner, with the set $\{ I^0, I^1, \ldots, I^{L-1} \}$
  - L is the number of scales
- We model this mathematically with a 3D scale-space representation : $S_I(x, y, t) = I^t(x, y)$



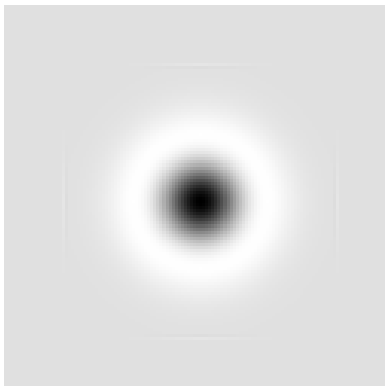
Feature detection

- A useful tool of the scale space is the **Laplacian of Gaussian** (LoG)
- This is the Laplacian operator $\nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$, applied to a scale : $\nabla^2 I * g_t$
  - This allows for detection at several scales
- Why Laplacian ? Laplacian gives a scalar, easy to evaluate
- Laplacian of Gaussian filter looks like a blob

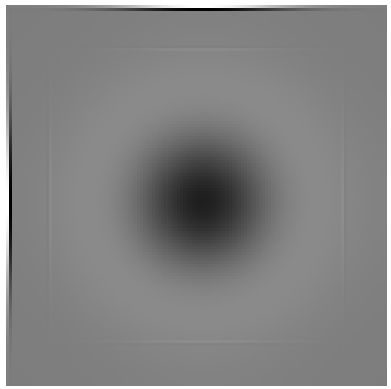
$\sigma = 1$



$\sigma = 5$



$\sigma = 11$



$\sigma = 19$

Feature detection

- Since the Laplacian is approximated with a discrete filter, both the smoothing and Laplacian operators are carried out with a convolution
  - $I^t = I * g_t$ (scale-space)
  - $\nabla^2 I = I * h$
- Furthermore : convolution is **associative and commutative**. Therefore, we have :

$$LoG_t(I) := \nabla^2 T_t(I) = h * I * g_t \quad (15)$$

$$= (h * g_t) * I \quad (16)$$

- This means that we can achieve the LoG in one convolution instead of two ($h * g_t$ can be precomputed)

Feature detection

- A fast approximation to the LoG : the **difference of Gaussians**
- The scale-space representation, using a Gaussian, verifies the difference equation : $\frac{\partial I^t}{\partial t} = \frac{1}{2}\nabla^2 I^t$
- We can approximate $\frac{1}{2}\nabla^2 I^t$ using a Taylor series approximation :

$$I^{t+\delta t} \approx I^t + (\delta t) * \frac{\partial I^t}{\partial t}$$

$$\frac{\partial I^t}{\partial t} \approx \frac{1}{\delta t}\left(I^{t+\delta t} - I^t\right)$$

- Therefore, we have the following approximation of the LoG :

$$\mathsf{LoG}_t(I) \approx \frac{1}{\delta t}\left(I^{t+\delta t} - I^t\right) \tag{17}$$

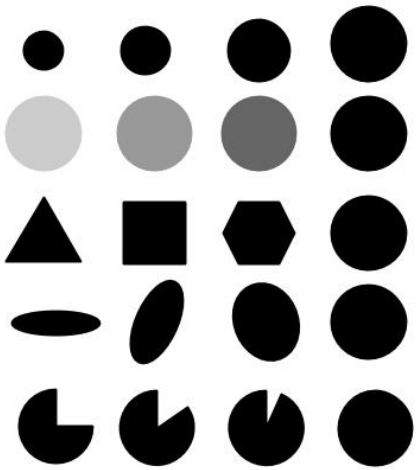
Feature detection - blob detection

- **Blobs** are connected regions of lighter or darker intensity than their surroundings
- Often used blob detector : we search for maxima in both space and scale of scale-normalised Laplacian of Gaussian : $t\,\mathrm{LoG}(I)$

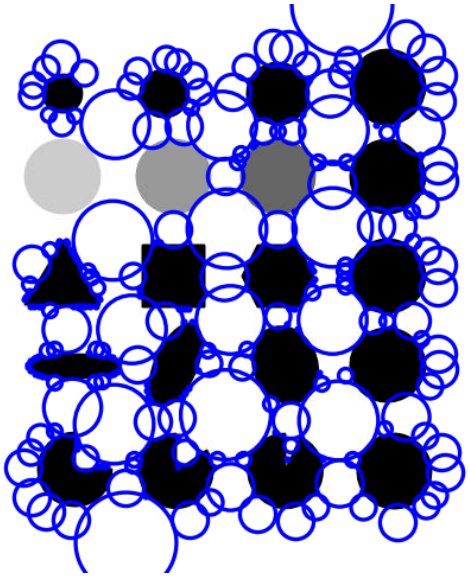
$$(\hat{x}, \hat{y}, \hat{t}) = \arg\max_{x,y,t} |t\,\mathrm{LoG}(I)| \tag{18}$$

- The LoG is often replaced by its faster version, the Difference of Gaussian

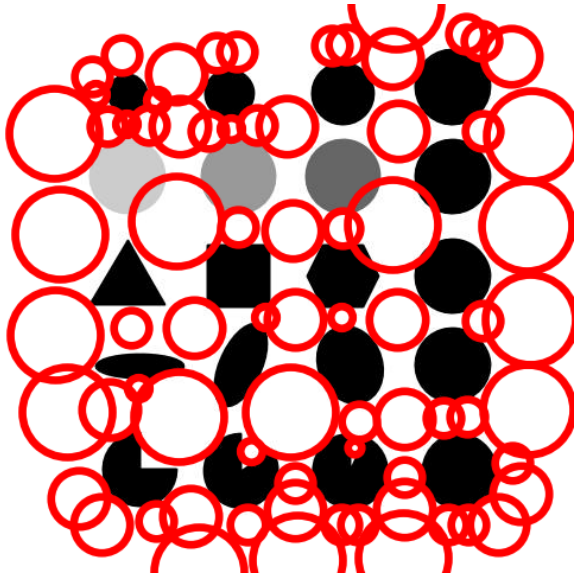
Feature detection



Input



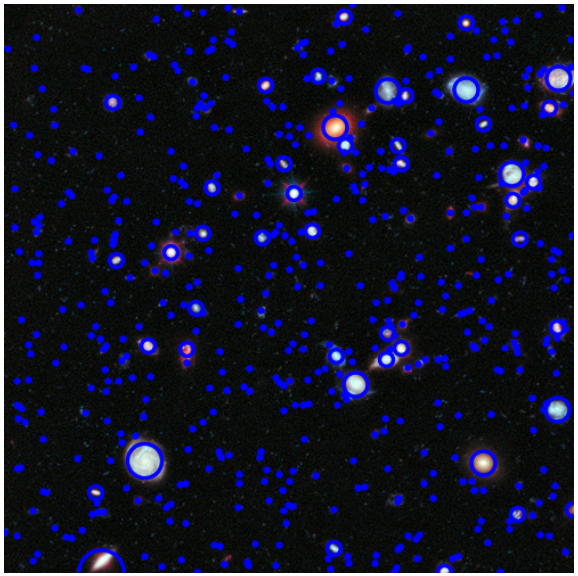
LoG



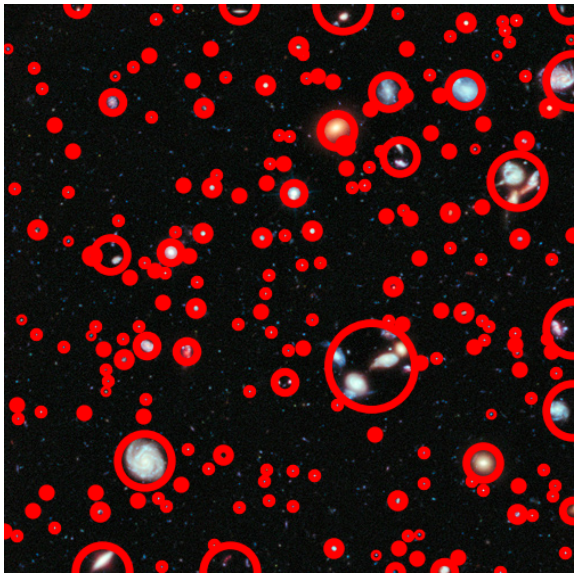
DoG

Feature detection

Input



LoG



DoG

Feature matching

- Another task linked to feature detection : feature **matching**
  - We wish to identify the same object with different image conditions : viewpoint, contrast etc

