Image Content Description

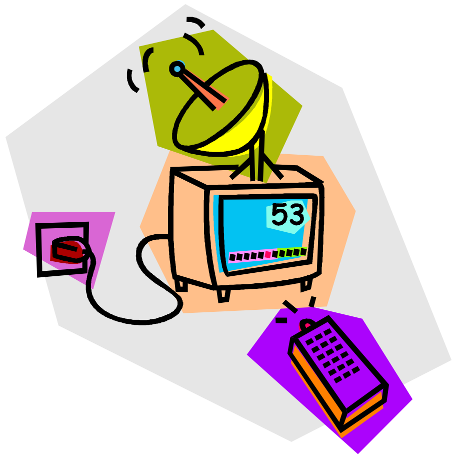
Marco CAGNAZZO
TELECOM ParisTech
cagnazzo@telecom-paristech.fr

Motivation

- Huge amount of audiovisual (AV) information available
  - Digital archives, WWW, Broadcast, personal and professionals databases, …
- Users confronted with almost infinite amount of AV data
- Information is useful only if it is easy to find, retrieve, access, filter and manage

Motivation

- Necessity of describing content:
  - Database retrieval
  - Broadcast channel selection
  - Multimedia editing and services
- This description should be *standardized* for *interoperability*
- Descriptions useful to: professionals, end users, computational systems
- The answer to these challenges is MPEG-7



What is MPEG?

- MPEG: Moving Picture Experts Group (Created in 1988)
- ISO (Int. Standards Organization) / IEC (Int. Electro-technical Commission)
  - ISO/IEC JTC 1 / SC 29 / WG 11
- Develops standards for the coded representation of moving pictures and associated audio

MPEG Standards

MPEG-1: Storage of moving picture and audio on storage media (CD-ROM)	*11 / 1992*
MPEG-2: Digital television	*11 / 1994*
MPEG-4: Coding of natural and synthetic media objects for multimedia applications	*v1: 09 / 1998* *v2: 11 / 1999*
MPEG-7: Multimedia content description for AV material	*09 / 2001*
H.264 : Advanced video coding	*12/2003*
HEVC: High Efficiency video coding	*12/2012*
H.265	*2019 or 2020*

Example of queries

- Text (keywords):
  - Find AV material with subject corresponding to some keywords
- Semantic description:
  - Find AV material corresponding to a specified semantic
- Image as an example:
  - Find an image with similar characteristics (global or local)
- A few notes of music:
  - Find corresponding musical pieces or movies
- Low level features (example: motion):
  - Find video with specific object motion trajectories

Practical work: Python

- Go to

  /cal/softs/anaconda/anaconda2/bin De là il faut faire
- Run

  ./spyder &
  - or

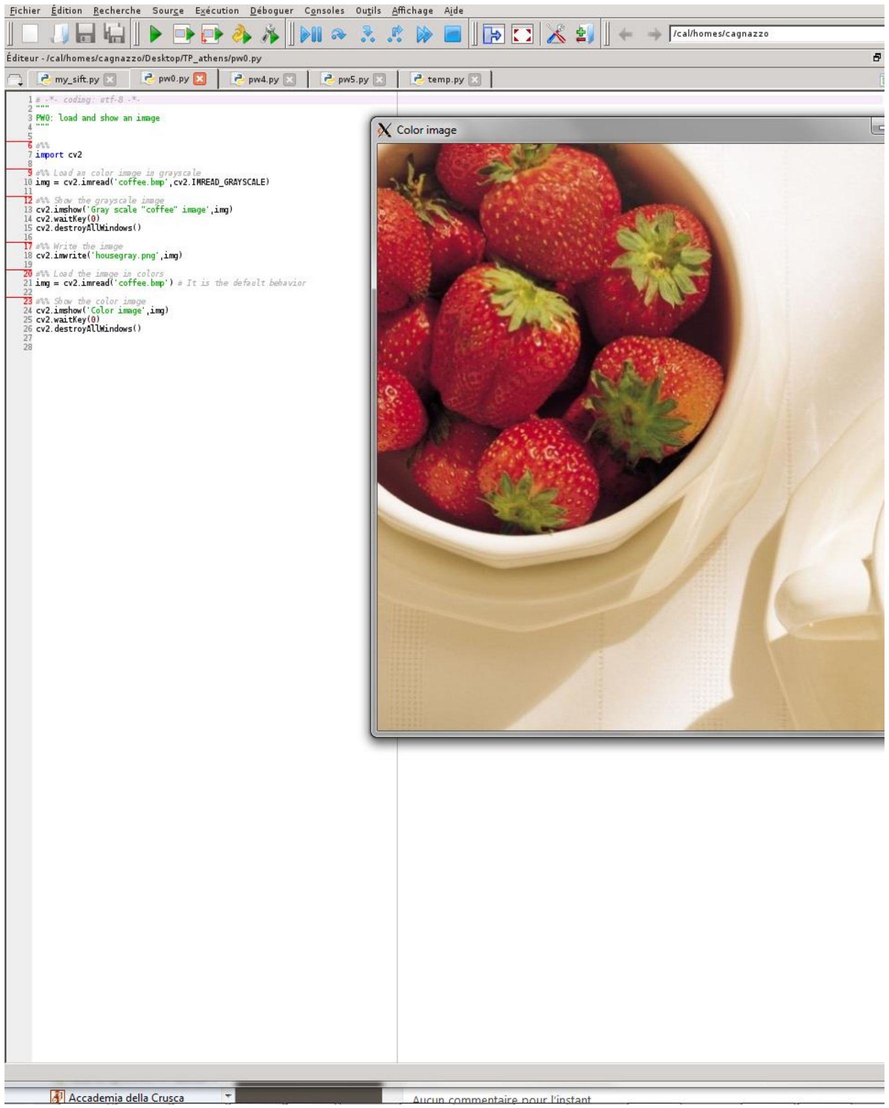
  ./ipython

- Link to the Python tutorial: find it here
  - cagnazzo.wp.imt.fr

- Download the scripts and the test images

PW0

- Load and display images


Fichier Édition Recherche Source Exécution Déboguer Consoles Outils Affichage Aide
/cal/homes/cagnazzo
Éditeur - /cal/homes/cagnazzo/Desktop/TP_athens/pw0.py
my_sift.py
pw0.py
pw4.py
pw5.py
temp.py
Color image
Accademia della Crusca
Aucun commentaire pour l'instant

```python
# -*- coding: utf-8 -*-
"""
PW0: load and show an image
"""

#%%
import cv2

#%% Load an color image in grayscale
img = cv2.imread('coffee.bmp',cv2.IMREAD_GRAYSCALE)

#%% Show the grayscale image
cv2.imshow('Gray scale "coffee" image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()

#%% Write the image
cv2.imwrite('housegray.png',img)

#%% Load the image in colors
img = cv2.imread('coffee.bmp') # It is the default behavior

#%% Show the color image
cv2.imshow('Color image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Feature, descriptors, descriptor value

- A feature is a deliberately faint concept
- Features cannot be compared without meaningful representation (Descriptor) and instantiation (Descriptor Value)

Descriptor (D)

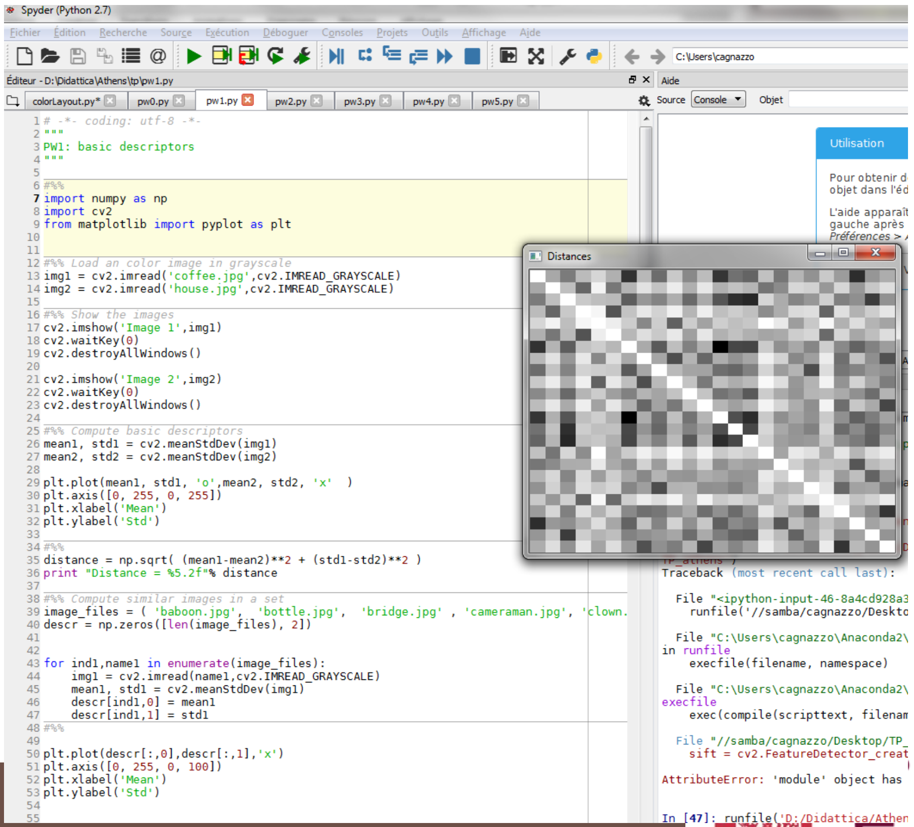
- Definition: A representation of a feature, defining its semantics and syntax (data type, legal values)
- Example: `RGB-color: [int, int, int]`

A Descriptor value is an instance of a Descriptor

PW1

- First basic descriptors
- Average and standard deviation
- Computed on whole the image or on a partition
- Results in a vector of numbers that can be compared within a set of images

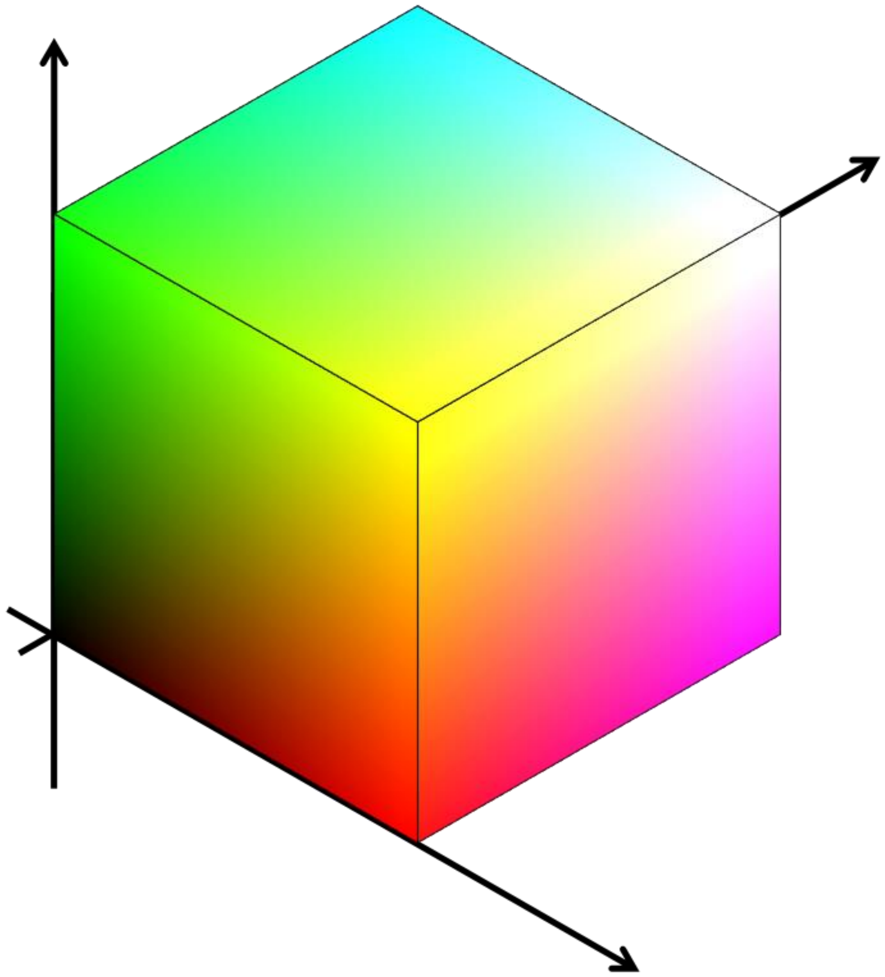
- Do the practical work PW1
  - Descriptors can be computed *image-wise* or *block-wise*

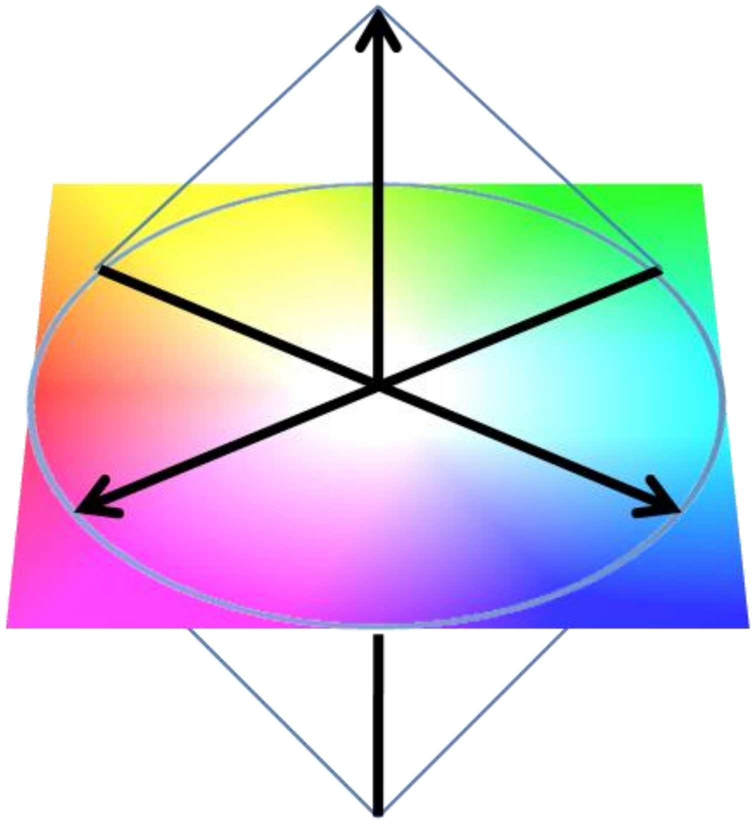

Spyder (Python 2.7)
Fichier Édition Recherche Source Exécution Déboguer Consoles Projets Outils Affichage Aide
C:\Users\cagnazzo
Éditeur - D:\Didattica\Athens\tp\pw1.py
colorLayout.py* pw0.py pw1.py pw2.py pw3.py pw4.py pw5.py
Aide
Source Console Objet
Utilisation
Pour obtenir d objet dans l'éd
L'aide apparaît gauche après Préférences >
Distances

