Coding Bootcamp Code in Python

MATPLOTLIB

And some matplotlib…

- Rich Python plotting library
  - scatter plot
  - line plot
  - bar plot/histogram
  - heatmap
  - 3D surface plot
- Highly customizable plots
  - LaTeX labels/annotation
- Plot to screen, various file formats

Lots of features, this barely scratches the surface.

Convention: `import matplotlib.pyplot as plt`

Simple line plot

- Data: lists or numpy arrays

```
x = np.linspace(0.0, 20.0, 500)
y = np.exp(-mu*x)*np.cos(2.0*np.pi*x)
```

- Add plot

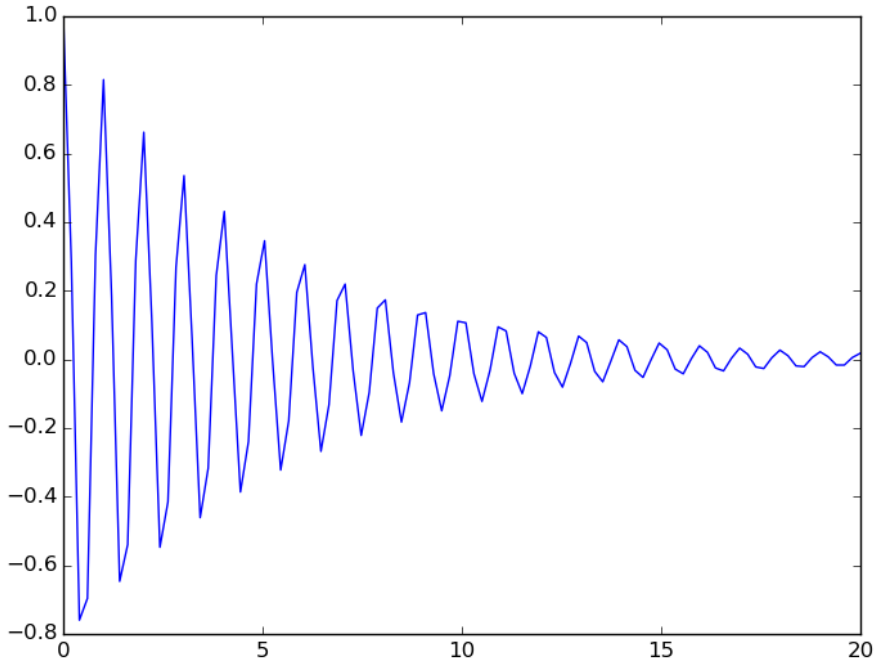
```
plt.plot(x, y)
```

- Show plot

```
plt.show()
```

- Save plot

```
plt.savefig(file_name)
```



Axis labels, annotation

- Add label for *x* and *y* axis

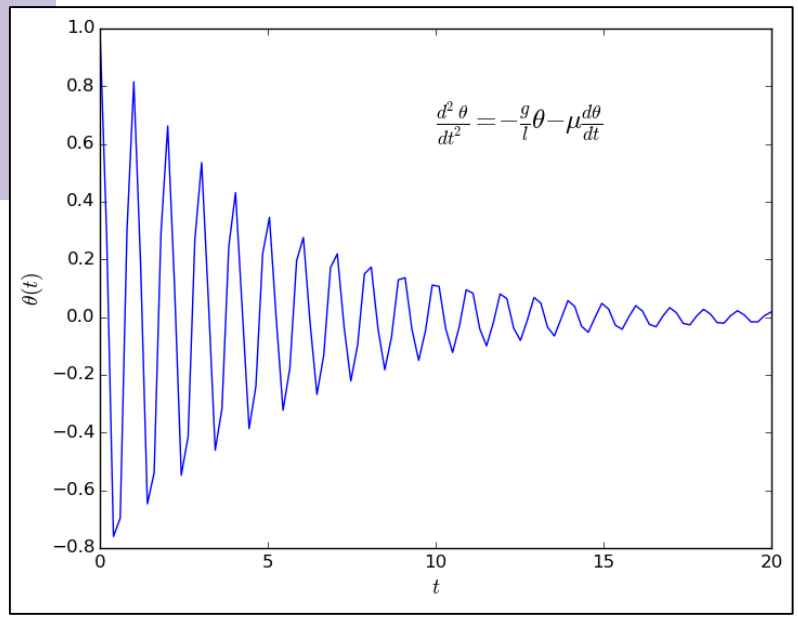
```
plt.xlabel(r'$t$', fontsize=14)
plt.ylabel(r'$\theta(t)$', fontsize=14)
```

- Add annotation

```
eq = (r'$\frac{d^2 \theta}{dt^2} = ' +
      r'- \frac{g}{l} \theta ' +
      r'- \mu \frac{d\theta}{dt}$')
plt.text(10.0, 0.65, eq, fontsize=18)
```

construct plot
≡
gradually enrich `plt`

LaTeX notation, rendered



Multiple functions on line plot

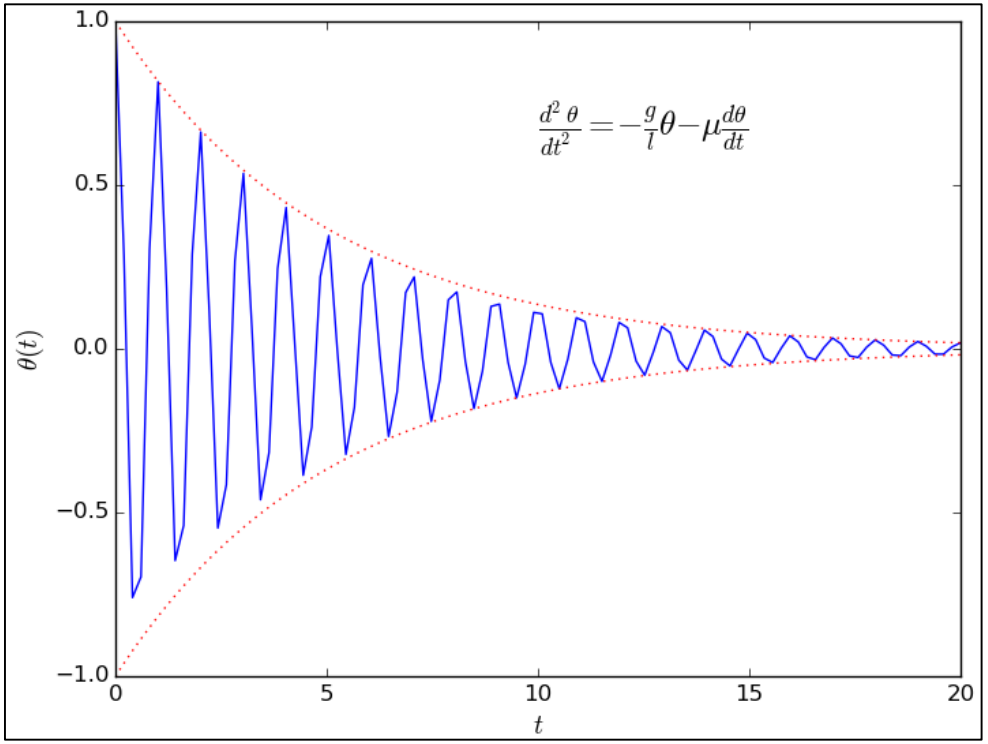
- Data

```
y_plus = np.exp(-mu*x)
y_min = -np.exp(-mu*x)
```