- Simplest approach : take average gradient in local region
- However, this is not particularly robust. A popular and commonly used approach is the **SIFT** algorithm
- This detects feature points, taking into account several scales, and produces a descriptor for each one

Feature matching - SIFT

SIFT algorithm

- The SIFT* algorithm is a famous approach for detecting features
- Consists of four steps :
  1. Scale-space extrema detection
  2. Keypoint localisation
  3. Orientation assignment
  4. Keypoint descriptor creation

- The SIFT* algorithm has quite a lot of steps, but results in a feature detector/descriptor which is robust, and accounts for translation, scale and rotation changes

* *D. Lowe* ***Distinctive Image Features from Scale-Invariant Keypoints****, IJCV, 2004*

SIFT algorithm - Scale-space extrema detection

- The first step is to detect extrema at multiple scales. These are potentially points of interest (keypoints)
  - Maxima and minima of difference of Gaussian (DoG) D
- The second step is to select the keypoints. For this, we reject points from the first step with low magnitude of D :
  - $|D| < \tau$
- This removes relatively smooth regions

Feature matching - SIFT

- Next, we wish to remove edges (which are not identifiable points)
- We calculate the Hessian of D

$$H = \begin{bmatrix} D_{xx} & D_{x,y} \\ D_{y,x} & D_{y,y} \end{bmatrix} \tag{19}$$

- The eigenvalues of H are proportional to the principal curvatures of D
- If there is one large eigenvalue, and one small, we have an edge
- We remove edge points with the following criterion :

$$\frac{\mathrm{Tr}(H)^2}{\mathrm{Det}(H)} > \frac{(r+1)^2}{r}, \tag{20}$$

- For some threshold r

Feature matching - SIFT

- These first two parts of the algorithm give a list of k keypoints $\{q_0, \ldots, q_k - 1\}$

$$q_i = (p_i, \sigma_i), \tag{21}$$

- Where p_i is the position and σ_i the scale of keypoint i
- Now, we create a **descriptor** for each interest point q_i
  - This descriptor should be rich enough to compare between images

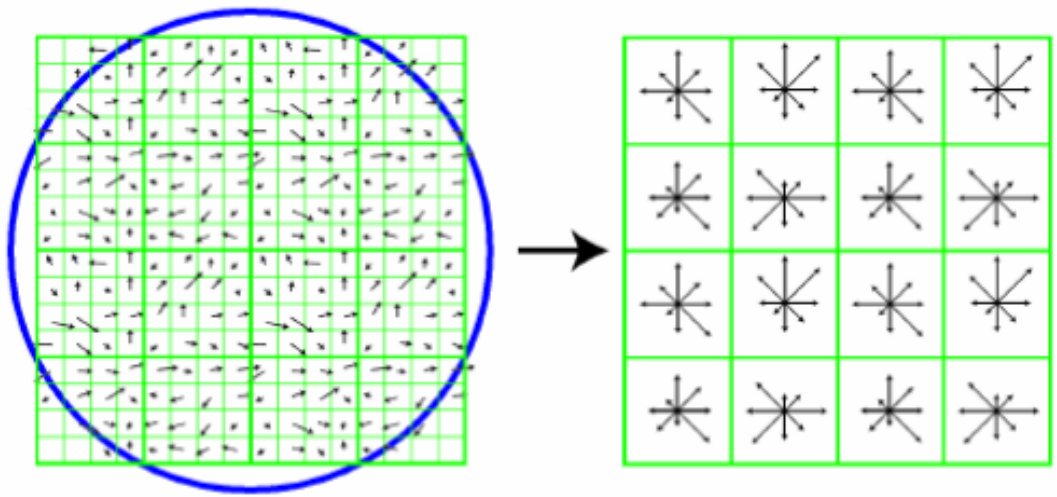
Feature matching - SIFT

- We create an **orientation assignment**
  - We calculate a local histogram of orientations
  - The local gradient direction is quantised into 36 bins : $\frac{1}{36}2\pi$ (10 degrees)
  - For a given point p_i, the histogram is created using the local gradient, weighted by the magnitude of the gradient
  - The orientation is given as the maximum of the histogram
- Now, the keypoints also have an orientation :

$$q_i = p_i, \sigma_i, \theta_i \tag{22}$$

Feature matching - SIFT

- The final step of SIFT is to create a rich local descriptor
- This is done by splitting a local 16×16 window up into 4×4 blocks of 4×4
- Each block has an 8 bin orientation histogram
- Finally, in order to obtain rotational invariance, the main orientation θ_i is subtracted from the gradients



Feature matching - SIFT

- SIFT allows for a robust, rich descriptor which can used for matching similar images patches with invariance to contrast, zoom, rotation (the local gradient histogram is relative to the principal direction)

Thanks to Yann Gousseau for the illustration

Feature matching

Summary

- Detecting prominent edges : Hough transform
- Blob detection : scale space (LoG, DoG)
- Feature matching : SIFT
- These simple tasks are extremely common pre-processing steps for more high-level problems

Summary

Image restoration

- A common task in computer vision is *motion estimation*
- Useful for : tracking, object segmentation

Image restoration

- The main hypothesis used for motion estimation is the **brightness constancy assumption**

- This may be false, for example in presence of light changes/shadows
  - Other priors will be added

Motion estimation

- The brightness constancy hypothesis is formalised as follows

Brightness constancy hypothesis

- Let $I : \mathbb{R}^{m \times n \times t} \rightarrow \mathbb{R}$ be an input **video**
- Let Ω be the domain definition of the video (of size (m, n, τ))
- We want to estimate a **vector field** $(\delta_x, \delta_y)^\intercal$, with a

$$I(x, y, t) = I(x + \delta_x, y + \delta_y, t + \delta_t)$$

Motion estimation - block matching

- Generally, we estimate motion in successive images : $\delta_t = 1$
- Ideally, we would like to solve something similar to the following problem $\arg\min_{\delta_x,\delta_y} \sum_{(x,y)\in\Omega}(I(x,y,t) - I(x+\delta_x, y+\delta_y, t+\delta_t))^2$
- We will see that there are several ways of doing this :
  1. Block matching
  2. Affine motion estimation
  3. Optical flow

Motion estimation - block matching

- The simplest approach in practical terms is called **block matching**

Block matching

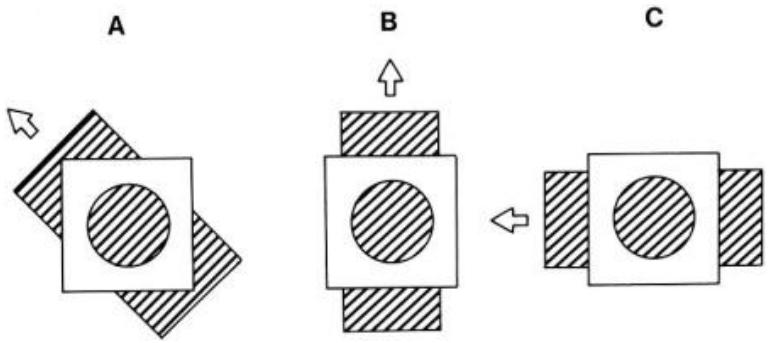
- For each pixel (x, y), we search in a zone radius of r
- We recall $\Psi = \{-a, \ \ldots, a\}^2$ is the patch (block) neighbourhood

$$\underset{(\delta_x,\delta_y)\in\Psi \ | \ \|(\delta_x,\delta_y)\|_\infty \leq r}{\arg\min} \underbrace{\sum_{(\ell,q)\in\Psi} \|I(x+\ell, y+q, t) - I(x+\delta_x+\ell, y+\delta_y+q, t+1)\|_2^2}_{\text{patch distance between } \Psi x,y,t \text{ and } \Psi_{x+\delta_x,y+\delta_y,t+1}}$$

$f(\cdot, \cdot, t)$ $f(\cdot, \cdot, t+1)$

Motion estimation - optical flow

- A problem with block matching, and in general motion estimation, is the **aperture problem**
- Motion estimation may be an ill-posed problem in certain conditions
  - Several solutions are possible



Motion estimation - optical flow

- Example of the aperture problem in a real example

- The solution to this is to **regularise** the motion field, using a **variational framework**
  - The motion should be **smooth** in some sense
- How do we find a smooth solution ? Variational methods !