```
# -*- coding: utf-8 -*-
"""
PW1: basic descriptors
"""

#%%
import numpy as np
import cv2
from matplotlib import pyplot as plt


#%% Load an color image in grayscale
img1 = cv2.imread('coffee.jpg',cv2.IMREAD_GRAYSCALE)
img2 = cv2.imread('house.jpg',cv2.IMREAD_GRAYSCALE)

#%% Show the images
cv2.imshow('Image 1',img1)
cv2.waitKey(0)
cv2.destroyAllWindows()

cv2.imshow('Image 2',img2)
cv2.waitKey(0)
cv2.destroyAllWindows()

#%% Compute basic descriptors
mean1, std1 = cv2.meanStdDev(img1)
mean2, std2 = cv2.meanStdDev(img2)

plt.plot(mean1, std1, 'o',mean2, std2, 'x'  )
plt.axis([0, 255, 0, 255])
plt.xlabel('Mean')
plt.ylabel('Std')

#%%
distance = np.sqrt( (mean1-mean2)**2 + (std1-std2)**2 )
print "Distance = %5.2f"% distance

#%% Compute similar images in a set
image_files = ( 'baboon.jpg',  'bottle.jpg',  'bridge.jpg' , 'cameraman.jpg', 'clown.
descr = np.zeros([len(image_files), 2])


for ind1,name1 in enumerate(image_files):
    img1 = cv2.imread(name1,cv2.IMREAD_GRAYSCALE)
    mean1, std1 = cv2.meanStdDev(img1)
    descr[ind1,0] = mean1
    descr[ind1,1] = std1
#%%

plt.plot(descr[:,0],descr[:,1],'x')
plt.axis([0, 255, 0, 100])
plt.xlabel('Mean')
plt.ylabel('Std')
```

```
Traceback (most recent call last):

  File "<ipython-input-46-8a4cd928a3
    runfile('//samba/cagnazzo/Deskto

  File "C:\Users\cagnazzo\Anaconda2\
in runfile
    execfile(filename, namespace)

  File "C:\Users\cagnazzo\Anaconda2\
execfile
    exec(compile(scripttext, filenam

  File "//samba/cagnazzo/Desktop/TP_
    sift = cv2.FeatureDetector_creat

AttributeError: 'module' object has


In [47]: runfile('D:/Didattica/Athen
```

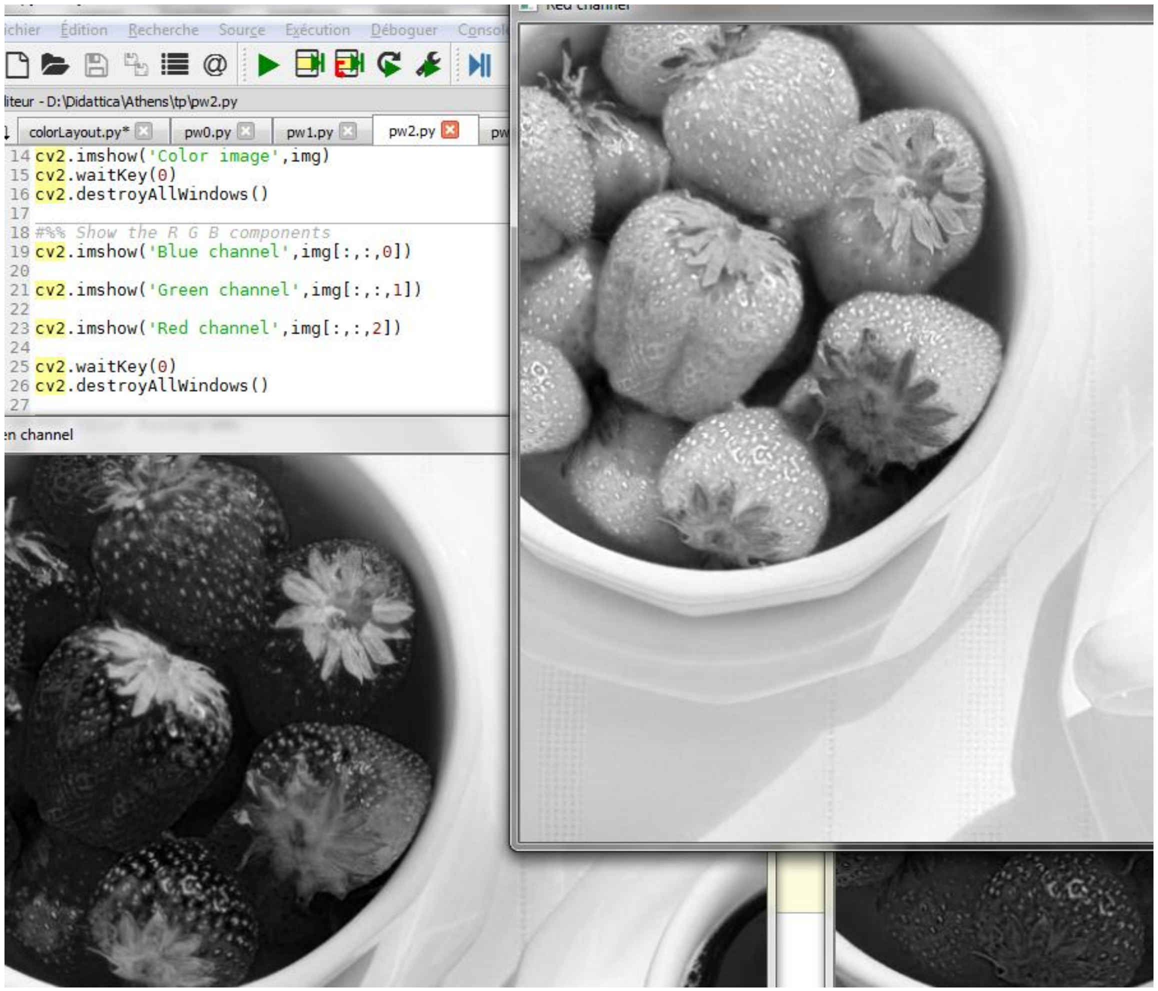
Color spaces





PW2

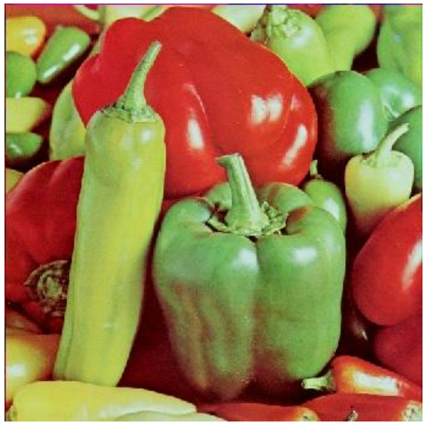
- Load and display a color image
- Show color components
- Change color space
- Use HSV color space to find objects



```
cv2.imshow('Color image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()