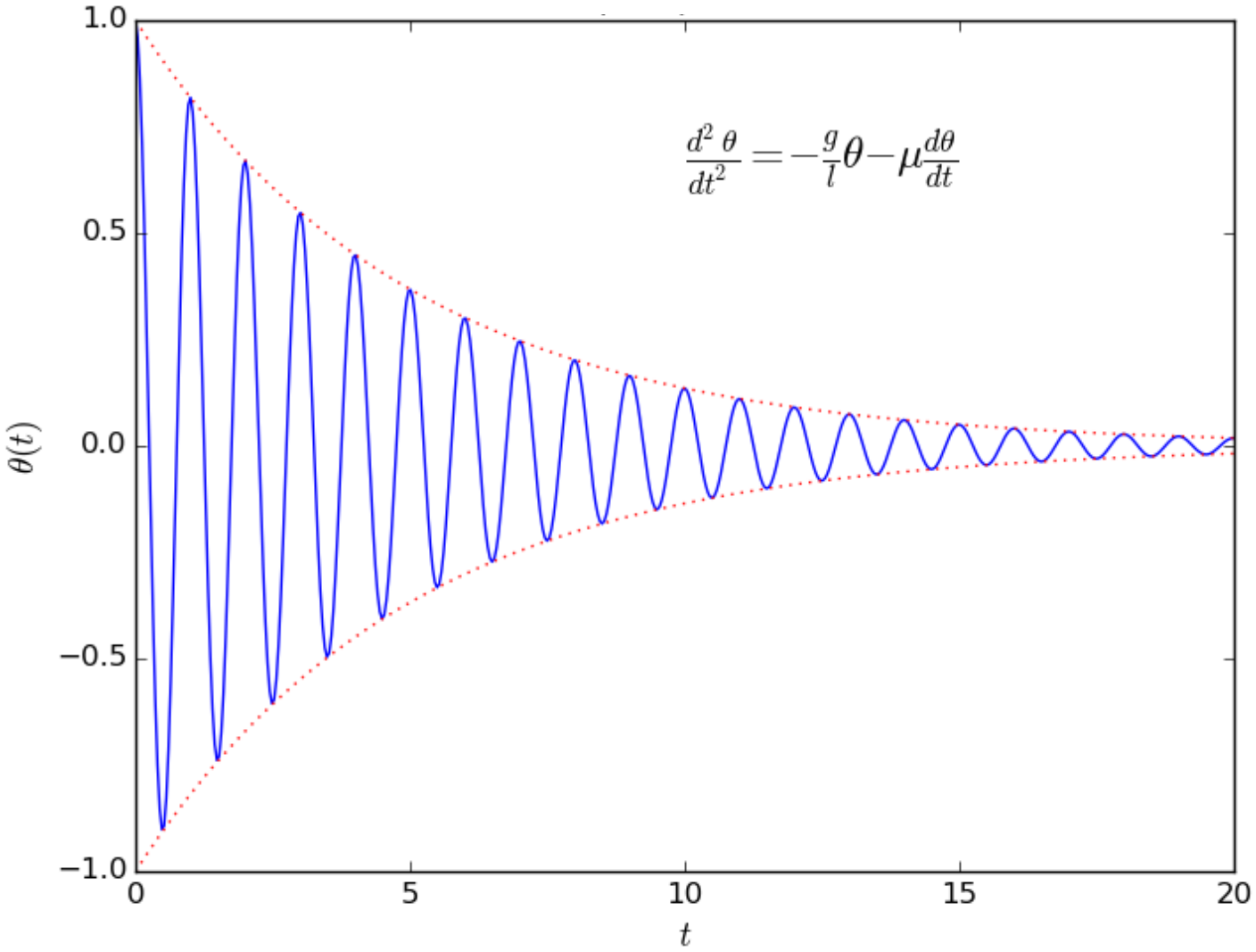
- Add to plot, line style, color

```
plt.plot(x, y_plus,
         ':', alpha=0.8,
         color='red',
         linewidth=0.9)
plt.plot(x, y_min,
         ':', alpha=0.8,
         color='red',
         linewidth=0.9)
```

line type



Complete line plot



Histogram

- Data:

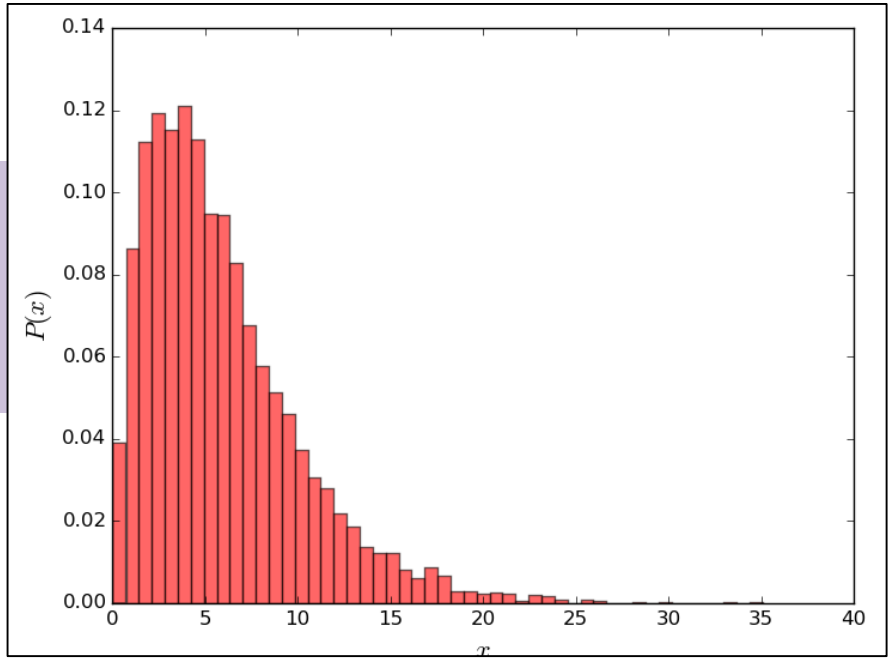
```
values = np.loadtxt('data.txt')
bins = 50
```

```
13.3146868758
9.66243828428
6.44936354431
2.81183151337
9.55644862073
…
```

data.txt

- Plot histogram

```
plt.hist(values, bins, normed=1,
         color='red', alpha=0.6)
plt.xlabel('$x$', fontsize=16)
plt.ylabel('$P(x)$', fontsize=16)
```