Motion estimation - optical flow

- Example of the aperture problem in a real example

- The solution to this is to **regularise** the motion field, using a **variational framework**
  - The motion should be **smooth** in some sense
- How do we find a smooth solution ? Variational methods !

Motion estimation - optical flow

- **Optical flow** is a method to estimate motion with a continuous approach
- We wish to estimate the **speeds** of the video $(u, v)^\intercal = (\frac{\delta_x}{\delta_t}, \frac{\delta_y}{\delta_t})^\intercal$ (we want the **derivatives** (infinitesimal displacements) here
- For this, we use a variational approach
- Recall the brightness constancy hypothesis :

$$I(x + \delta_x, y + \delta_y, t + \delta_t) = I(x, y, t). \tag{23}$$

Motion estimation - optical flow

- If we consider that the displacements $(\delta_x, \delta_y, \delta_t)$ are small, then we can carry out a Taylor series development around (x, y, t)

$$I(x+\delta_x, y+\delta_y, t+\delta_t) \approx I(x,y,t) + I_x(x,y,t)\delta_x + I_y(x,y,t)\delta_y + I_t(x,y,t)\delta_t$$

- Using the above equation and Equation (23), we have :

$$\begin{aligned}
\iff & \quad I_x(x,y,t)\delta_x + I_y(x,y,t)\delta_y + I_t(x,y,t)\delta_t = 0 \\
\iff & \quad I_x(x,y,t)\frac{\delta_x}{\delta_t} + I_y(x,y,t)\frac{\delta_y}{\delta_t} = -I_t(x,y,t) \\
\iff & \quad I_x(x,y,t)u + I_y(x,y,t)v = -I_t(x,y,t)
\end{aligned}$$

- Note, abuse of notation : $u = u(x,y,t)$ and $v = v(x,y,t)$, for better readability

Motion estimation - optical flow

- This gives the variational formulation of the brightness constancy hypothesis :

Variational formulation of brightness constancy hypothesis

$$J(u,v) = \int_\Omega \left(I_x(x,y,t)u + I_y(x,y,t)v + I_t(x,y,t)\right)^2 dx\, dy$$

- However, recall that our original goal* is to **regularise** (make smooth) the optical flow
- One way of doing this is to find a solution with a small gradient

* *Remember the aperture problem*

Motion estimation - optical flow

- The regularised variational formulation leads to the Horn-Schunck method of estimating optical flow

Horn-schunck variational formulation for optical flow

$$J(u,v) = \int_\Omega \left(I_x(x,y,t)u + I_y(x,y,t)v + I_t(x,y,t)\right)^2 + \lambda \left(\|\nabla u\|_2^2 + \|\nabla v\|_2^2\right)\, dx\, dy \tag{24}$$

- To solve this, we need to solve the associated Euler-Lagrange equations
- We do not go into this here, requires much more time
  - In short : solve a partial differential equation with a numerical scheme

* *B.K.P. Horn and B.G. Schunck,* ***Determining optical flow****, Artifical Intelligence, 1981*

Motion estimation - optical flow

- Example of optical flow, colour wheel convention 

Motion estimation - optical flow

- Example of optical flow, colour wheel convention 

Motion estimation - optical flow

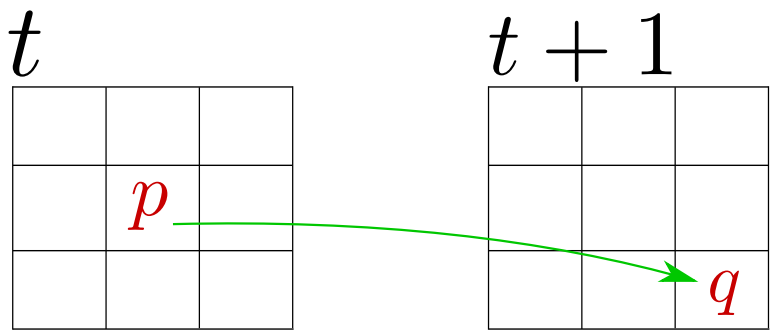
- Example of optical flow, colour wheel convention 

Motion estimation - parametric motion

- Until this point, we have considered **non-parametric** motion
  - Each pixel could have **any motion**
- Parametric motion : **global motion**, motion described by a small number of global parameters
  - The motion of neighbouring pixels is similar

Motion estimation - parametric motion

- Until this point, we have considered **non-parametric** motion
  - Each pixel could have **any motion**
- Parametric motion : **global motion**, motion described by a small number of global parameters
  - The motion of neighbouring pixels is similar
- We will be using linear algebra here, so for ease of notion, we define
  - $p = (p_x, p_y)^\intercal$ a pixel at image t
  - $q = (q_x, q_y)^\intercal$ the position of that pixel in image $t + 1$



- Several types of parametric motion are used : translation, **affine motion**, homographies

Motion estimation

- Affine motion is a specific type of parametric motion
- Can describe the following situations : translation, rotation, zooming
  - Covers most types of **camera motion**

Affine motion

- An **affine transformation** in two dimensions between p and q requires 6 parameters :
  - $\delta = (\delta_x, \delta_y)^\intercal$, two translation coefficients
  - $\theta = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$
- The affine transformation is defined as :

$$q = \theta p + \delta \tag{25}$$

Motion estimation

- Different specific cases covered by affine motion :

Type	Formula	Number of parameters
Translation	$p = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + (\delta_x, \delta_y)^\intercal$	2
Rotation	$p = \begin{bmatrix} \cos(\varphi) & -\sin(\varphi) \\ \sin(\varphi) & \cos(\varphi) \end{bmatrix} + (0,0)^\intercal$	1
Zoom	$p = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} + (0,0)^\intercal$	2

Motion estimation

- Different specific cases covered by affine motion :

Original image

Motion estimation

- Different specific cases covered by affine motion :

Translation

Motion estimation

- Different specific cases covered by affine motion :

Rotation

Motion estimation

- Different specific cases covered by affine motion :

Zoom

Motion estimation

- How to estimate affine motion ?
  1. Set up a series of N points in image t, $p^{(i)}, \{0, \ldots, N-1\}$
  2. Estimate the corresponding points $q^{(i)}$ in image $t+1$ (with block matching, for example)
  3. Solve a least squares problem :

$$\arg\min_{\delta,\theta} \sum_i \|(\theta p^{(i)} + \delta) - q^{(i)}\|_2^2 \tag{26}$$

- If we can reformulate this problem as $\arg\min \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2^2$, the solution is given by the normal equations

$$(\mathbf{A}^\intercal \mathbf{A})^{-1} \mathbf{A}^\intercal \mathbf{y} \tag{27}$$

- We now have to define $\mathbf{A}$, $\mathbf{x}$ and $\mathbf{y}$

Motion estimation

- Define the linear ordering* of the parameters $X = [\ \delta_x, \delta_y, a, b, c, d]^\intercal$
- Recall that $\begin{cases} q_x = & ap_x + bp_y + \delta_x \\ q_y = & cp_x + dp_y + \delta_y \end{cases}$. Therefore :