#%% Show the R G B components
cv2.imshow('Blue channel',img[:,:,0])

cv2.imshow('Green channel',img[:,:,1])

cv2.imshow('Red channel',img[:,:,2])

cv2.waitKey(0)
cv2.destroyAllWindows()
```

MPEG-7 standardized color descriptors (1/5)

- Dominant Color(s): *DominantColor*
  - 1-8 dominant colors in image / region
  - D = { ($\mathbf{c}_i$ p_i v_i), i=1,...,N, s}
  - Color, percentage of pixels of this color, variance of color value (optional), spatial coherency of color repartition
  - *Similarity retrieval, color browsing*

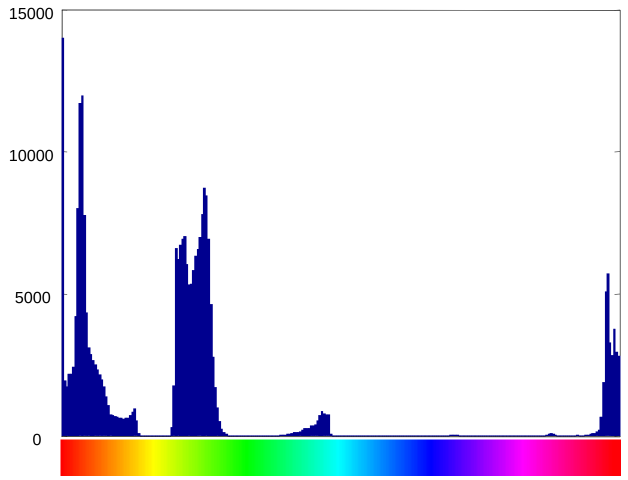


🟥	**40%**	**0.05**
🟩	**31%**	**0.01**
🟧	**12%**	**0.02**

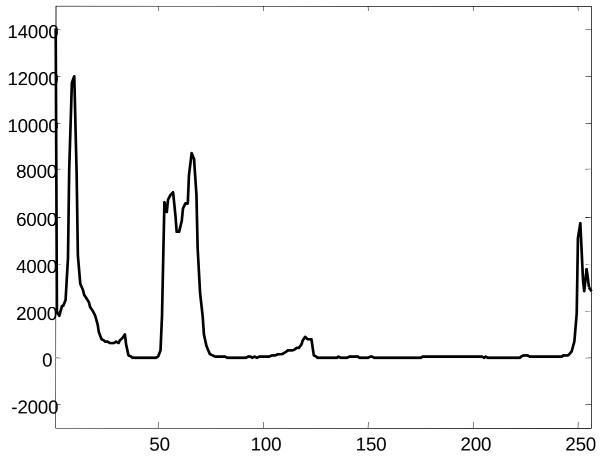
MPEG-7 standardized color descriptors (2/5)

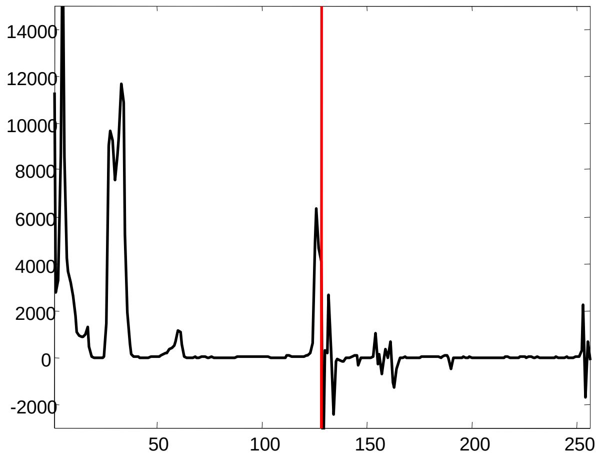
- Color Content (histogram): *ScalableColor*
  - HSV color histogram (256 bins)
    - H is quantized on 4 bits; S and V on 2 each
    - Non uniform quantization (enhance small values)
    - Haar transform (Scalability)
  - scalable in number of coefficients / bits per coefficients
  - slower than ***DominantColor*** but more effective in some application
  - *Image-to-image matching, color-based retrieval*

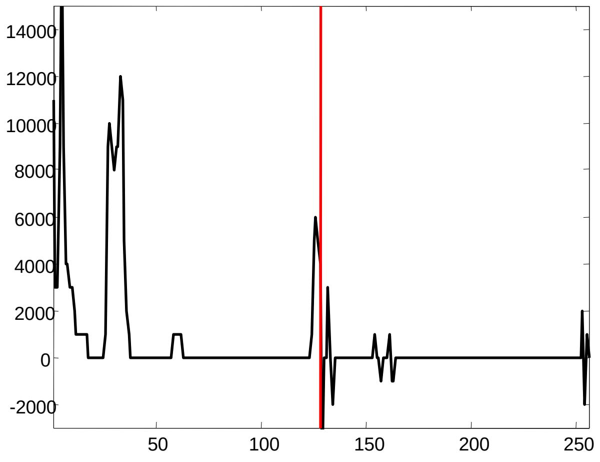
MPEG-7 standardized color descriptors (2/5)



ScalableColor

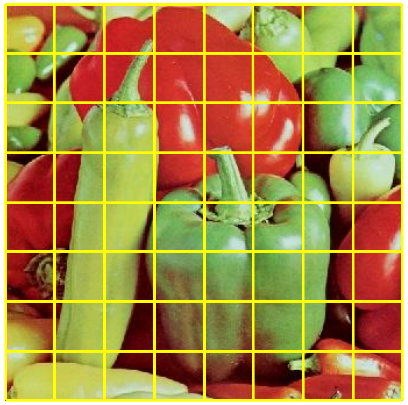
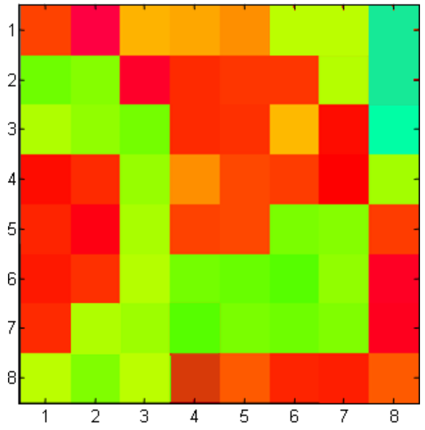


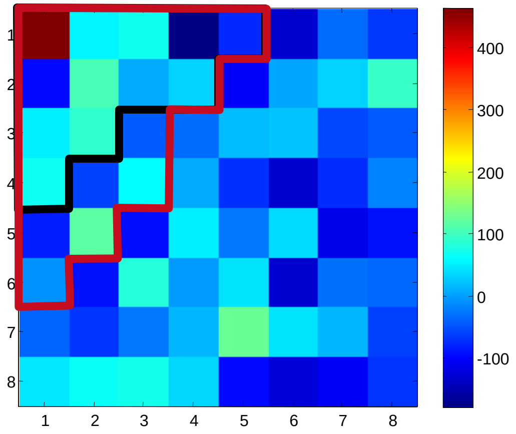




MPEG-7 standardized color descriptors (3/5)

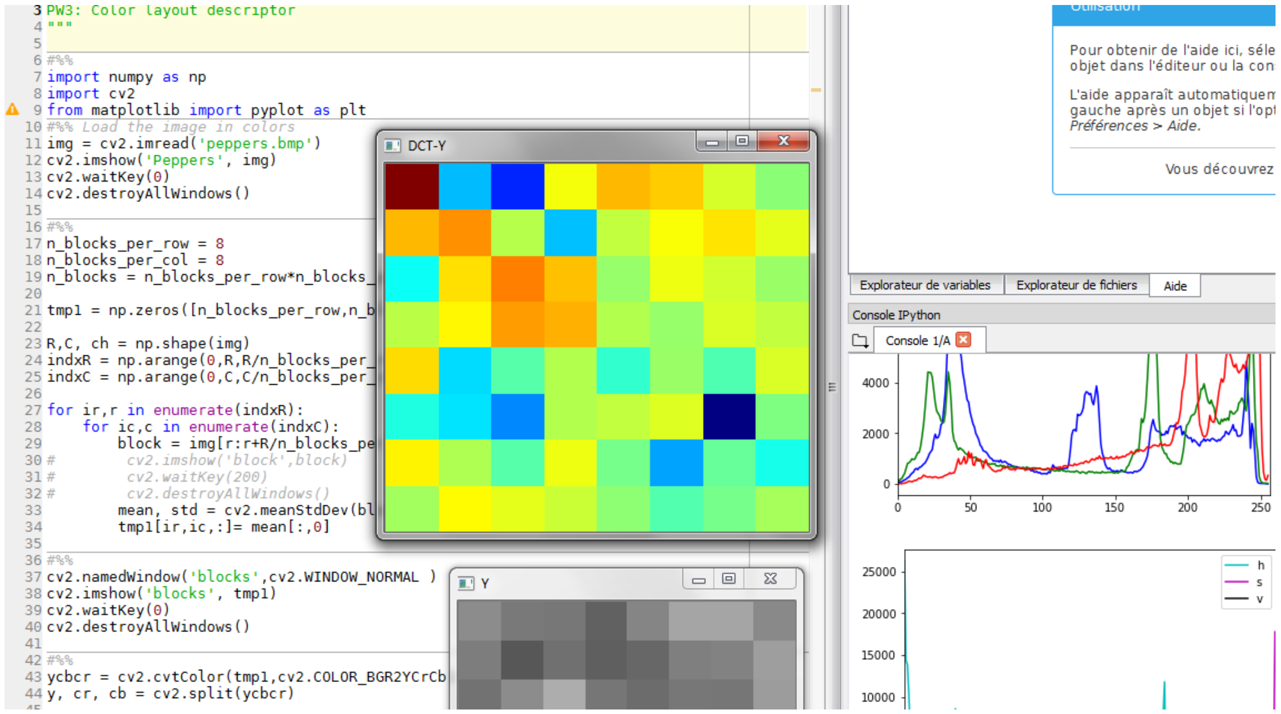
- Color content + its layout: *ColorLayout*
  - Input picture → 64 blocks → Matrix of color representatives
  - RGB → YCbCr color conversion
  - 8x8-DCT, zig-zag scanning and weighting → Keep 12 or 18 coefficients
  - *Low complexity image-to-image matching, sketch-to-image matching*



PW3

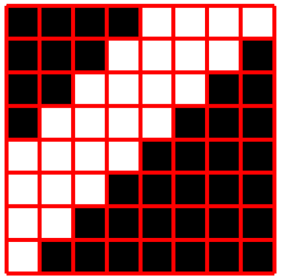
- Check out pw3.py and colorLayout.py
- Compute the colorLayout descriptor of several images and use it to compare them



```
PW3: Color layout descriptor
"""

#%%
import numpy as np
import cv2
from matplotlib import pyplot as plt
#%% Load the image in colors
img = cv2.imread('peppers.bmp')
cv2.imshow('Peppers', img)
cv2.waitKey(0)
cv2.destroyAllWindows()

#%%
n_blocks_per_row = 8
n_blocks_per_col = 8
n_blocks = n_blocks_per_row*n_blocks_

tmp1 = np.zeros([n_blocks_per_row,n_b

R,C, ch = np.shape(img)
indxR = np.arange(0,R,R/n_blocks_per_
indxC = np.arange(0,C,C/n_blocks_per_

for ir,r in enumerate(indxR):
    for ic,c in enumerate(indxC):
        block = img[r:r+R/n_blocks_pe
#        cv2.imshow('block',block)
#        cv2.waitKey(200)
#        cv2.destroyAllWindows()
        mean, std = cv2.meanStdDev(bl
        tmp1[ir,ic,:]= mean[:,0]