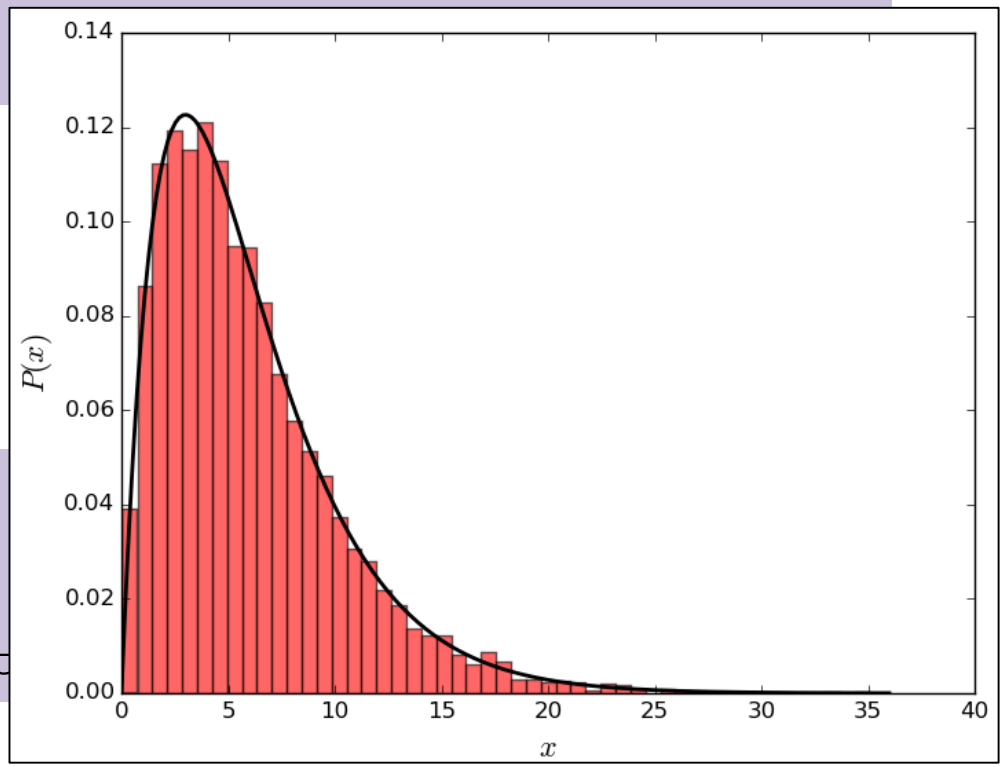
Line plot on histogram

- Data:

```
x = np.linspace(floor(np.min(values)), ceil(np.max(values)),
                200)
y = sp.stats.gamma.pdf(x, 2, scale=3.0)
plt.plot(x, y, linewidth=2.0,
         color='black')
```

- Reminder

```
import numpy as np
import scipy as sp
import scipy.stats
import matploblib.pyplot as plt
```



Heat map data

- Function to plot

```
def f(x, y, x0=0.0, freq=1.0, beta=0.5):
    r = np.sqrt((x - x0)**2 + y**2)
    return np.exp(-beta*r)*np.cos(2.0*np.pi*freq*r)
```

- Data

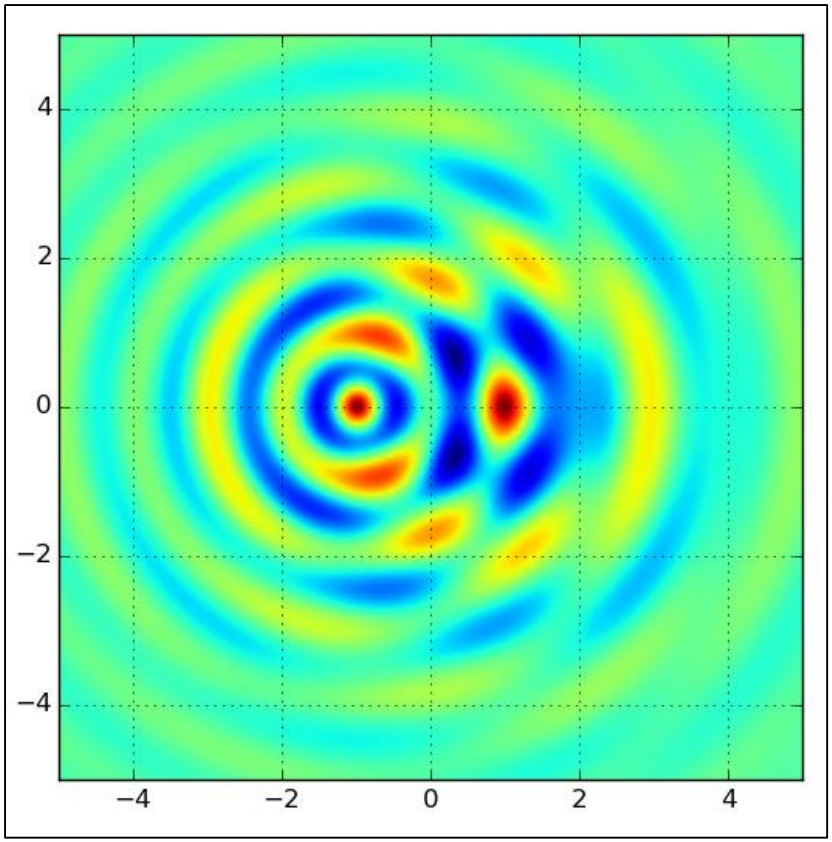
```
x = np.linspace(x_min, x_max, points)
y = np.linspace(y_min, y_max, points)
X, Y = np.meshgrid(x, y)
z = f(X, Y, x0=x0_1, freq=f_1) + f(X, Y, x0=x0_2, freq=f_2)
```

Heat map plot

- Plot map

```
plt.imshow(z, extent=[x_min, x_max, y_min, y_max])
plt.grid(True)
```