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & p_x^{(1)} & p_y^{(1)} & 0 & 0 \\ 1 & 0 & p_x^{(2)} & p_y^{(2)} & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & p_x^{(N)} & p_y^{(N)} & 0 & 0 \\ 0 & 1 & 0 & 0 & p_x^{(1)} & p_y^{(1)} \\ 0 & 1 & 0 & 0 & p_x^{(2)} & p_y^{(2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 1 & 0 & 0 & p_x^{(N)} & p_y^{(N)} \end{bmatrix} \downarrow 2N \quad , \quad \mathbf{y} = \begin{bmatrix} q_x^{(1)} \\ q_x^{(2)} \\ \vdots \\ q_x^{(N)} \\ q_y^{(1)} \\ q_y^{(2)} \\ \vdots \\ q_y^{(N)} \end{bmatrix} \downarrow 2N$$

* *This ordering is completely arbitrary*

Motion estimation

- We now have a generic method of determining global motion between two images
- Note : this does not work for more complex motions such a camera panning
- Main technical questions
  - Which points p to choose ? Random points, feature detection
  - How to estimate local motion ? Block matching

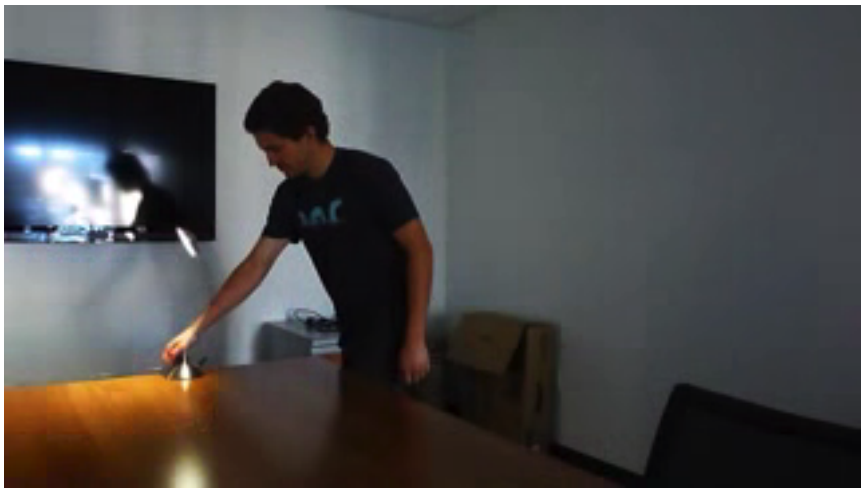
Motion estimation - summary

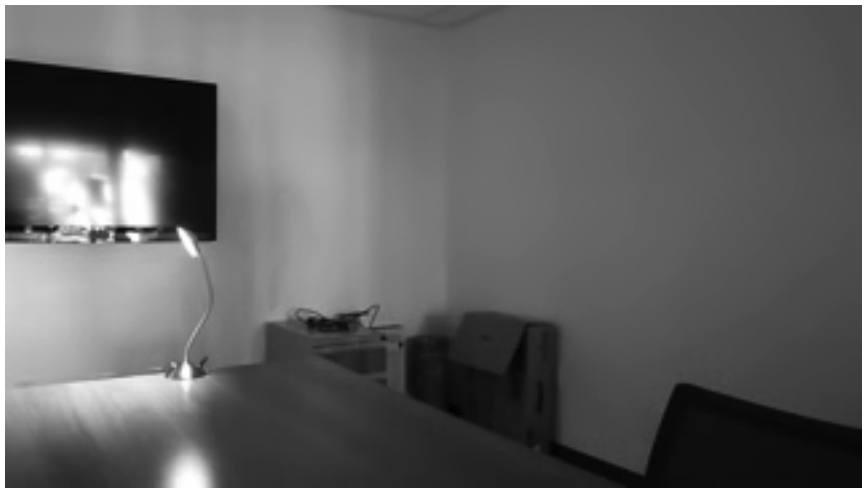
- Block matching
  - Advantages : easy to understand and implement
  - Disadvantages : long execution time, no regularisation (aperture problem)
- Optical flow :
  - Advantages : regularisation, smooth flow field, more robust than block matching
  - Disadvantages : theory more difficult to understand, numerical schemes difficult to implement
- Affine motion :
  - Advantages : easy to impose robustness (take many points), a clear motion model
  - Disadvantages : mostly applies to camera motion or large rigid object motion

Summary

Background estimation

- In video data, it is necessary to separate **moving foreground** and **static background**
  - This is necessary to detect and analyse people/objects
  - Example : a robot navigating around obstacles, isolating moving objects for training
- Also known as background detection/subtraction, foreground detection etc. (mostly the same problem)





Background estimation

- There is a wide array of techniques for this problem
- Here we look at the following
  - Simple temporal filters : mean, median
  - Gaussian mixture models
  - Robust principle component analysis
- We denote a video $\mathbf{X}$ as a $mn \times t$ matrix
  - Each vector of the matrix is a (flattened/vectorised) image
  - (m, n) are the height and width of the image, respectively
  - t is the number of frames of the video

Background estimation

- We can model the video in the following manner :

$$\mathbf{X} = \mathbf{B} + \mathbf{S} \tag{28}$$

- $\mathbf{B}$ is the background
- $\mathbf{S}$ is the foreground. We use $\mathbf{S}$ because it is **sparse** (only non-zero in a few places)

Background estimation

- Simplest approach : temporal filtering
- Each pixel is defined as the median or mean
  - Median is better than mean, robust to the foreground

$$\mathbf{B}_i = \text{median}(\mathbf{X}_{i,.}) \tag{29}$$

- One the background is estimated, foreground region for frame t is simply defined in the positions i where

$$|\mathbf{X}_{i,t} - \mathbf{B}_i| > \tau \tag{30}$$

- τ is a manually set threshold

Background estimation

- Example of temporal median background estimation

Input frame

Estimated background

Foreground

- As you can see, there are several situations which may be problematic
  - Shadows
  - Moving background
  - The threshold is arbitrary

Background estimation

- Example of temporal median background estimation

Input frame

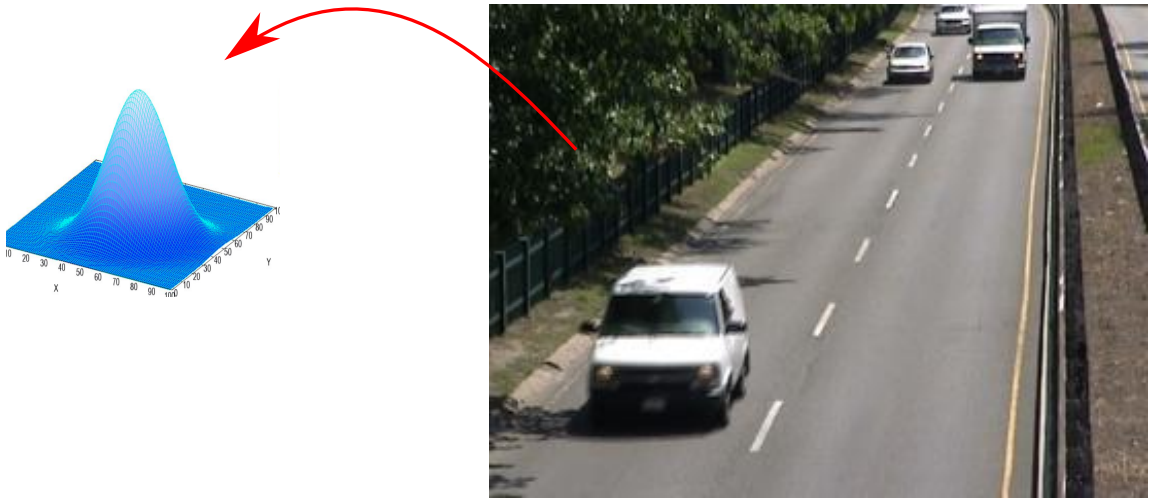
Estimated background

Foreground

- As you can see, there are several situations which may be problematic
  - Shadows
  - Moving background
  - Manually set threshold