#%%
cv2.namedWindow('blocks',cv2.WINDOW_NORMAL )
cv2.imshow('blocks', tmp1)
cv2.waitKey(0)
cv2.destroyAllWindows()

#%%
ycbcr = cv2.cvtColor(tmp1,cv2.COLOR_BGR2YCrCb
y, cr, cb = cv2.split(ycbcr)
```

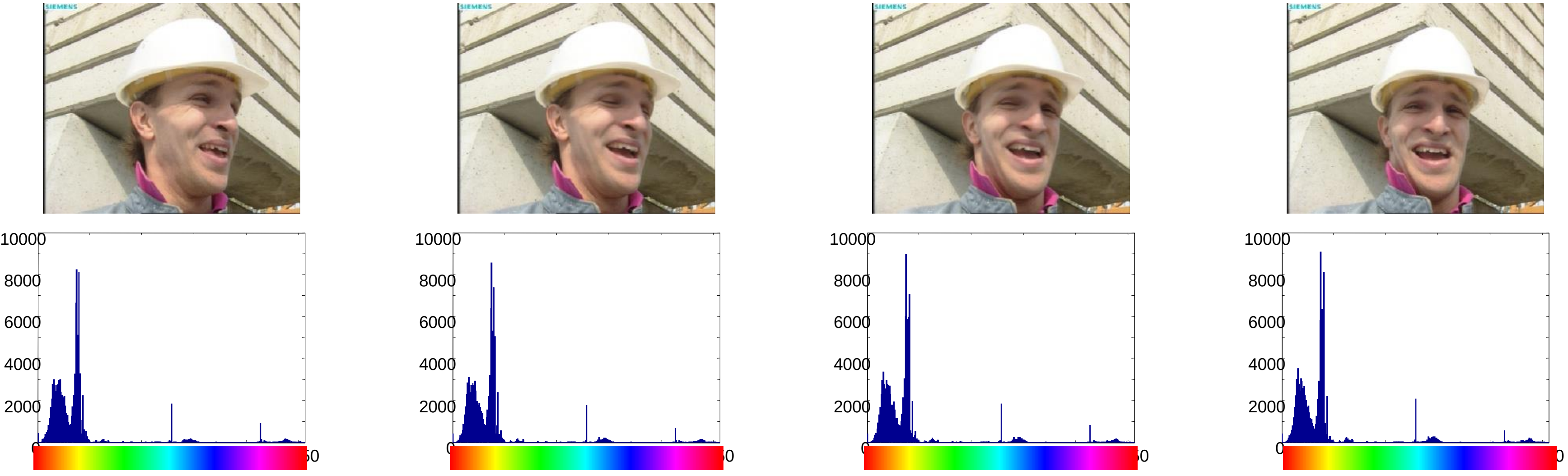
MPEG-7 standardized color descriptors (4/5)

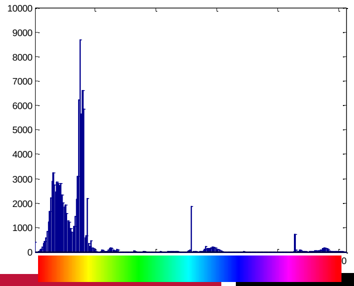
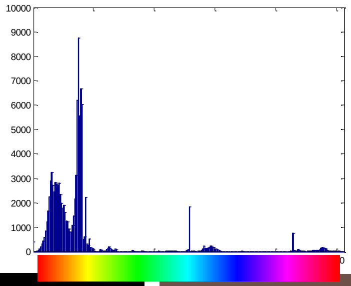
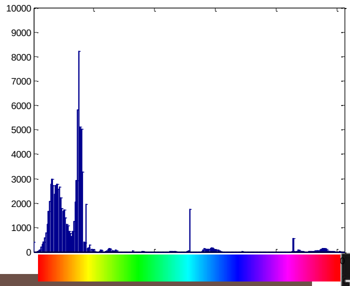
- Color content + coherence of repartition: *ColorStructure*
  - A 8x8 structure element scans the image (sliding window)
  - HMMD histogram of structure elements containing a particular color, quantized non-uniformly into 32, 64, 128 or 256 bins
  - *Enhanced still image retrieval*



MPEG-7 standardized color descriptors (5/5)

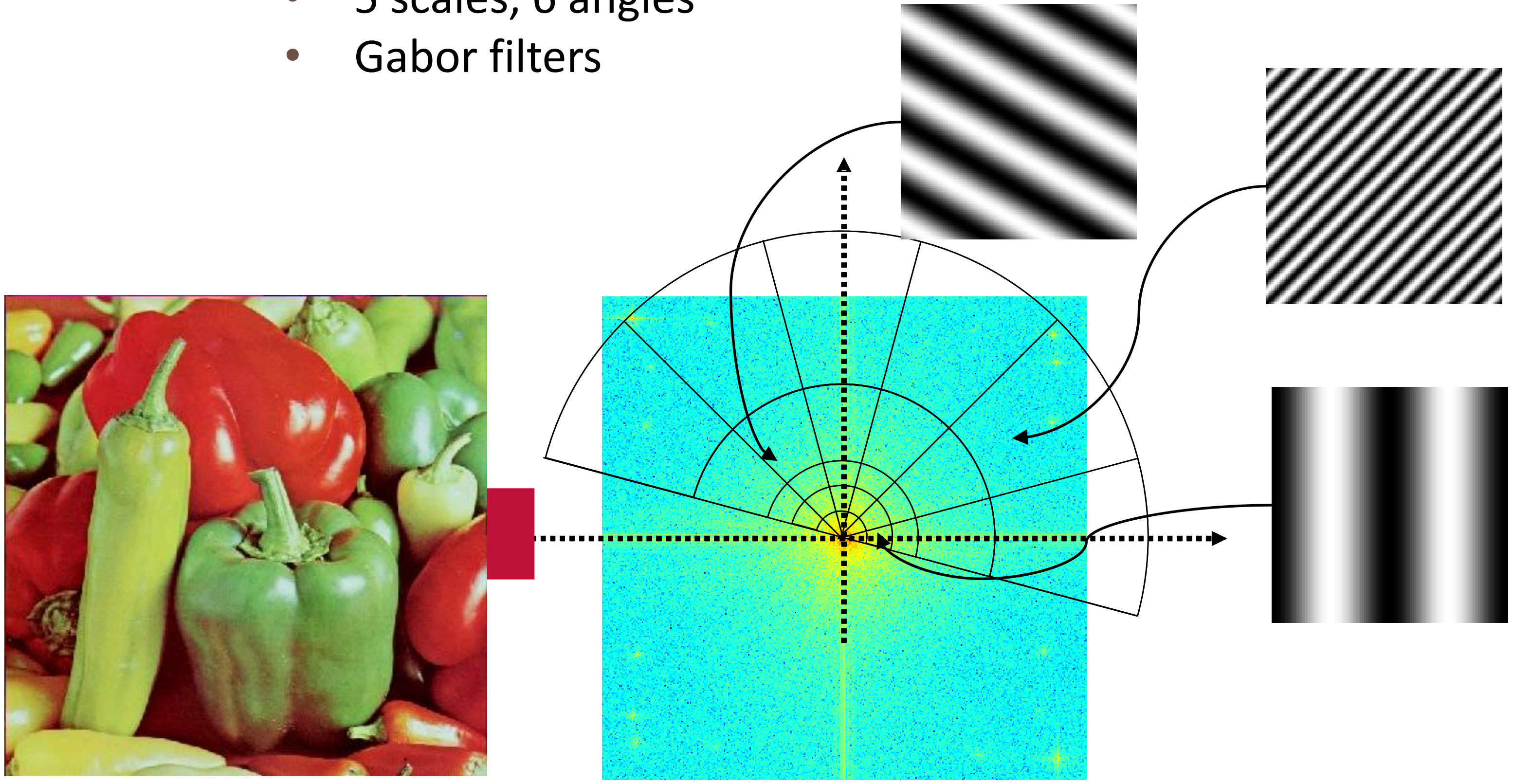
- Color content of Group of Pictures/Frames: *GoFGoPColor*
  - *ScalableColor* computed on average, median, or intersection of histograms
  - *GoF/GoP browsing / retrieval*



Texture (1/5)

- Characterization of homogeneous textures
  HomogeneousTexture
- Frequency plane divided into 30 channels
  - 5 scales, 6 angles
  - Gabor filters



Texture (2/5)

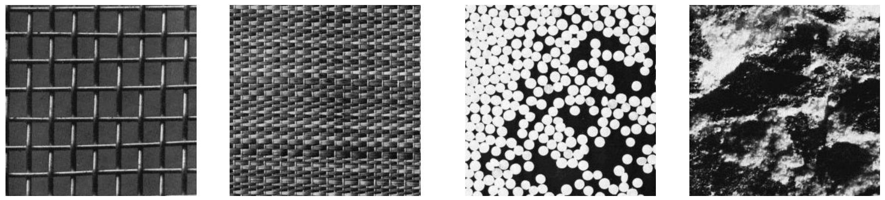
HomogeneousTexture Descriptor provides a precise quantitative description of a texture

- Computation based on filtering (directional filters)
- For each of the 30 channels
  - Energy
  - Energy standard deviation
- Image mean and STD
- Total of 62 parameters
  - Each quantified to 8 bits
- *Useful for retrieval in large collection of patterns*
  - *Similarity search/retrieval for aerial photos*

Texture (3/5)

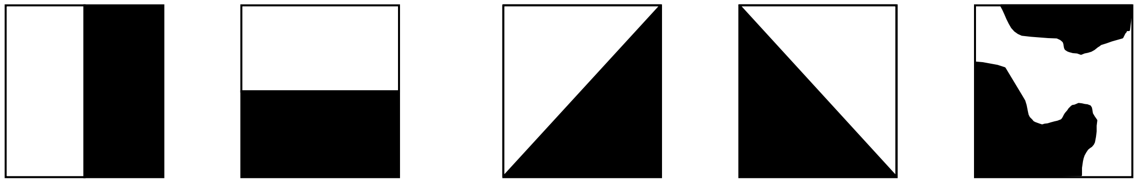
TextureBrowsing Descriptor useful for representing homogeneous textures for browsing applications

- Provides concise (12 bits) perceptual characterization
- Uses the 30 Gabor filters
- 2 dominant orientations are extracted (3 bits each)
- Regularity (2 bits)
- Coarseness (2x 2 bits)



Texture (4/5)

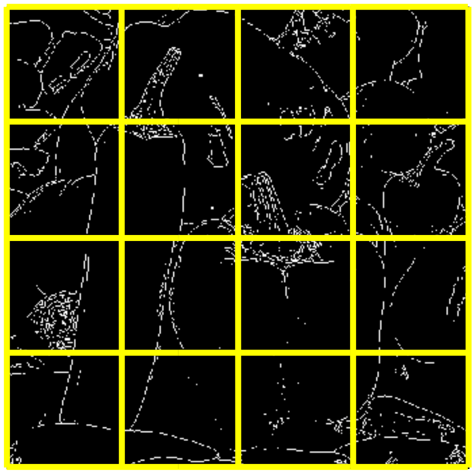
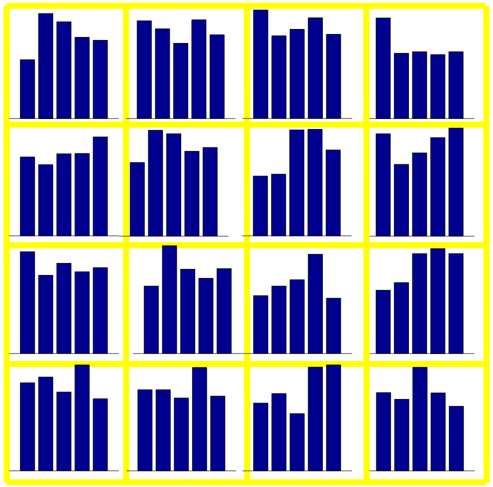
- Characterization of structures in generic images
- Edges content and layout: *EdgeHistogram Descriptor*
- Edge types histograms on 16 sub-images
- 5 kinds of edges
- Each bin quantized to 3 bits: 240 bits in total


Kinds of edges

Texture (5/5)

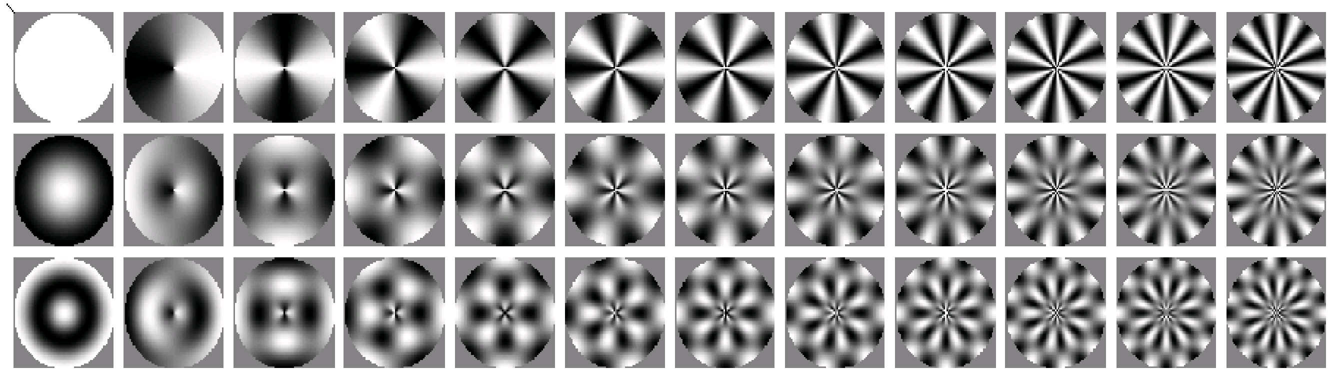
EdgeHistogram Descriptor

Image-to-image matching (by sketch)

Shape (1/6)

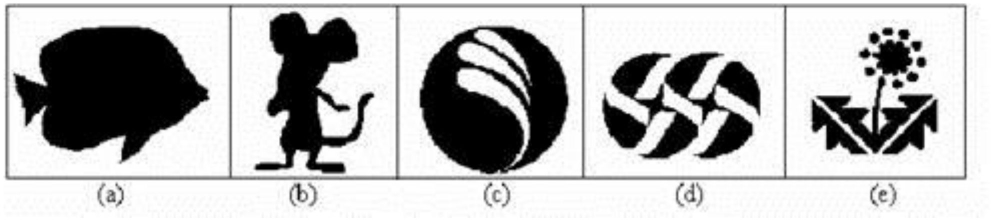
- Region Shape Descriptor *RegionShape*
  - Shape of objects, simply or manifoldly connected
  - Robust to boundary deformation