Explore color maps, helps interpret data! Brewer schemes



3D surface plot I

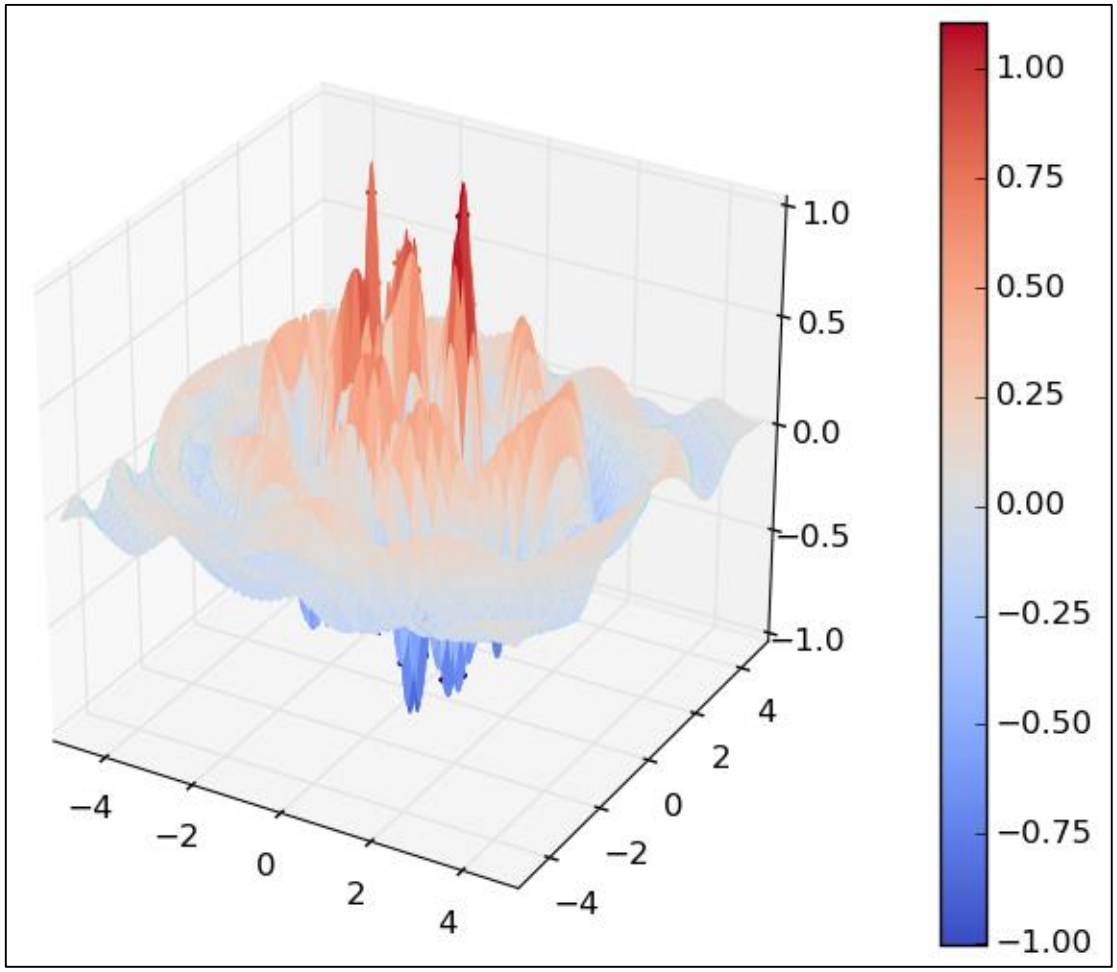
- Importing extra modules

```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
```

- Plot

```
figure = plt.figure()
axes = figure.gca(projection='3d')
axes.set_xlim(x_min, x_max)
axes.set_ylim(y_min, y_max)
axes.set_zlim(z_min, z_max)
surface = axes.plot_surface(X, Y, z,
                            rstride=4, cstride=4,
                            cmap=cm.coolwarm,
                            linewidth=0)
figure.colorbar(surface)
```

3D surface plot II



References

- ColorBrewer 2.0: advice on choosing appropriate color maps
  http://colorbrewer2.org/
- Overview of data visualization types & libraries for Python
  https://python-graph-gallery.com/

Code Pack 27

A. Matplotlib:

1-Figures_Subplots_and_layouts

2-Plotting_Methods_Overview

3-HowToSpeakMPL

4-Limits_Legends_and_Layouts

5-Artists

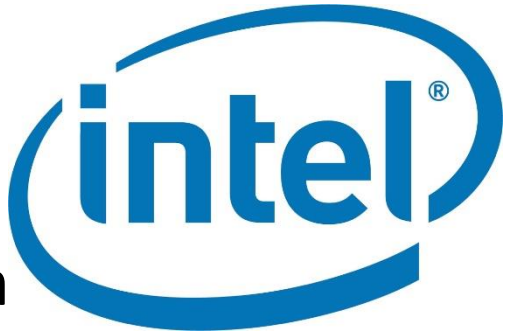
6-mpl_toolkits

Coding Bootcamp Code in Python

INTRODUCTION TO OPENCV

OpenCV

- OpenCV is an Image Processing library created by Intel on 1999 and maintained by Willow Garage.
  - First Release was in 2000
- Available for C, C++, and Python
- Newest update is version 4.0
- Open Source and free (3-clause BSD Licen
- Easy to use and install



opencv-python

- Officially, OpenCV releases two types of Python interfaces,
- `cv` and `cv2`.

- `cv` (legacy): all OpenCV data types are preserved as such.
  - For example, when loaded, images are of format `cvMat`, same as in C++.
  - For array operations, there are several functions like `cvSet2D`, `cvGet2D`, etc. And some discussions say, they are slower.

- `cv2` (latest): everything is returned as NumPy objects like `ndarray` and native Python objects like `lists`, `tuples`, `dictionary`, etc.
  - NumPy is a highly stable and fast array processing library.

Easy installation (now!):

- **`pip install opencv-python`** (core modules)
- `pip install opencv-contrib-python` (extra modules)

- And also:
- **`pip install numpy`**
- `pip install matplotlib`
- `pip install scipy`

```
C:\Users\rjamp>python
Python 3.7.1 (default, Dec 10 2018, 22:54:23) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>> cv2.__version__
'4.0.0'
>>>
```

Getting Started with Images

- First:

```
import cv2

#or then and very common
import cv2 as cv
```

- Then:
- read an image - `cv2.imread()`
- display it - `cv2.imshow()`
- save it - `cv2.imwrite()`

Importing an image

- Create a Python module and write the following code:

```
import cv2
img = cv2.imread('duomo.jpg',1)
cv2.imshow("Output Window", img)
cv2.waitKey()
```

- This code imports an image and outputs it to a window and waits for any user keyboard input to terminate.

`cv2.imread()` function

- This function is used to load an image and store it into a variable

```
img = cv2.imread('duomo.jpg',1)
```

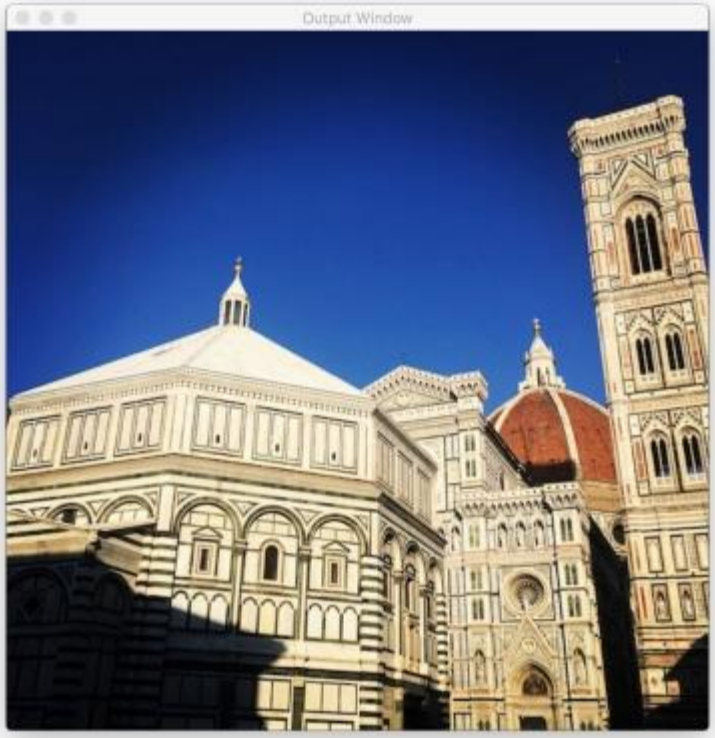
- This function accepts 2 parameters:

1. The filename of the image
2. Colour Approach:

    1: Colour, neglecting transparency

    0: Greyscale

    -1: Colour together with the alpha channel




```
img = cv2.imread('duomo.jpg',1)
```


```
img = cv2.imread('duomo.jpg',0)
```

`cv2.imshow()` function

- This function is used to display an image in a window. `cv2.imshow("Output Window", img)`

- This function accepts 2 parameters:

1. The name of the output window
2. The image to be displayed in the output window

- The window automatically fits the image size.
- Matplotlib can be used as an alternative

`cv2.waitKey()` function

- This is a keyboard binding function

`cv2.waitKey()`

- A single argument value in milliseconds:

1. 0 or no argument: wait indefinitely for keyboard interrupt
2. Any other value: display the window for the duration of that value in `ms`

- This function returns the ASCII value of the key pressed and if stored in a variable, it can be used to perform subsequent logical operations.

Getting Started with Videos

- cv2.VideoCapture()
- cv2.VideoWriter()

`cv2.VideoCapture()` Object

- The video capture object allows us to manipulate captured frames from a camera.

```
cap = cv2.VideoCapture(0)
```

- The argument is either the video filename or camera index, 0 for webcam. Allows the handling of each frame.

- After being used, the capture has to be released:

```
cap.release()
```

Using the webcam feed

```
cap = cv2.VideoCapture(0)
while(True):
	# Capture frame-by-frame
	ret, frame = cap.read()

	# Our operations on the frame come here
	gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

	# Display the resulting frame
	cv2.imshow('frame',gray)
	if cv2.waitKey(1) & 0xFF == ord('q'):
		break

# When everything done, release the capture
cap.release()
```

Importing a video

```
cap = cv2.VideoCapture('vtest.avi')
while(cap.isOpened()): #returns true when there is another frame

                                #to process
                ret, frame = cap.read()

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

                cv2.imshow('frame',gray)
                if cv2.waitKey(1) & 0xFF == ord('q'):
                        break

cap.release()
```

`cv2.VideoWriter()` Object

- The video writer object allows us to save a video to an output file.

```
out = cv2.VideoWriter('output.avi',fourcc, 20.0, (640,480))
```

- FourCC is a 4-byte code used to specify the video codec. The list of available codes can be found in fourcc.org. It is
- platform dependent.

- After being used, the capture has to be released:

```
out.release()
```

Drawing Functions in OpenCV

- OpenCV can draw different geometric shapes with these functions:

```
cv2.line()
cv2.circle()
cv2.rectangle()
cv2.ellipse()
cv2.putText()
```

…

Handle mouse events in OpenCV

- Draw stuffs with your mouse
- Use it as a Paint-Brush

```
cv2.setMouseCallback()
```

- Can bind trackbar to OpenCV windows
- Trackbar as the Color Palette

```
cv2.getTrackbarPos()
cv2.createTrackbar()
```

…

Code Pack 28

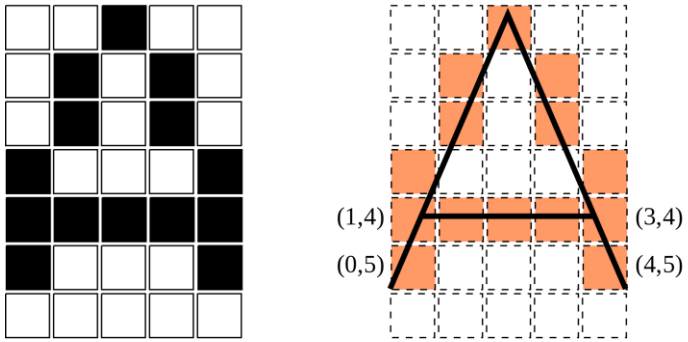
1. Introduction to OpenCV with Python

Coding Bootcamp Code in Python

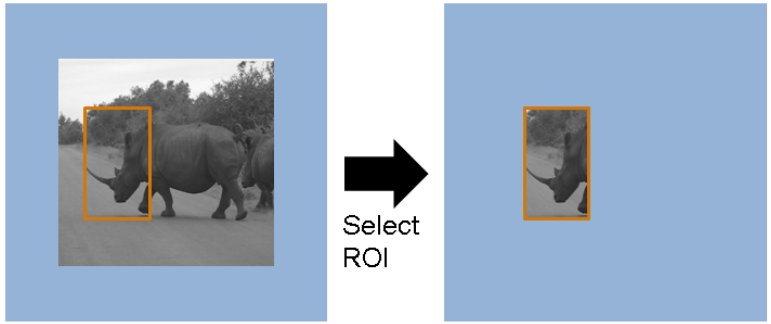
INTRODUCTION TO OPENCV: CORE OPERATIONS

Basic Operations on Images

- Access pixel values and modify them



- - Access image properties (`img.shape`, `img.size`…)
- Setting Region of Interest (ROI)



- Splitting and Merging images

Arithmetic Operations on Images

- Addition, subtraction or bitwise operation.
- `cv2.add()`
- `cv2.addWeighted()`

- `cv2.subtract()`

- `cv2.bitwise_and()`
- `cv2.bitwise_or()`
- `cv2.bitwise_xor()`
- `cv2.bitwise_not()`

https://docs.opencv.org/4.0.1/d2/de8/group__core__array.html

Performance Measurement and Improvement Techniques

- In image processing, it is mandatory that your code is not only providing the correct solution, but also in the fastest manner.

- Useful functions to measure the performance of your code.
- `cv2.getTickCount`
- `cv2.getTickFrequency`

Code Pack 28

1. ~~Introduction to OpenCV with Python~~
2. Core Operations

Coding Bootcamp Code in Python

INTRODUCTION TO OPENCV: IMAGE PROCESSING IN OPENCV

Changing Colorspaces

In OpenCV is also possible to convert images from one colorspace to another, like:

- BGR ↔ Gray
- BGR↔ HSV
- etc

There are more than 150 colorspace conversion methods available in OpenCV.

- `cv2.cvtColor()`
- `cv2.inRange()`
- …

Changing colorspace makes the simplest method in object tracking:

- Take each frame of the video

- Convert from BGR to HSV color-space

- Threshold the HSV image for a range of blue color

- Now extract the blue object alone and get the new position.