Background estimation

- Another approach*: modelling the background probabilistically
- Each pixel is associated with a probability distribution
  - Simplest distribution, Gaussian
- This is a more robust approach to setting the background threshold
  - Threshold is adaptive to each pixel
- **Online approach**, only acces to frames earlier in time



* *C. Wren et al.* ***Pfinder: Real-time tracking of the human body****, PAMI, 1997*

Background estimation

Running Gaussian average background estimation

Data: X, α : learning parameter
Result: μ : average, σ^2 : variance
Initialisation
for $i \in (0, \ldots, mn-1)$ **do**
  $\mu_i \rightarrow$ Local spatial average
  $\sigma^2_{i,j} \rightarrow$ A default value
end
for $t \in (0, T-1)$ **do**
  for $i \in (0, \ldots, m-1) \times (0, \ldots, n-1)$ **do**
    $\mu_i = \alpha X_{i,t} + (1-\alpha)\mu_i$
    $\sigma_i^2 = \alpha(X_{i,t} - \mu_i)^2 + (1-\alpha)\sigma_i^2$
  end
end

Background estimation

- Once the mean μ and variance σ^2 are known for the current frame t, the threshold is determined based on variance

$$\mathbf{S}_i = \begin{cases} X_{i,t} & \text{if} \quad \frac{|X_{i,t} - \mu_i|}{\sigma_i} > \tau \\ 0 & \text{otherwise} \end{cases} \tag{31}$$

- The threshold is not global : adaptive to local variance
- A variant of the algorithm : only update background if pixel is classified as background
  - To avoid slowly moving foreground influencing the background
- Drawback : simple model

Background estimation

- Another, more sophisticated approach, uses Gaussian mixture models*
- Each pixel is associated with a **mixture of Gaussian distributions**
  - Choose K Gaussians with parameters μ^j, $\mathbf{\Sigma}^j_{i,t}$, for $j = 1 \dots K$
  - $\mathbf{\Sigma}^j$ is the covariance matrix of the colour values of pixel (i, t)
    - $\mathbf{\Sigma}^j_{i,t}$ is set to be diagonal (indepenant colours)

$$\mathbb{P}[X_{i,t}] = \sum_{j=1}^{K} w^j_{i,t} \mathcal{N}(X_{i,t} \mid \mu^j_i, \mathbf{\Sigma}^j_{i,t}) \tag{32}$$

- $w^j_{i,t}$ is the weight of the j^{th} Gaussian for pixel i

* *C. Stauffer and W. Grimson,* ***Adaptive background mixture models for real-time tracking****, CVPR 1999*

Background estimation

- Example of background estimation

Background estimation

- Example of background estimation (single Gaussian)

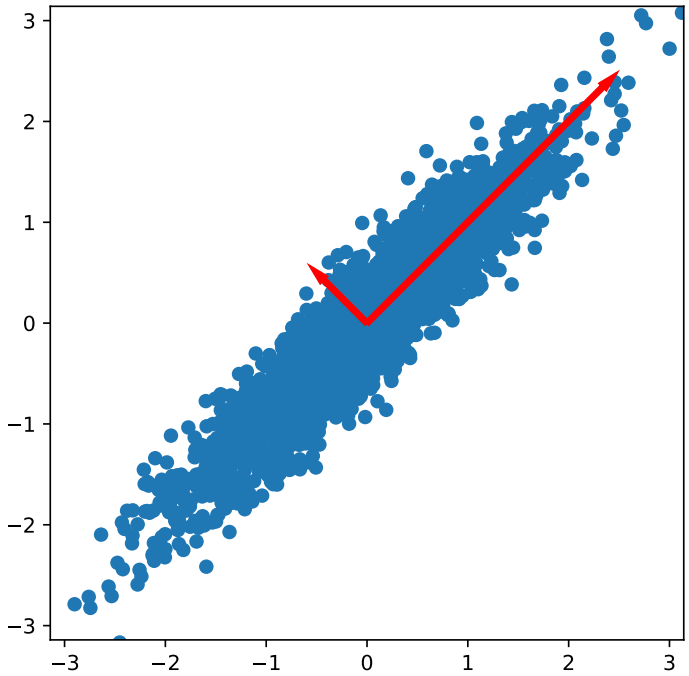
Background estimation

- A final approach : **Robust Principal Component Analysis***
- Why would Principal Component Analysis be useful ?
- Firstly, what is PCA ?

* *E. Candés et al,* ***Robust Principal Component Analysis****, Journal of the ACM, 2011*

Background estimation

- PCA is a dimensionality reduction tool
- Goal of PCA : find the components which best describe the data
  - Orthogonal transformation
  - Orders elements from highest to lowest variance

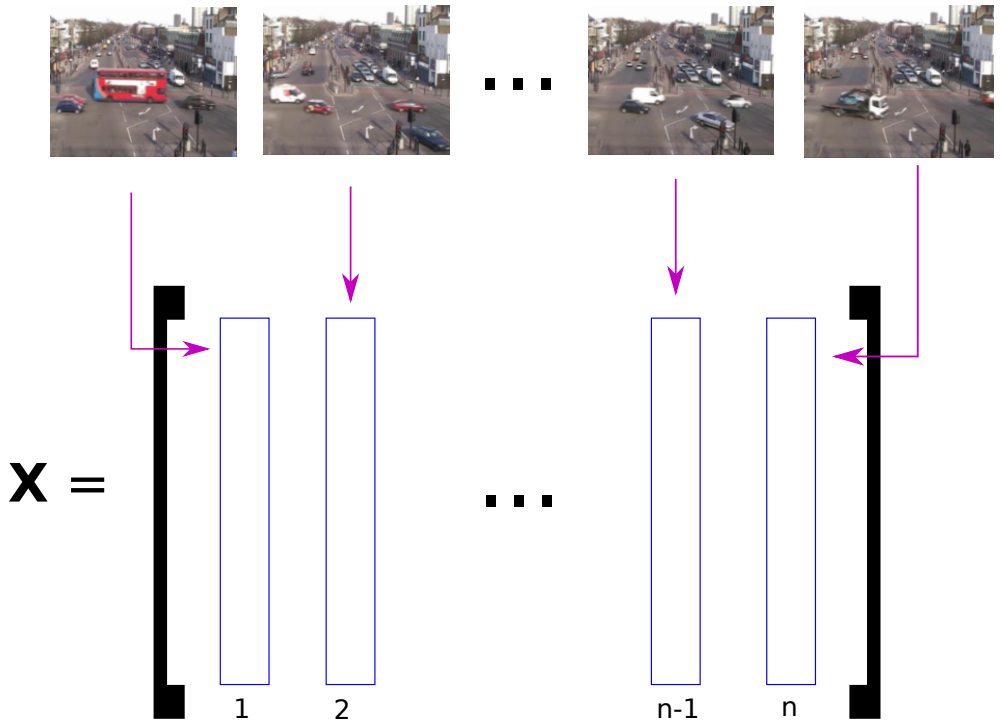


- In summary: few, well-chosen vectors form a basis to descibe the data

Background estimation

- Why is Principal Component Analysis useful here ?