  - Computes 35 Angular-Radial Transform (ART) Coefficients
  - ART is the polar correspondant of 2D-DCT
  - *Useful for tracking shapes in video data processing*



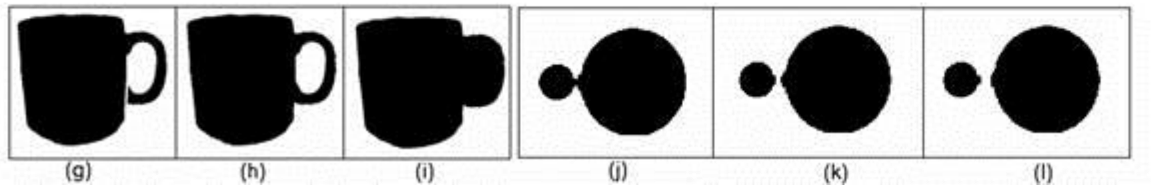
ART basis functions

Shape (2/6)

- Region Shape Descriptor *RegionShape*
- Examples of shapes that can be described:



- Similar shapes: (g) similar to (h), not to (i)
- (j), (k), (l) similar one to another



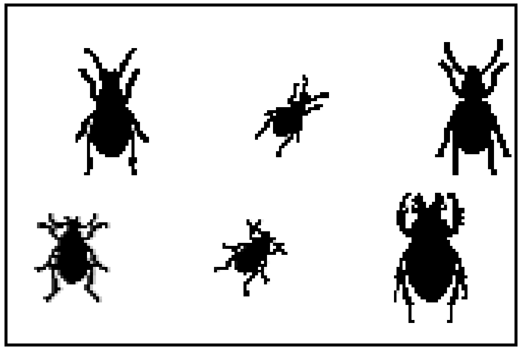
- Small size (17.5 bytes), fast extraction time and matching

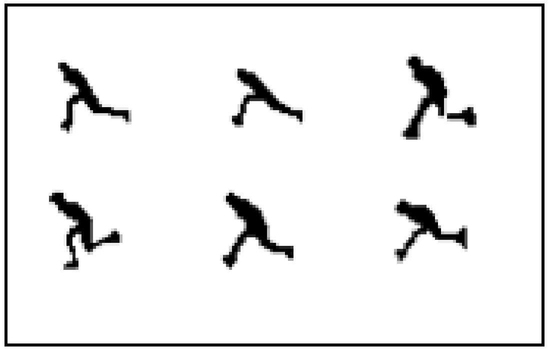
Shape (3/6)

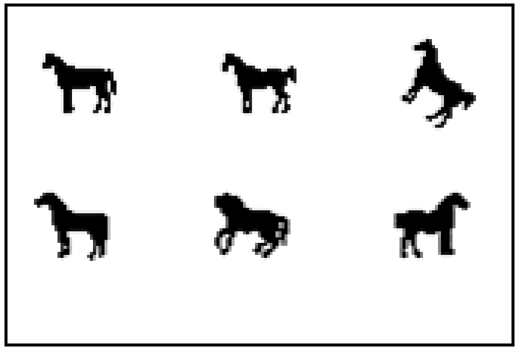
- Contour based shape descriptor *ContourShape*
  - Captures shape features
  - Robust to: non-rigid motion, partial occlusion, perspective transformations
  - Based on Curvature Scale-Space representation of the contour
    - Contour decomposed into convex and concave parts
    - Length, prominence and position of these parts are evaluated
  - *Enables similarity-based retrieval*

Shape (4/6)

- Contour based shape descriptor *ContourShape*



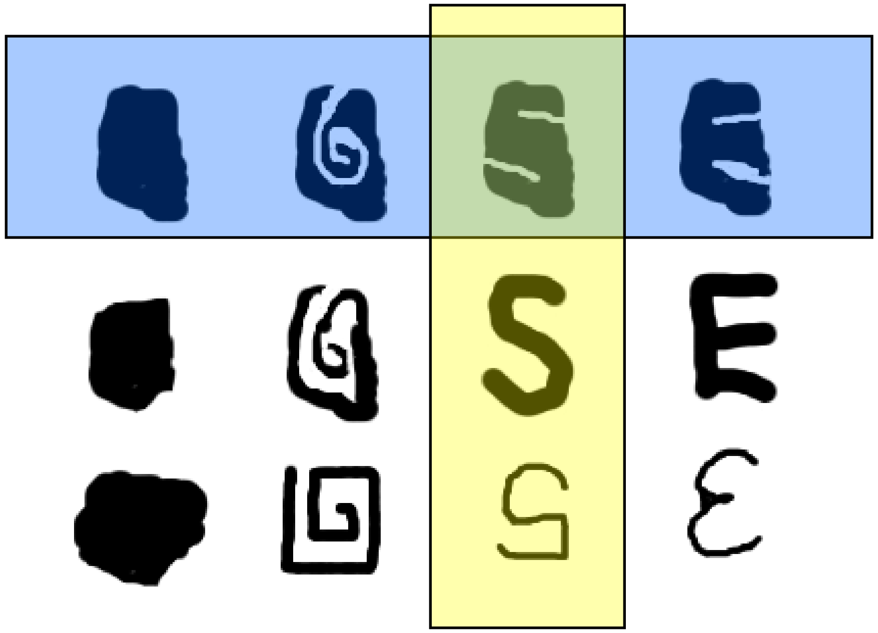




1) shape generalization properties (perceptual similarity among different shapes),
2) robustness to non-rigid motion (man running),
3) robustness to partial occlusion (tails or legs of the horses) and due to perspective transformations, which are common in the images and video

Shape (5/6)

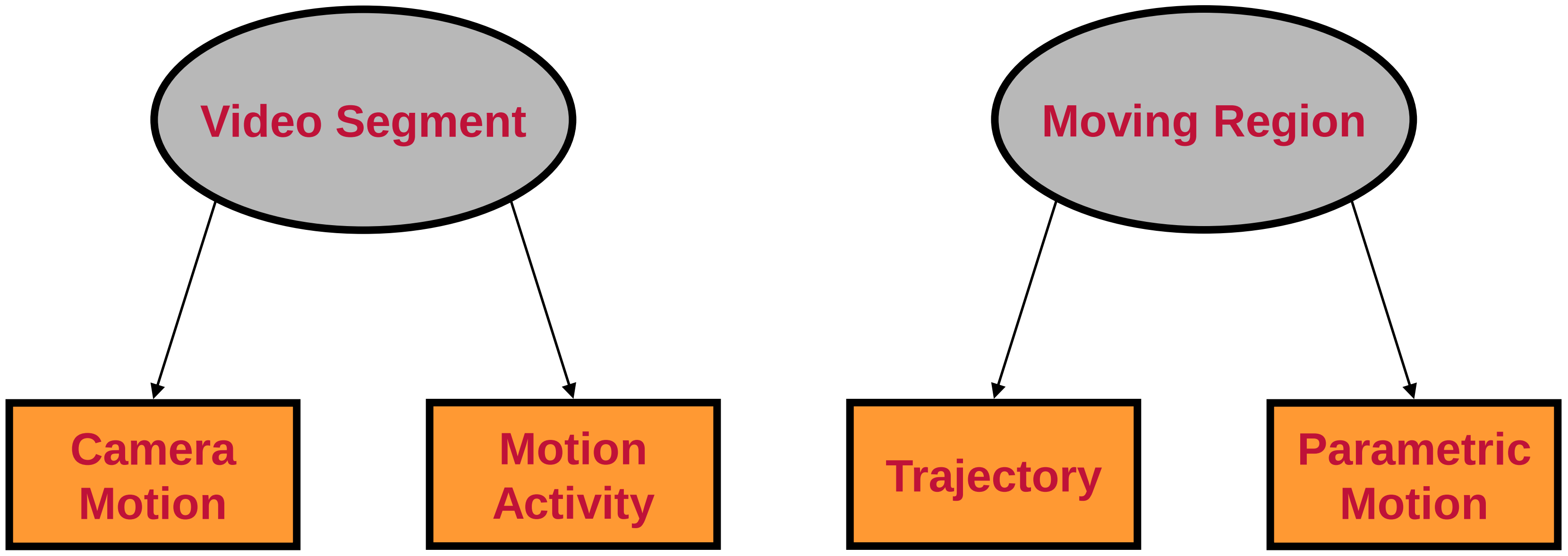
- Comparison *RegionShape* vs *ContourShape*
- Blue: Similar shapes by Region-Based
- Yellow: Similar shapes by Contour-Based



Shape (6/6)

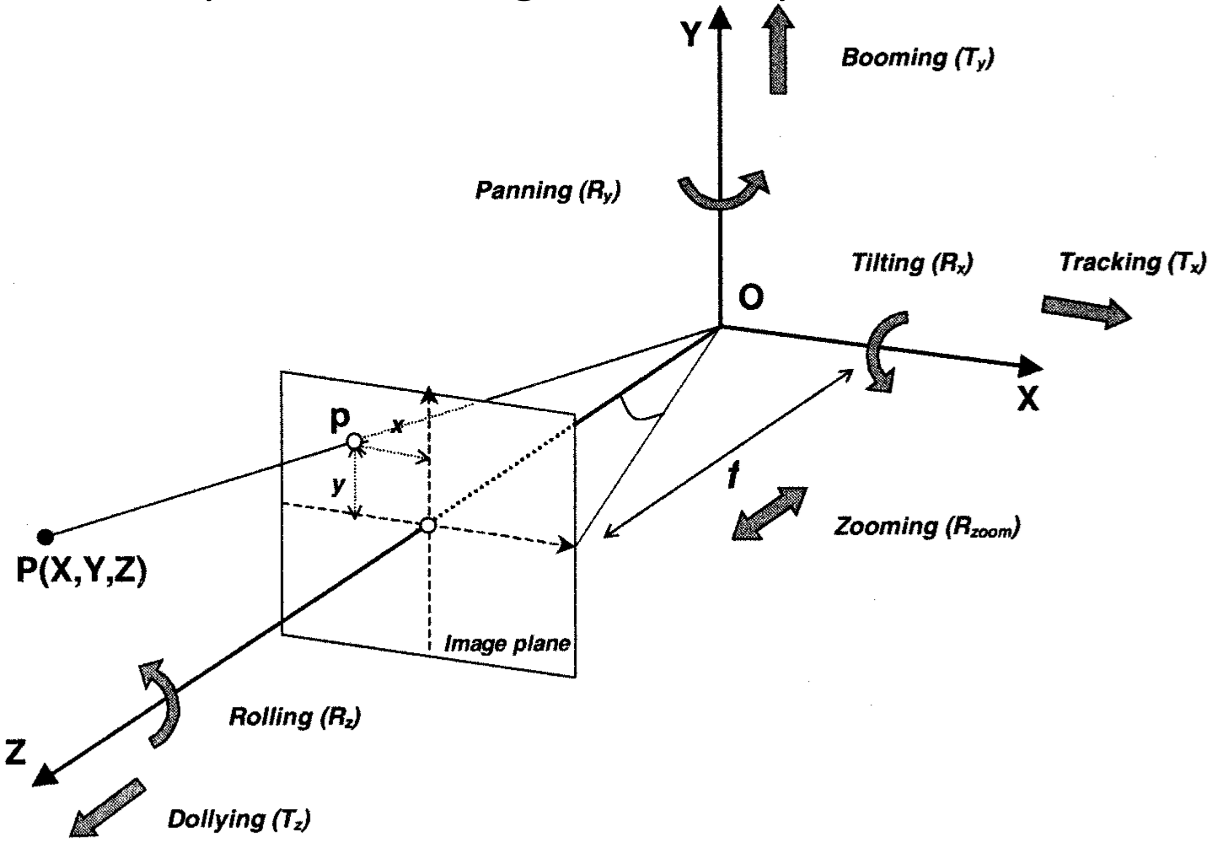
- Three-dimensional shape descriptor *Shape3D*
  - Based on 3D meshes
  - Histogram of features representing local curvature properties of the 3D surface
  - 48 bits giving similarity measure
  - *Search and retrieval of 3D mesh models*

Motion (1/6)


Video Segment
Moving Region
Camera Motion
Motion Activity
Trajectory
Parametric Motion

Motion (2/6)

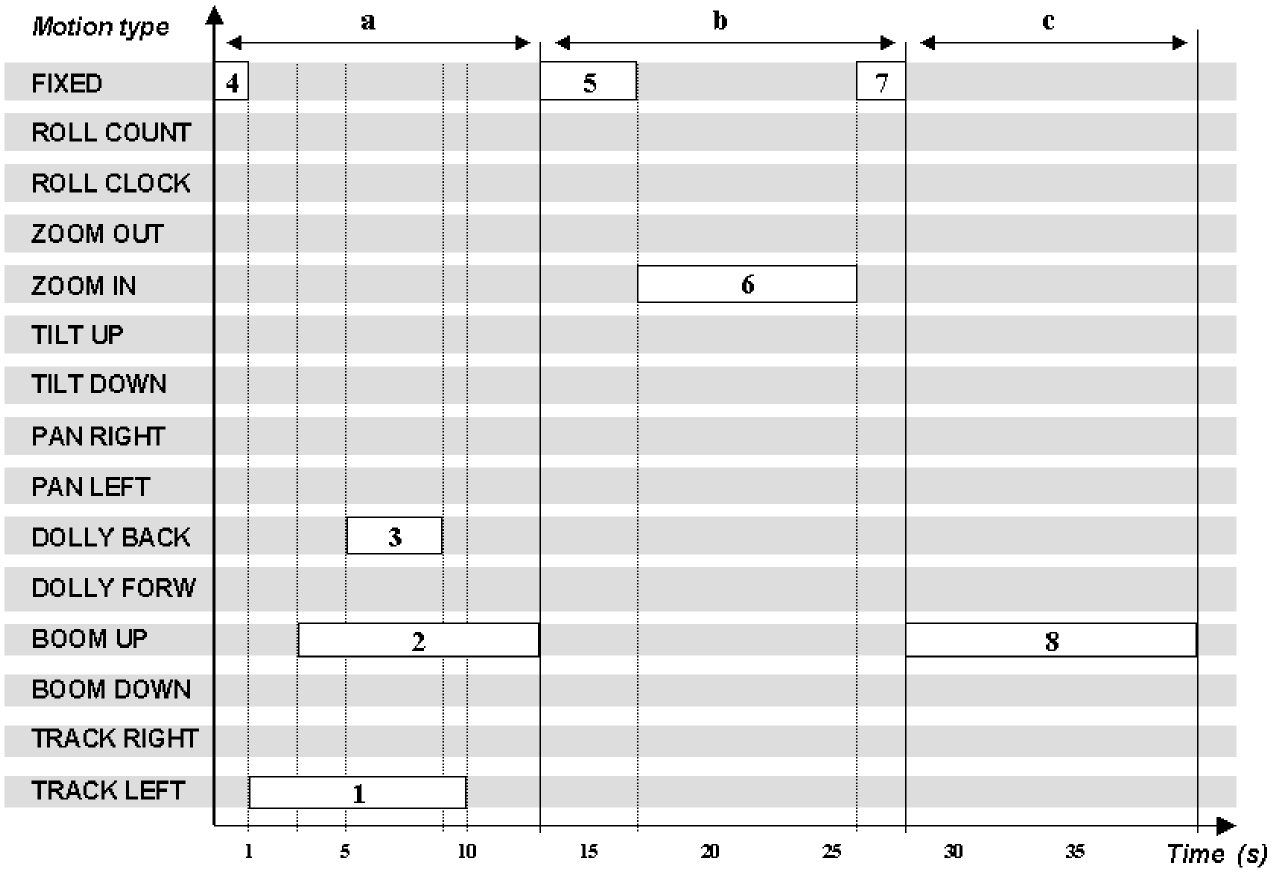
- Camera Motion *CameraMotion*
- 3D camera motion parameters
- can be automatically extracted or generated by the device



Motion (3/6)

CameraMotion

- Building blocks (BB) of the descriptor
  - Start time, duration, speed
  - A descriptor is made up of several BB
  - Mixture mode and non-mixture mode
- *Browsing, high level queries*

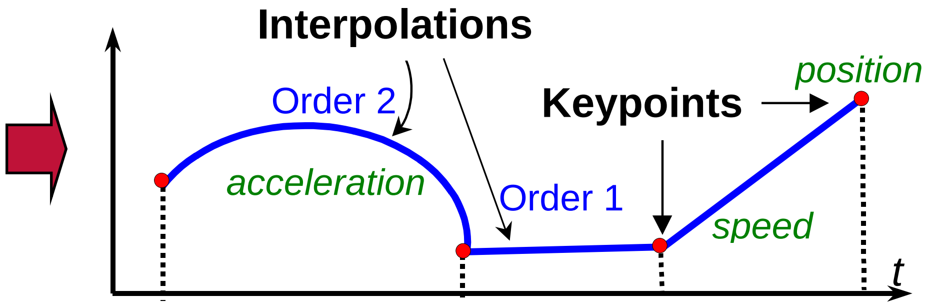


Motion (4/6)

- Motion Activity Descriptor *MotionActivity*
- Captures the notion of intensity of action in a video