Image Thresholding - Simple thresholding

- Thresholding is a method of image segmentation.
- **Simple thresholding** - If pixel value is greater than a threshold value, it is assigned one value (may be white), else it is assigned another value (may be black).

Image Thresholding - Adaptive Thresholding

- **Adaptive Thresholding** - In this, the algorithm calculate the threshold for a small regions of the image.
- So we get different thresholds for different regions.

Image Thresholding - Functions

- `cv2.threshold()`
- `cv2.adaptiveThreshold()`

Geometric Transformations of Images

- Scaling

`cv2.resize()`

- Translation

`cv2.warpAffine()`

- Rotation

`cv2.getRotationMatrix2D()`

- Affine Transformation

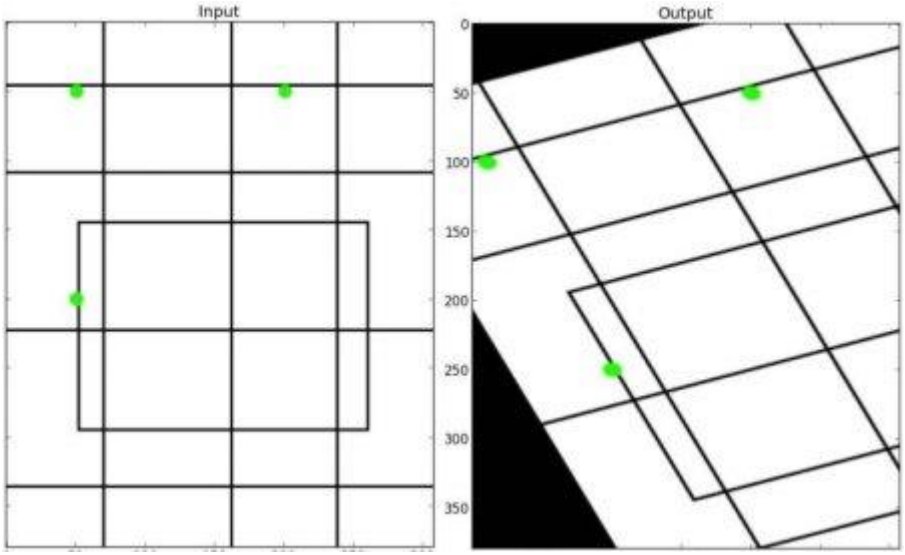
`cv2.getAffineTransform()` then

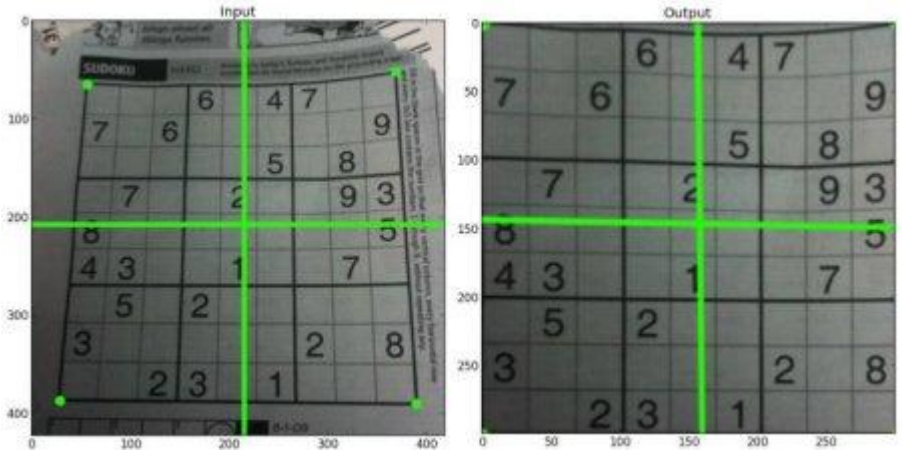
`cv2.warpAffine()`

- Perspective Transformation

`cv2.getPerspectiveTransform()` then
`cv2.warpPerspective()`

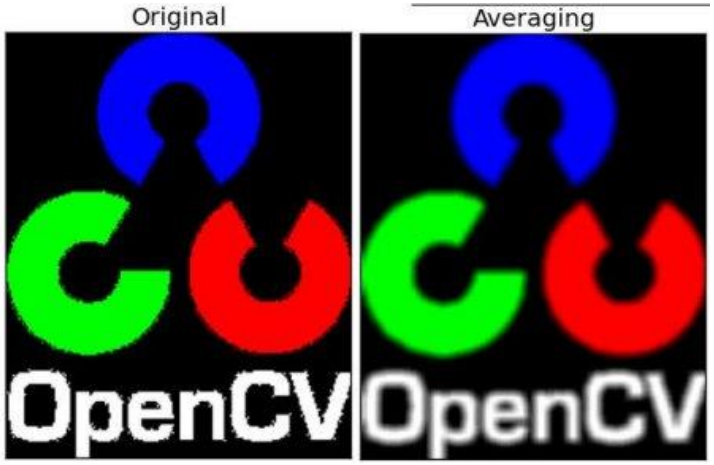
Affine Transformation and Perspective Transformation



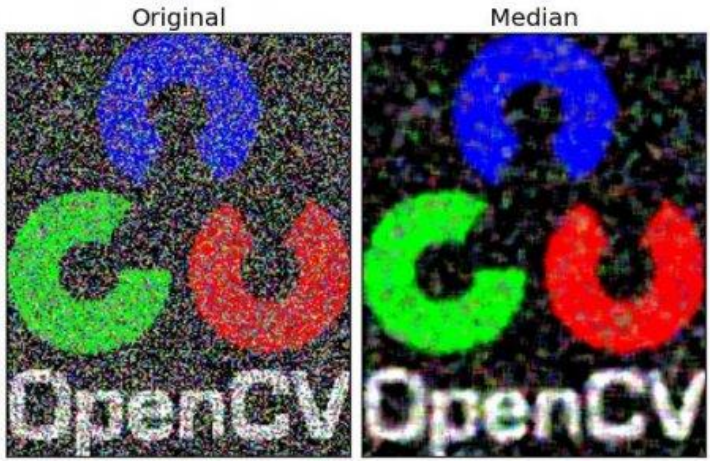


Smoothing Images - Blur images with various low pass filters.