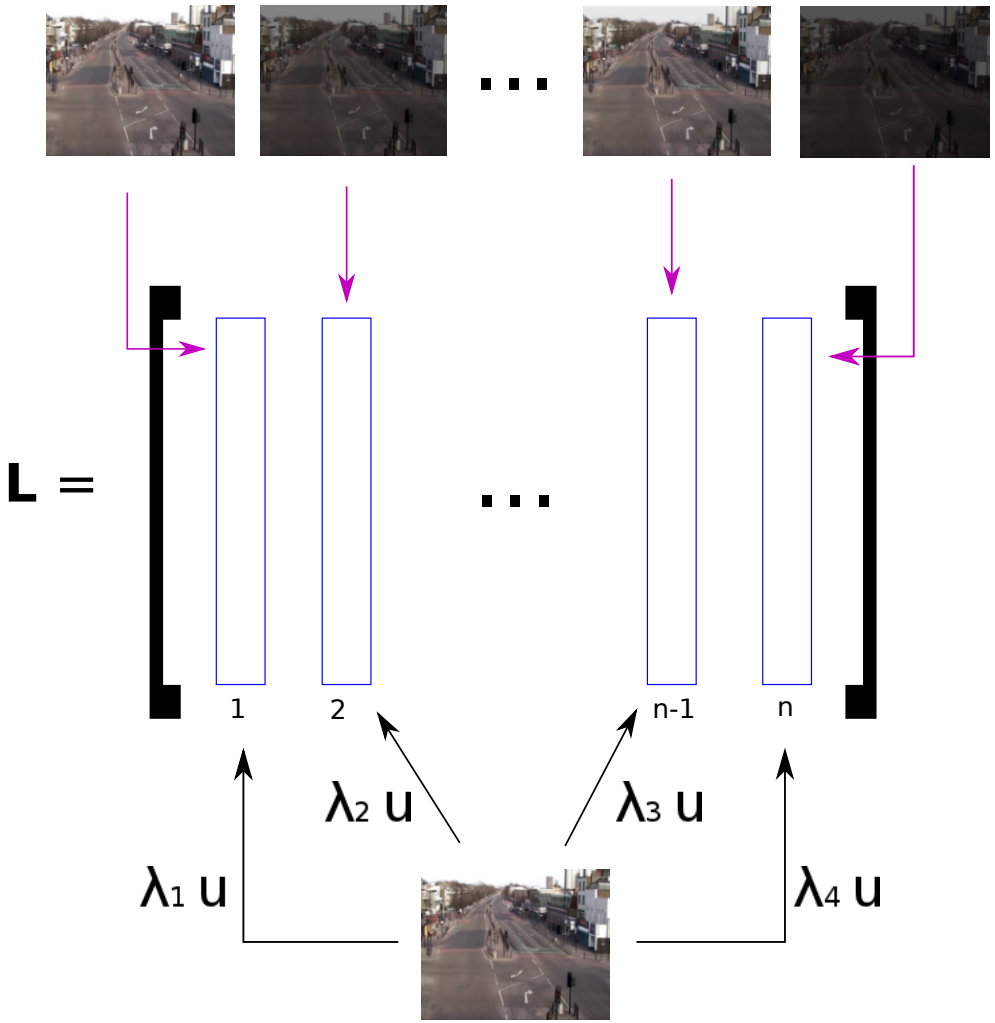
Creation of data matrix $\mathbf{X}$ from video



Background estimation

- Why is Principal Component Analysis useful here ?

  Background with global lighting changes : $\mathrm{rank}(\mathbf{B}) = 1$



Background estimation

- Principal Component Analysis is equivalent to solving a low-rank approximation problem

Low-rank approximation

$$\underset{\mathbf{L}\in\mathbb{R}^{m\times n}}{\arg\min}\ \frac{1}{2}\|\mathbf{X}-\mathbf{L}\|_F^2 \quad \text{subject to} \quad \operatorname{rank}(\mathbf{L}) \leq r \tag{33}$$

- $\mathbf{X}$: input (data) matrix
- $\mathbf{L}$: low-rank approximation of $\mathbf{X}$
- $\mathbf{X}_F^2 = \sum_i \sum_j \mathbf{X}_{i,j}^2$: Frobenius norm

Background estimation

- Principal Component Analysis is equivalent to solving a low-rank approximation problem

Low-rank approximation

$$\underset{\mathbf{L}\in\mathbb{R}^{m\times n}}{\arg\min}\ \frac{1}{2}\|\mathbf{X}-\mathbf{L}\|_F^2 \quad \text{subject to} \quad \operatorname{rank}(\mathbf{L}) \leq r \tag{33}$$

- $\mathbf{X}$: input (data) matrix
- $\mathbf{L}$: low-rank approximation of $\mathbf{X}$
- $\mathbf{X}_F^2 = \sum_i \sum_j \mathbf{X}_{i,j}^2$: Frobenius norm

Solution to Equation (33) uses singular value decomposition of $\mathbf{X}$

- Known as the Eckart-Young-Mirsky theorem (1936)
- The largest k singular values are used

Background estimation

Robust Principal Component Analysis

- *Sparse* term $\mathbf{S}$ is added : $\mathbf{X} \approx \mathbf{B} + \mathbf{S}$
- Low-rank approximation becomes robust to outliers

Input frame

$=$

Low-rank background

$+$

Sparse foreground

Background estimation

- Unfortunately, matrix rank is **non-convex** difficult to optimise :
- Rank constraint replaced by **nuclear norm** penalisation
  - Nuclear norm is a close convex approximation of rank

Robust Principal Component Analysis (relaxed version)[†]

$$\underset{\mathbf{B},\mathbf{S}\in\mathbb{R}^{m\times n}}{\arg\min} \|\mathbf{X}-\mathbf{B}-\mathbf{S}\|_F^2 + \lambda_*\|\mathbf{B}\|_* + \lambda\|\mathbf{S}\|_1. \tag{34}$$

- $\|\mathbf{B}\|_*$ nuclear norm : $\sum_i v_i$
  - v_i are the singular values of the singular-value decomposition of $\mathbf{B}$
- λ_* and λ are weighting parameters

[†] E. J. Candès, X. Li, Y. Ma, J. Wright, *Robust principal component analysis?*, Journal of the ACM (JACM),

Summary

Object detection

- A final, extremely common, problem in computer vision is **object detection**
- By object detection, we mean in fact several things :
  - Detection of presence
  - Object localisation (where is the object ?)
  - Object recognition (eg whose face is this ?)
- Object detection, ie analysing objects in a scene, is one of the hardest problems in computer vision. Why is this ? Variations in :
  - Pose, viewpoint, zoom, rotation
  - Lighting conditions, colour
  - Noise, occlusion
- This is an extremely vast field, and may depend on the type of object
- In this introduction to object detection, we concentrate on the classical example of **faces**

Face detection

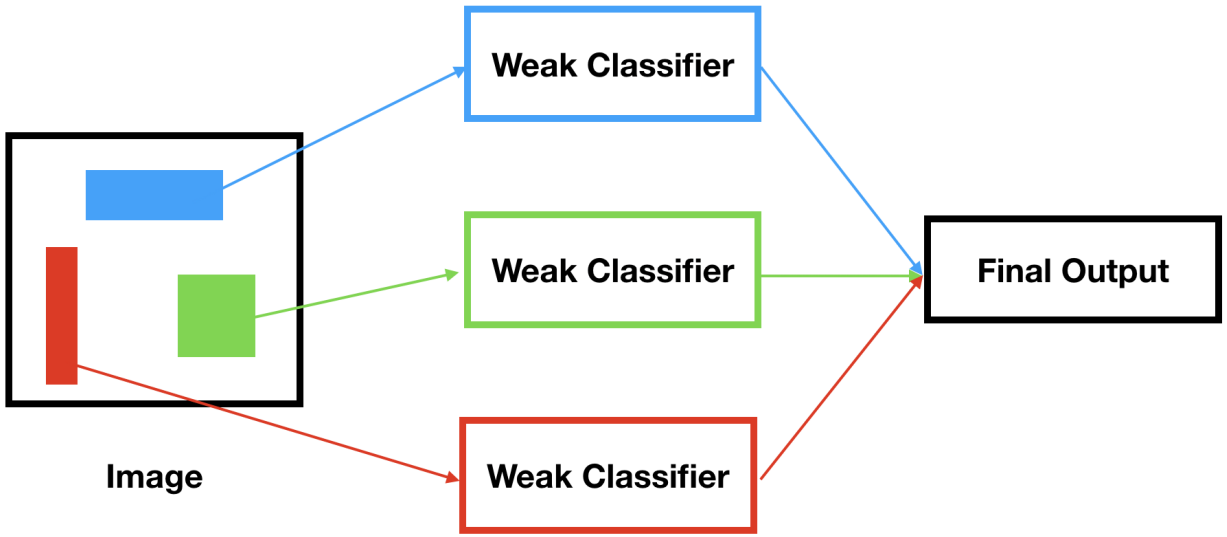
- Since the task is one of the most complex, there is no one unifying framework
- There are several general types of approaches :
  - Feature-based
    - Low-level features (edge, skin texture)
    - Shape models
    - Mix of features, Viola-Jones
  - Image-based
    - Statistical approach : PCA, SVM
    - Eigenfaces
    - Neural networks