  - Slow sequence, fast paced sequence, action sequence
- Attributes
  - Intensity of activity: std of motion vector magnitude (5 bins)
  - Direction of activity (optional)
  - Spatial distribution (optional)
  - Temporal distribution (optional)
- *Video browsing, content-based querying, surveillance, …*

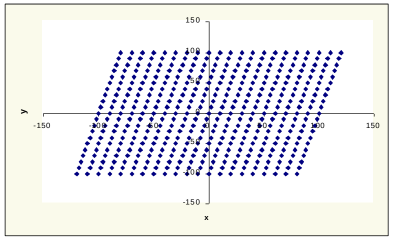
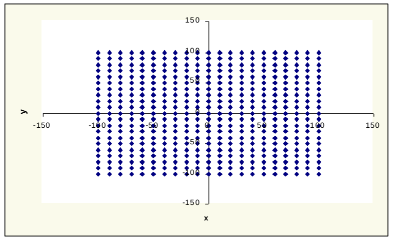
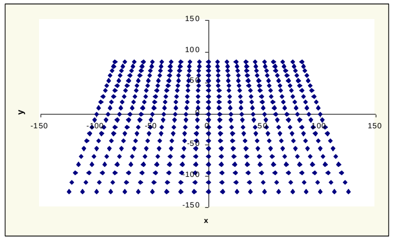
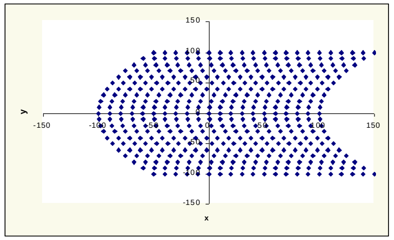
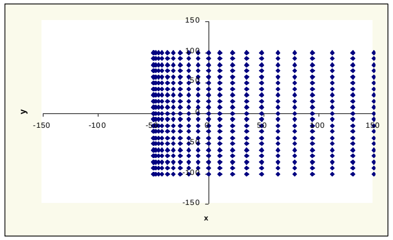
Motion (5/6)

- Motion Trajectory Descriptor *MotionTrajectory*
  - Simple, high level feature: space/time localization of a point
  - It consists in a list of key-points (space and time) and a set of (optional) interpolation functions
  - Motion parameters (speed, acceleration) are easily obtained
  - Independent of space/time resolution, compact and scalable
  - *Similarity retrieval, high level queries*
  - *Surveillance (dangerous trajectory or speed), sport (specific actions revelation)*



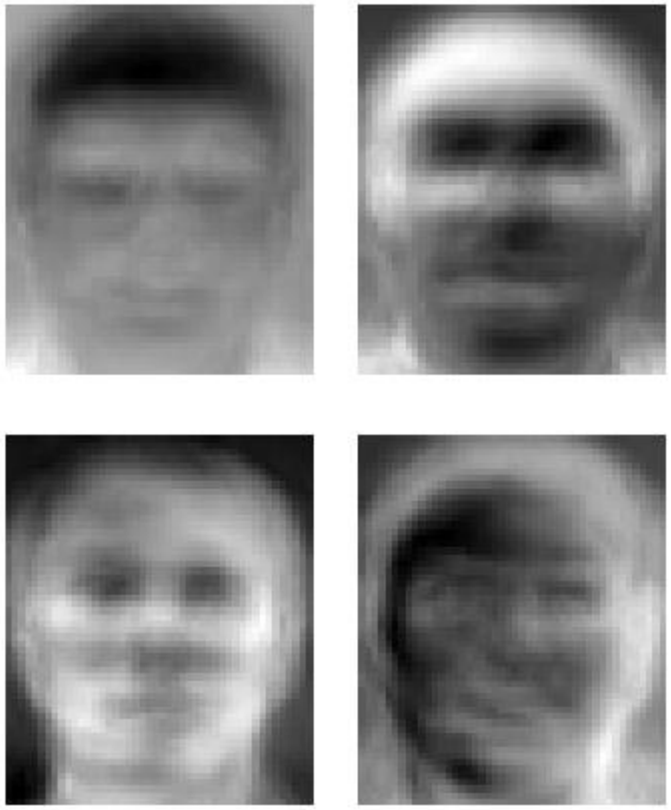
Motion (6/6)

- Parametric Motion Descriptor *ParametricMotion*
  - Describes object motion in video as a 2D parametric model
    - translational; rotation/scaling; affine; perspective; quadratic
  - Parameter extraction: a typical inverse problem
    - Regularization and optimization
  - Associated with arbitrary objects (group of pixels)
  - Similarity measure between motion descriptions
  - *Similarity retrieval, high level queries*

Face

- Human Face Descriptor *FaceRecognition*
  - Face descriptor based on Principal Component Analysis
  - Normalized face image, 56x46 pel, eyes in (24,16), (24,31)
  - Raster scan to a 2576-elements vector
  - Projection onto a set of 48 basis vectors
    - Eigenvectors of a face vector training set: *eigenfaces*
  - Robust to view-angle and illumination changes
  - *Database retrieval, user authentication, facility access, video surveillance*


Example of eigenfaces

MPEG-7 Visual Amendment 1 (2004)

- *GofGopFeature*
  - Generic and extensible container to use several description tools to describe the representative feature over GoFGoP
- *ColorTemperature*
  - Perceptual temperature feeling of illumination color
  - Intended for browsing and display preference control purposes
  - Four perceptual temperature categories are provided; hot, warm, moderate, and cool.
- *IlluminationInvariantColor*
  - This descriptor wraps the color descriptors *DominantColor*, *ScalableColor*, *ColorLayout*, and *ColorStructure*.