- Averaging – `cv2.blur()`

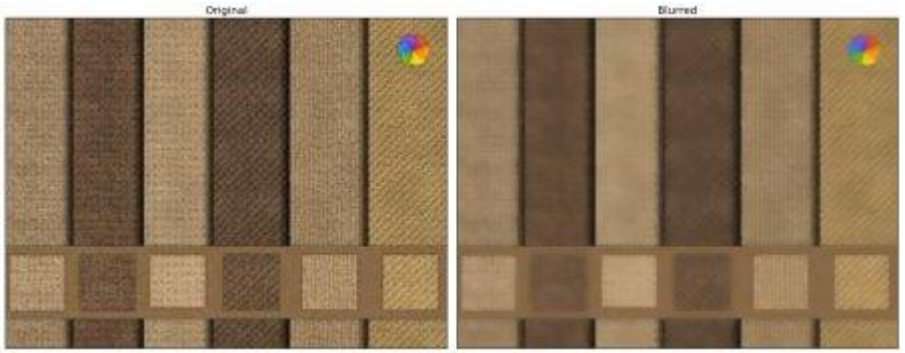


- Median – `cv2.medianBlur()`



- Gaussian - `cv2.GaussianBlur()`

- Bilateral- `cv2.bilateralFilter()`



Morphological Transformations

- Erosion, Dilation, Opening, Closing …

`cv2.erode(), cv2.dilate(), cv2.morphologyEx()…`

Base

Erosion

Closing

Dilation

Opening

Morphological Transformations

Image I

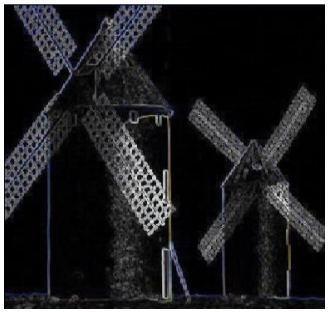
Erosion I⊖B

Dilatation I⊕B

Opening IoB= (I⊖B)⊕B

Closing I•B= (I⊕B)⊖B

Grad(I)= (I⊕B)-(I⊖B)

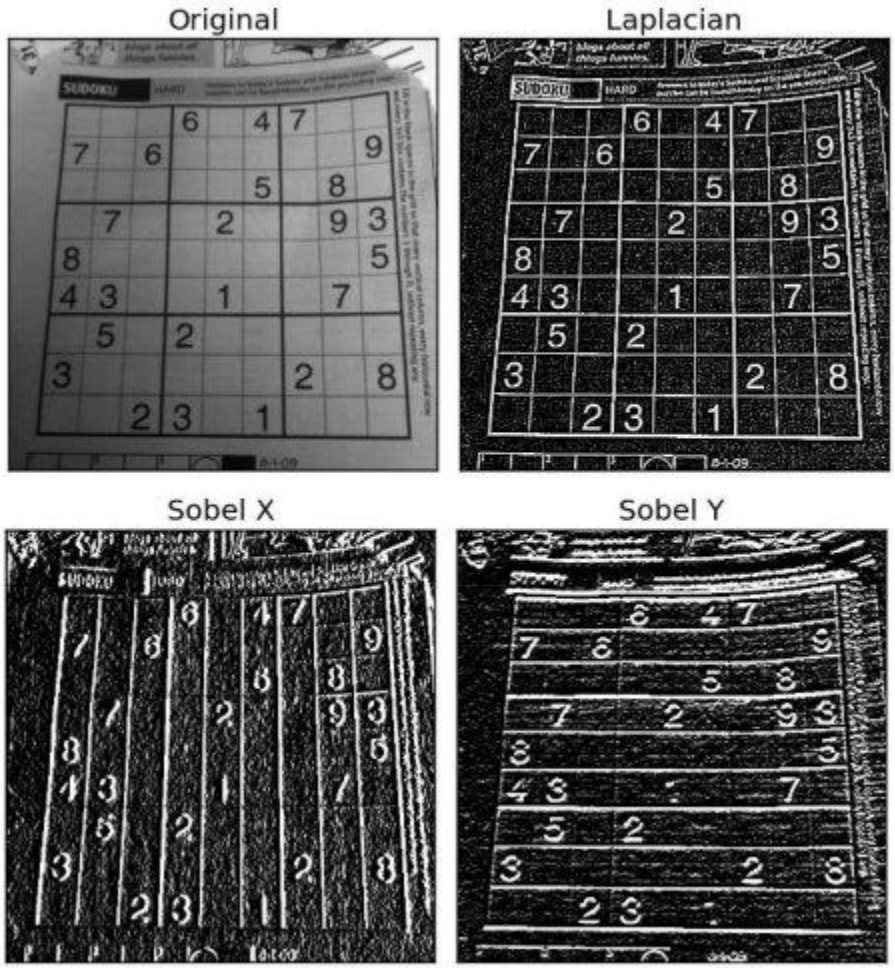


TopHat(I)= I - (I⊖B)

BlackHat(I)= (I⊕B) - I

Image Gradients

- `cv2.Sobel(), cv2.Laplacian()`



Canny Edge Detection

- Canny Edge Detection is a popular edge detection algorithm.
- It was developed by John F. Canny in 1986.
- `cv2.Canny()`

Image Pyramids

- Image segmentation by pyramids - Gaussian and Laplacian pyramids
- **Gaussian pyramid**: Used to downsample images
- **Laplacian pyramid**: Used to reconstruct an upsampled image from an image lower in the pyramid (with less resolution)

- `cv2.pyrUp()`
- `cv2.pyrDown()`

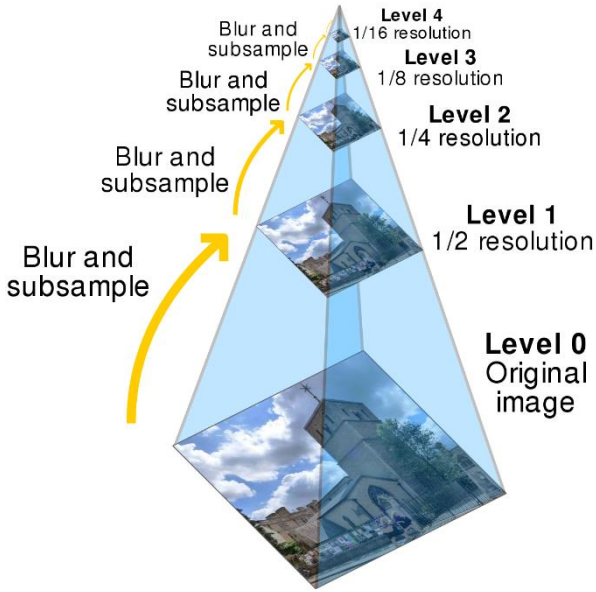
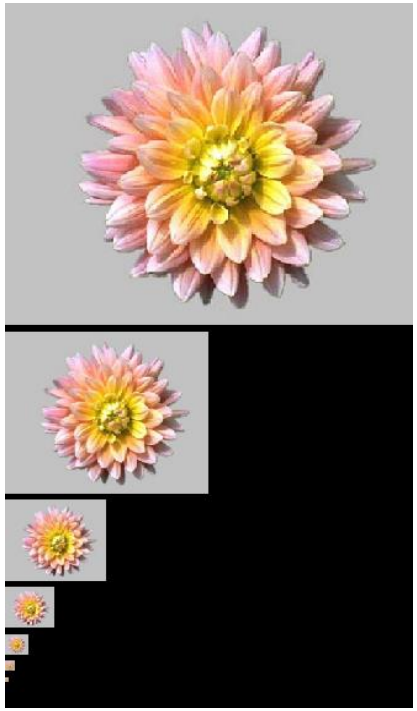