Face detection

- First, some notation. This is the first classification problem we have come across
- Let $\{\mathbf{x}_0, \ldots, \mathbf{x}_{N-1}\}$ represent the database, with $\mathbf{x}_0$ the i^{th} input data point (a vector/flattened image)
  - N is the size of the database
- y_i is the i^{th} data vector's label

Viola-Jones algorithm

- The main idea of the Viola-Jones algorithm is to combine several "weak" classifiers to create a robust detection
  - Weak in the sense that their performances do not have to be great
- This is known as **boosting**
- While the training of the algorithm is long, the classification is fast
  - Each classifier has few parameters



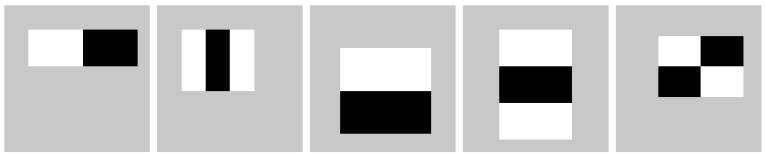
Face detection

Viola-Jones algorithm

- The Viola-Jones algorithm has the following steps :
  1. Extract Haar features
  2. Create integral image
  3. Adaboost training
  4. Cascaded classification

Face detection

- Haar features get their name from the Haar wavelet basis
- The basis of Haar wavelets is a discontious 1–1 function
- Thus, the 2D Haar features are differences in different rectangular regions



- These features make sense for faces, as we can see below

Images from Wikipedia and ***An Analysis of the Viola-Jones Face Detection Algorithm****, Y-Q. Wang, IPOL 2014*

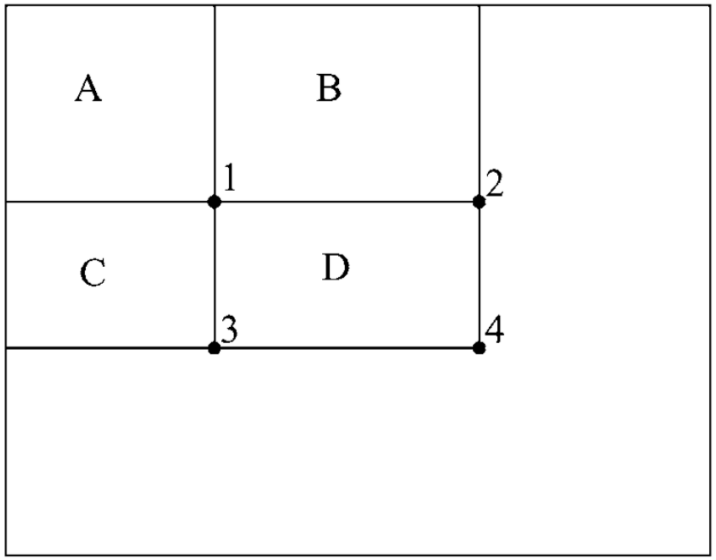
Face detection

- These Haar patterns are calculated for all different sizes of rectangle, non-centred, in a predefined defined image size
- Many different features
  - 162336 features for a 24×24 size image
- How are these features calculated quickly ? **Integral image**
  - Tool for calculating pixel differences over rectangular regions in images
- Define $\tilde{I}$, the integral image of I, recursively as :

$$\tilde{I}(x,y) = \tilde{I}(x-1,y) + \tilde{I}(x,y-1) + I(x,y)$$
$$\forall (x,y) \in \{1,\ldots,m-1\} \times \{1,\ldots,n-1\} \quad (35)$$

Face detection

- The sum of the values in the region D can be calculated using simple subtractions from $\tilde{I}$
  - Area$(D) = \tilde{I}(x_4, y_4) - \tilde{I}(x_3, y_3) - \tilde{I}(x_2, y_2)$



- This greatly reduces the cost of calculating Haar features
  - This approach has been widely used in general in computer vision

Image from https://medium.com/

Face detection

- Back to the Viola-Jones algorithm
- Let $f_i(\mathbf{x})$ be the i^{th} Haar feature of the image $\mathbf{x}$
- In this algorithm, the simple (weak) classifiers are given by :

$$h(x) = \mathbb{1}_{\alpha f_i(\mathbf{x}) < \alpha \tau} \tag{36}$$

- τ is a threshold, $\alpha \in \{-1, 1\}$ is the *polarity* of the classifier
- The polarity (or toggle) can be switched to make the classifier choose either decision half-plane to give a positive result

Face detection

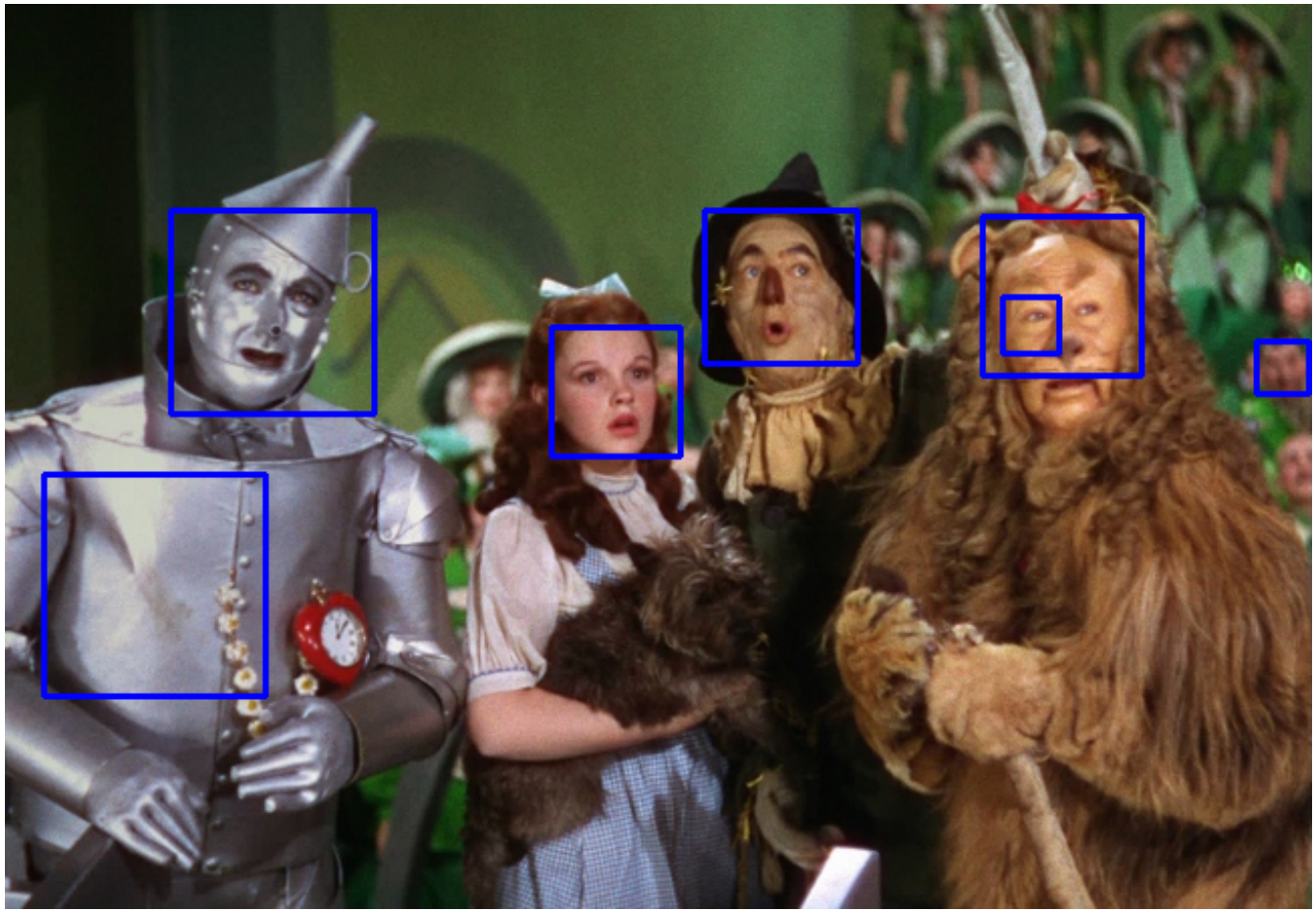
- The final step of the Viola-Jones algorithm is to look for faces over all positions and scales
  - However, this would take a very long time with a good classifier directly
  - Vast majority of proposals are easy to reject : only spend time on likely ones
- To address this problem, the Viola-Jones algorithm adopts a **cascaded approach** :
  - Simple classifiers at the beginning, increasingly complex towards end
  - The vast majority of detections are negatives and easy to detect at the beginning of the cascade
  - Each level is trained by Adaboost
- The final detections have gone through every layer and had positive results