  - One or more color descriptors processed by the illumination invariant method can be included in this descriptor

MPEG-7 Visual Amendment 1 (2004)

- *ShapeVariation*:
  - Describes shape variations in terms of Shape Variation Map and the statistics of the region shape description of each binary shape image in the collection.
  - ***StaticShapeVariation, DynamicShapeVariation***
    - 35 quantized ART coefficients on a 2-dimensional histogram of group of shape images, with or without the background
  - For the statistics, a set of standard deviations of 35 coefficients of the *RegionShape* are used

- *AdvancedFaceRecognition*:
  - This descriptor of face identity is robust to variations in pose and illumination conditions

MPEG-7 Visual Amendment 1 (2004)

Media-centric description schemes: three visual *description schemes* are designed to describe several types of visual contents

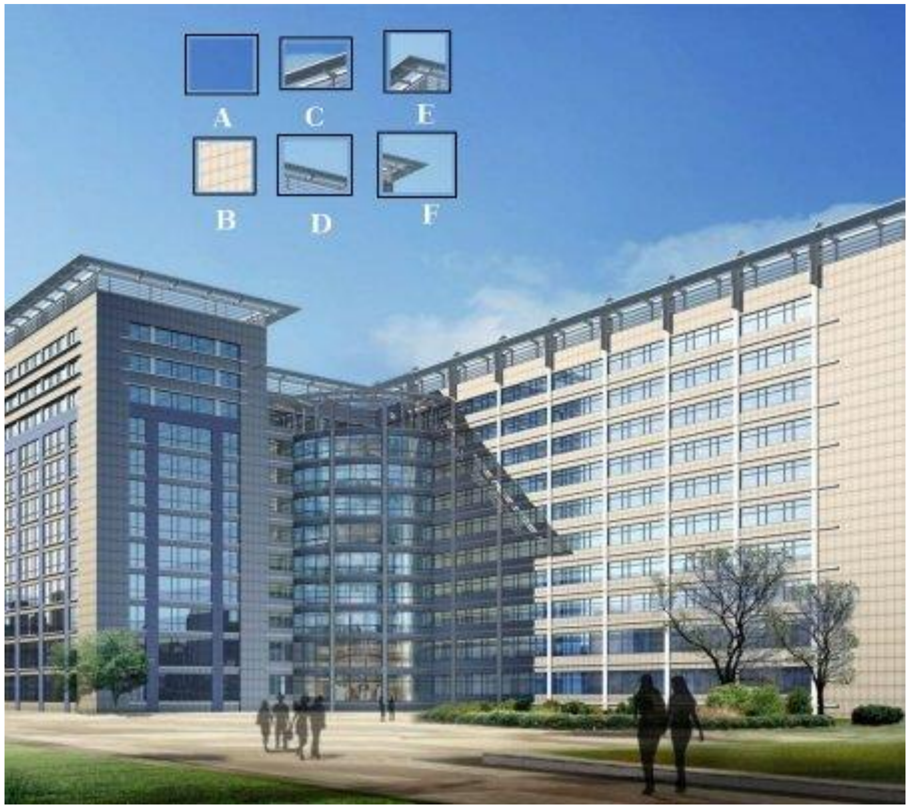
- ***StillRegionFeatureType***
  - contains several elementary descriptors to describe the characteristics of arbitrary shaped still regions
- ***VideoSegmentFeatureType***
  - supports ordinary video without shape
- ***MovingRegionFeatureType***
  - Supports arbitrary shaped video sequences

MPEG-7 Visual Amendments 2, 3, and 4

- MPEG-7 Visual Amendment 2
  - Published 2006, Corrigendum 2007
  - New visual descriptors providing additional functionalities
  - 3D perceptual shape descriptor
- MPEG-7 Visual Amendment 3 (2009)
  - Image signature tool
- MPEG-7 Visual Amendment 4
  - Video signature tool (2010)

Harris corner detection

- Corners are unambiguous features



Harris corner detection

- Compute, in the current window, the autocorrelation surface as a function of the displacement, $E(\Delta \boldsymbol{u})$
- Using Taylor expansion, $E(\Delta \boldsymbol{u}) \approx \Delta \boldsymbol{u}^T \boldsymbol{A} \Delta \boldsymbol{u}$ where

$$\boldsymbol{A} = \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

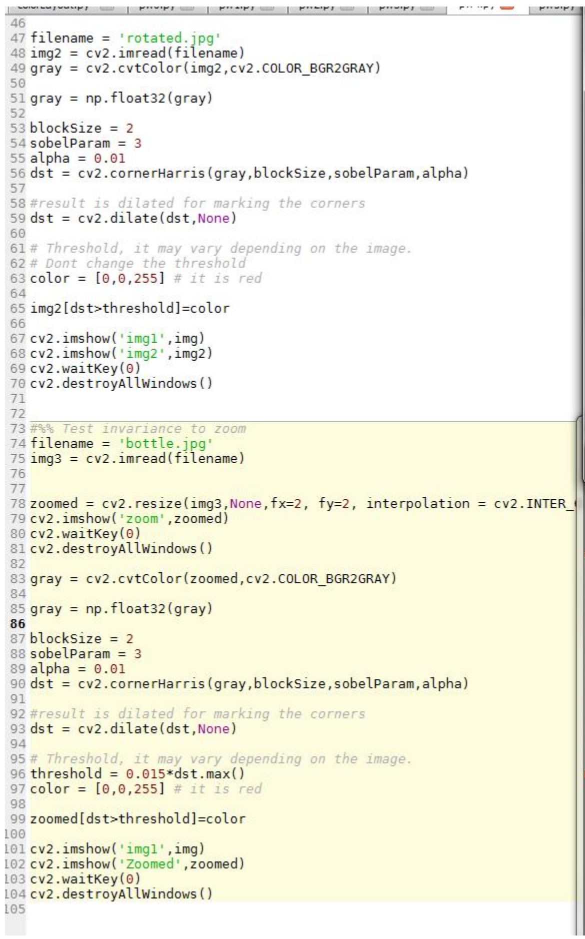
- The 2 eigenvectors of $\boldsymbol{A}$ are the directions of fastest and slowest changes in the current window
- The eigenvalues λ_0, λ_1 are the intensity of the changes
- There is a corner when the smallest eigenvalue is large enough:

$$\lambda_0 \geq T_1$$
$$\lambda_0 \lambda_1 - \alpha(\lambda_0 + \lambda_1)^2 \geq T_2$$

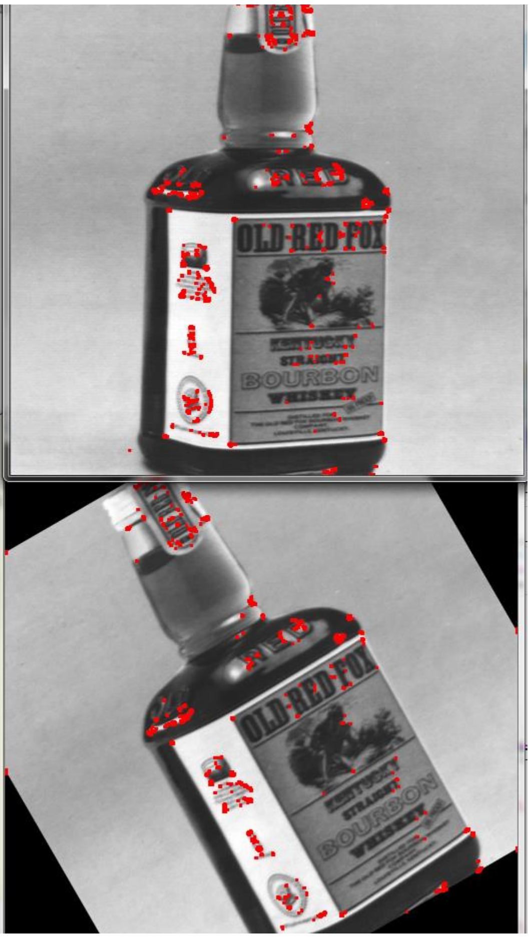
The first is the Tomasi-Shi detector; the second is the Harris detector

PW4

- Use pw4 to run the Harris corner detector
- Change the parameters and the image
- Test rotation and scale invariance



```

filename = 'rotated.jpg'
img2 = cv2.imread(filename)
gray = cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)

gray = np.float32(gray)

blockSize = 2
sobelParam = 3
alpha = 0.01
dst = cv2.cornerHarris(gray,blockSize,sobelParam,alpha)

#result is dilated for marking the corners
dst = cv2.dilate(dst,None)

# Threshold, it may vary depending on the image.
# Dont change the threshold
color = [0,0,255] # it is red

img2[dst>threshold]=color

cv2.imshow('img1',img)
cv2.imshow('img2',img2)
cv2.waitKey(0)
cv2.destroyAllWindows()


#%% Test invariance to zoom
filename = 'bottle.jpg'
img3 = cv2.imread(filename)


zoomed = cv2.resize(img3,None,fx=2, fy=2, interpolation = cv2.INTER_
cv2.imshow('zoom',zoomed)
cv2.waitKey(0)
cv2.destroyAllWindows()

gray = cv2.cvtColor(zoomed,cv2.COLOR_BGR2GRAY)

gray = np.float32(gray)

blockSize = 2
sobelParam = 3
alpha = 0.01
dst = cv2.cornerHarris(gray,blockSize,sobelParam,alpha)

#result is dilated for marking the corners
dst = cv2.dilate(dst,None)

# Threshold, it may vary depending on the image.
threshold = 0.015*dst.max()
color = [0,0,255] # it is red

zoomed[dst>threshold]=color

cv2.imshow('img1',img)
cv2.imshow('Zoomed',zoomed)
cv2.waitKey(0)
cv2.destroyAllWindows()

```



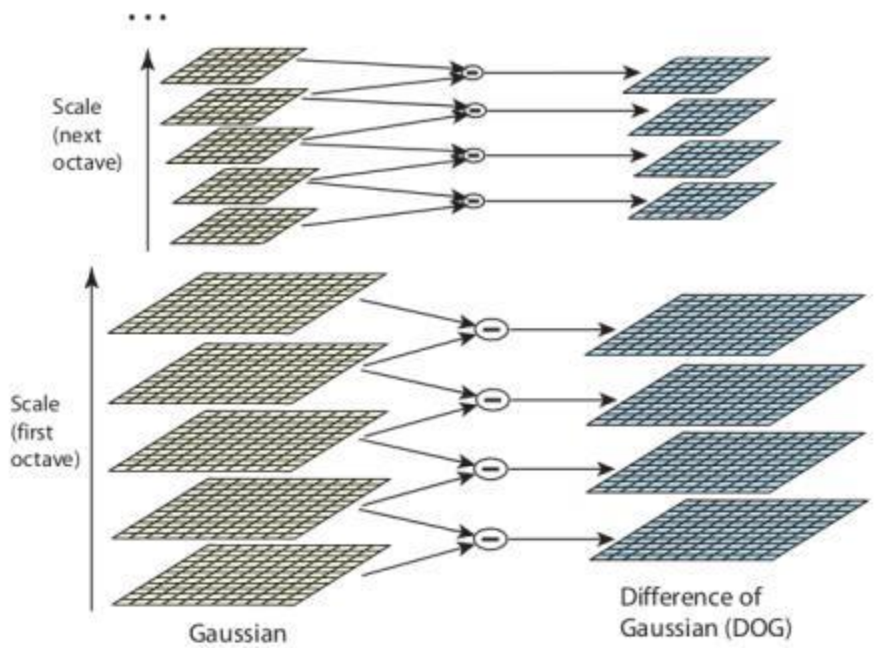
SIFT

- Scale invariance can be difficult to achieve with Harris detector

SIFT

1. Difference of Gaussians

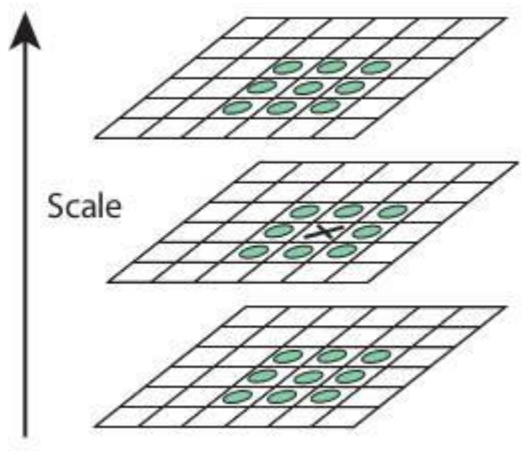
Difference of Gaussian is obtained as the difference of Gaussian blurring of an image with two different variances, let it be σ and $k\sigma$. This process is done for different octaves of the image in Gaussian Pyramid



SIFT

2. Local extrema over space and scale

One pixel is compared with its 8 neighbours as well as 9 pixels in next scale and 9 pixels in previous scales. If it is a local extrema, it is a potential keypoint. It means that keypoint is best represented in that scale.



SIFT

3. Keypoint localization

First, low local extrema are removed

Strong local extrema may be corners or edges

Using concepts from HCD, if the ratio of the ACM eigenvalues is larger than a threshold, the keypoint is removed

SIFT

4. Orientation

A neigbourhood is taken around the keypoint location depending on the scale,

The gradient magnitude and direction is calculated in that region.

The orientation histogram with 36 bins covering 360 degrees is created, weighted by gradient magnitude

The highest peak in the histogram is taken and any peak above 80% of it is also considered to calculate the orientation.

SIFT

5. Keypoint Descriptor

A 16x16 neighbourhood around the keypoint is taken.

It is devided into 16 sub-blocks of 4x4 size.

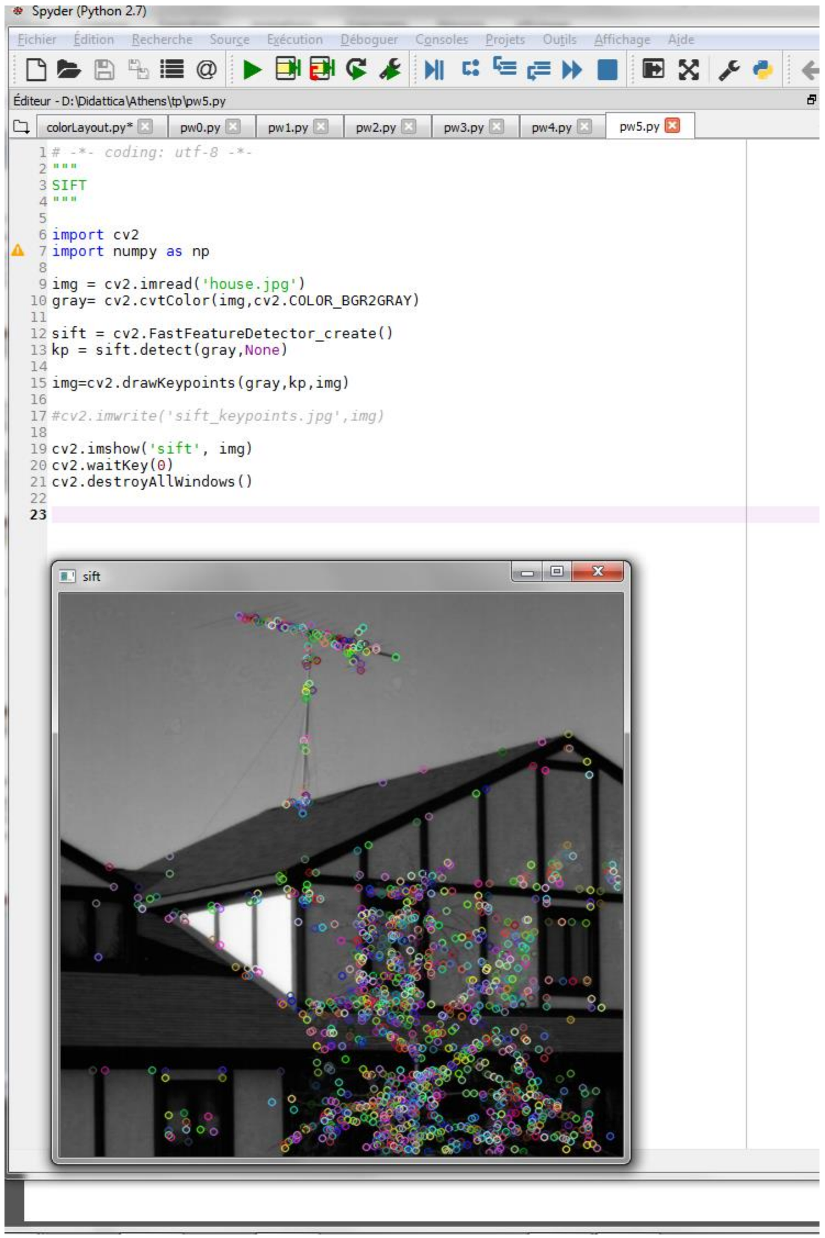
For each sub-block, 8 bin orientation histogram is created. The descriptor is a vector of 128 elements

6. Keypoint Matching

Keypoints between two images are matched by identifying their nearest neighbours. But in some cases, the second closest-match may be very near to the first. It may happen due to noise or some other reasons. In that case, ratio of closest-distance to second-closest distance is taken. If it is greater than 0.8, they are rejected. It eliminaters around 90% of false matches while discards only 5% correct matches.

PW: Sift

- Play with PW5 : rescale, shift, rotate…


Spyder (Python 2.7)
Fichier Édition Recherche Source Exécution Déboguer Consoles Projets Outils Affichage Aide
Éditeur - D:\Didattica\Athens\tp\pw5.py
colorLayout.py* pw0.py pw1.py pw2.py pw3.py pw4.py pw5.py
sift

```
# -*- coding: utf-8 -*-
"""
SIFT
"""

import cv2
import numpy as np

img = cv2.imread('house.jpg')
gray= cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

sift = cv2.FastFeatureDetector_create()
kp = sift.detect(gray,None)

img=cv2.drawKeypoints(gray,kp,img)

#cv2.imwrite('sift_keypoints.jpg',img)

cv2.imshow('sift', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Bag of Words

- High level classification (category recognition)
- Bag-of-features approaches
  - Unordered collection of feature descriptors (visual words)
  - Compute the histogram of VWs on the query image
  - Compare this distributions with those found on the training images
  - VWs can be SIFT, Harris corners, other features