Example of Viola-Jones face detection

Input image

Face detection

Example of Viola-Jones face detection



Detected faces

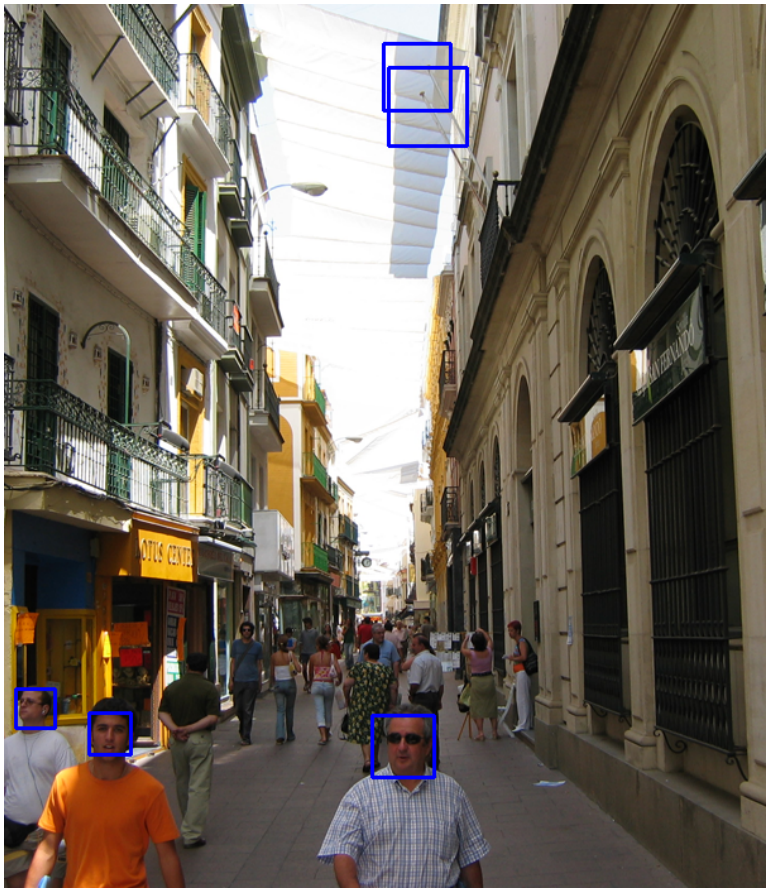
Face detection

Example of Viola-Jones face detection

Input image

Face detection

Example of Viola-Jones face detection



Detected faces

Example of Viola-Jones face detection

Input image

Example of Viola-Jones face detection

Detected faces

Face detection

- The details of Viola-Jones are quite intricate
- Main takeaway : create very simple features, use many simple classifiers
- Combined, these classifiers give a **fast** and **robust** detection
- This general approach, hand-crafted features followed by statistical detection is a standard computer vision approach to object detection (before deep learning ...)

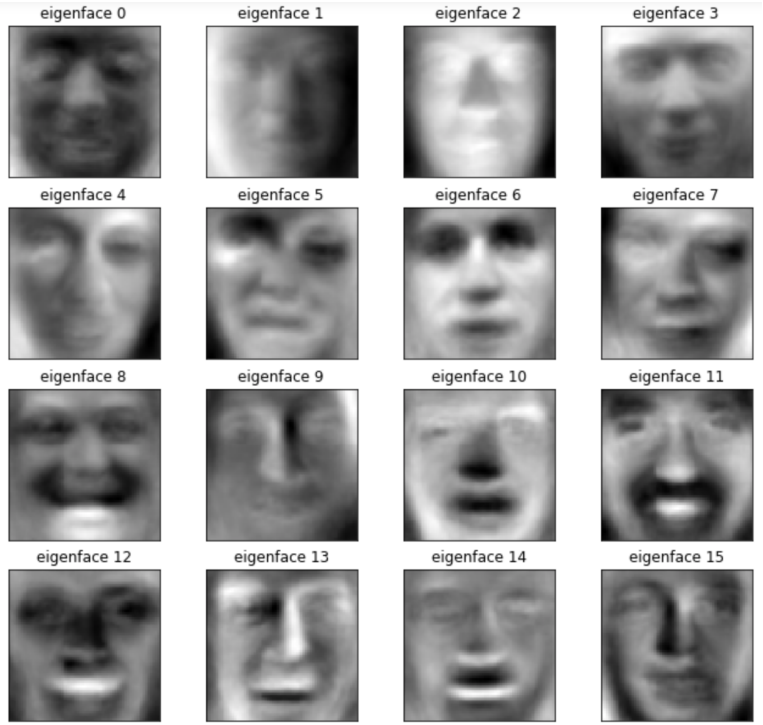
Face classification

- **Face recognition** is the task of identifying a face present in a database
  - Multi-class classification
- A commonly used algorithm, for face recognition : Eigenfaces*
- This is quite similar in spirit to the RPCA for background recognition

* *M. Turk and A. P. Pentland,* ***Face recognition using eigenfaces****, CVPR 2001*

Face classification

- A PCA is carried out on a database of faces. This give a basis $\{\mathbf{v}_0, \ldots, \mathbf{v}_{M-1}\}$ to describe the face images
  - M is the number of elements in the basis with $M < N$



Face recognition

- A new query face $\hat{\mathbf{x}}$ is projected onto the basis
- The coefficients $\hat{e}(j)$ of the projection of $\hat{\mathbf{x}}$ are used to find the closest face in the database, with the Euclidian distance between the coefficient vector of $\hat{\mathbf{x}}$ and the coefficients of the elements of the database

$$\arg\min_i \sum_j (\hat{e}(j) - e_i(j))^2 \tag{37}$$

Conclusion

Conclusion

- We have looked at the following, fundamental, problems in computer vision
  - Segmentation
  - Feature detection
  - Motion estimation
  - Background estimation
  - Object detection/recognition
- Several prominent mathematical tools/techniques are often employed :
  - Calculus of variations (variational approaches)
  - Linear algebra (principal direction of variation)
    - Principal Component Analysis (PCA)
  - Hand-crafted feature detection + standard classifiers

Conclusion

- Most of the approaches seen here are based, in one way or another, on the idea of **features** :
  - Edges, corners
  - Haar features
- Next week, we look at two, specialised, more recent, commonly used approaches
  - Patch-based methods
  - Deep learning based methods
- At the heart of these approaches are two answers to the problem of which features to use
  - Small squares of images (patches)
  - The algorithm itself **learns the features** (deep learning)