Image Pyramids



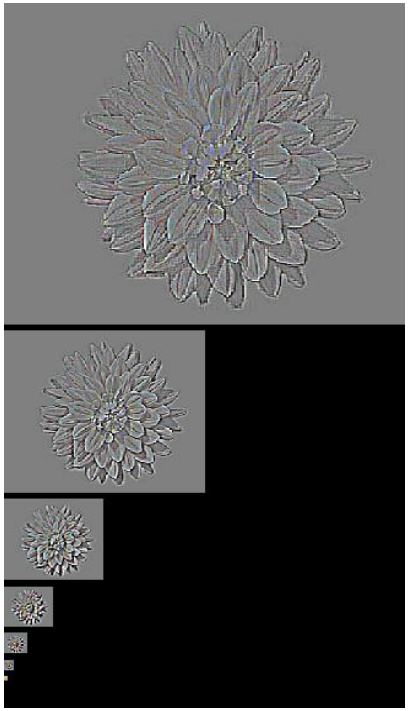
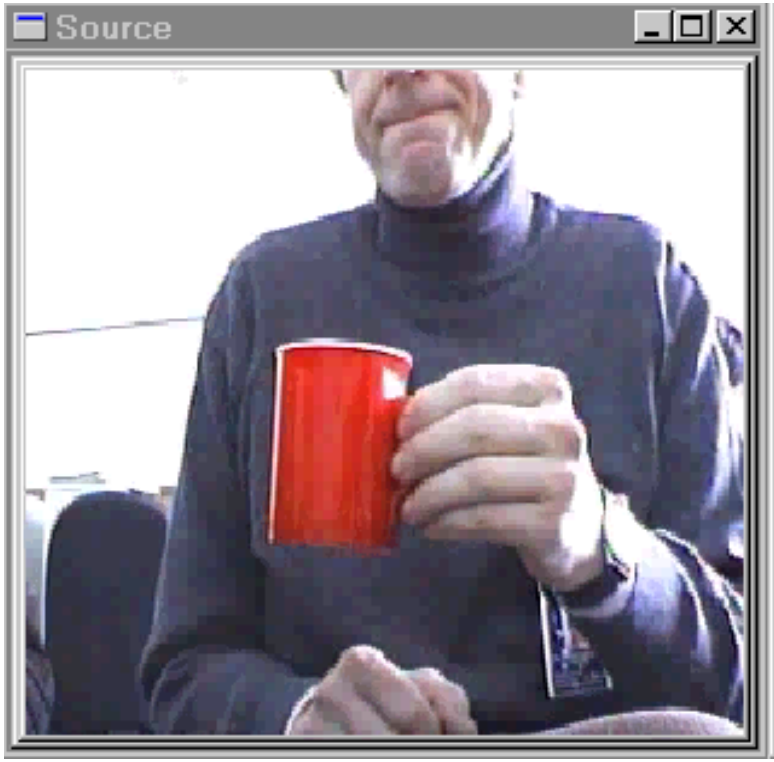
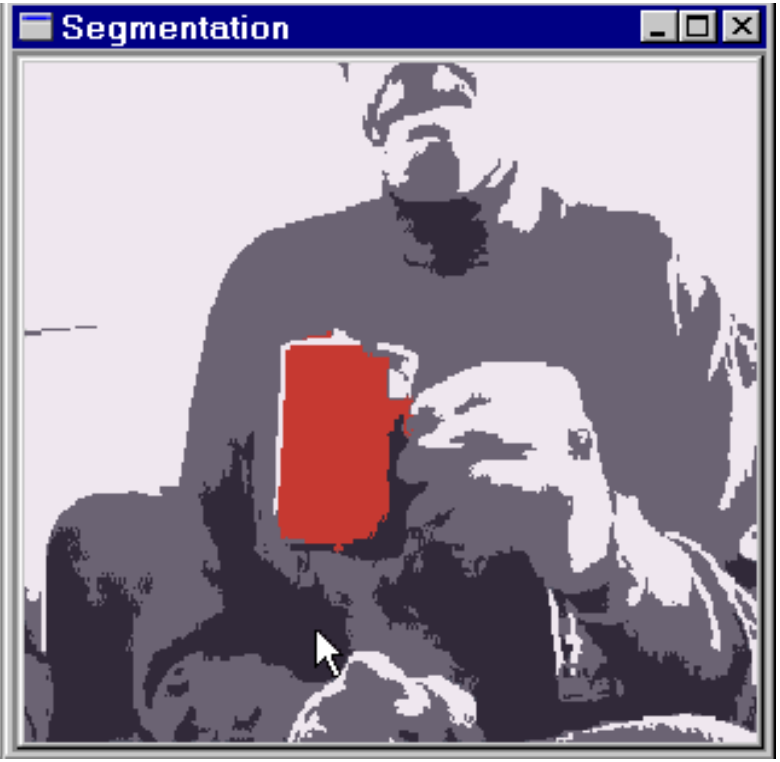


Image Pyramids

On still pictures



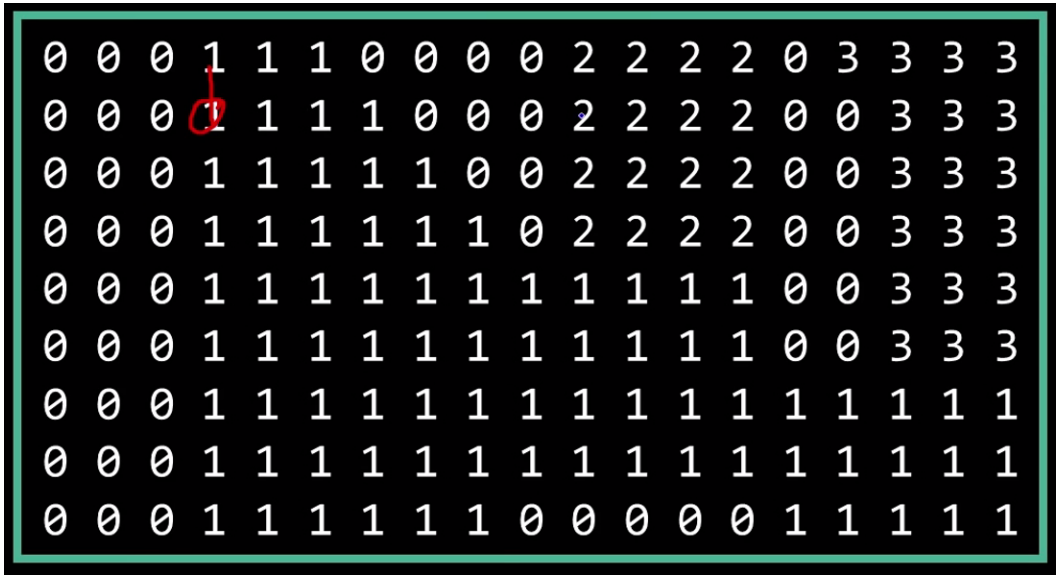
And on movies



Contours in OpenCV

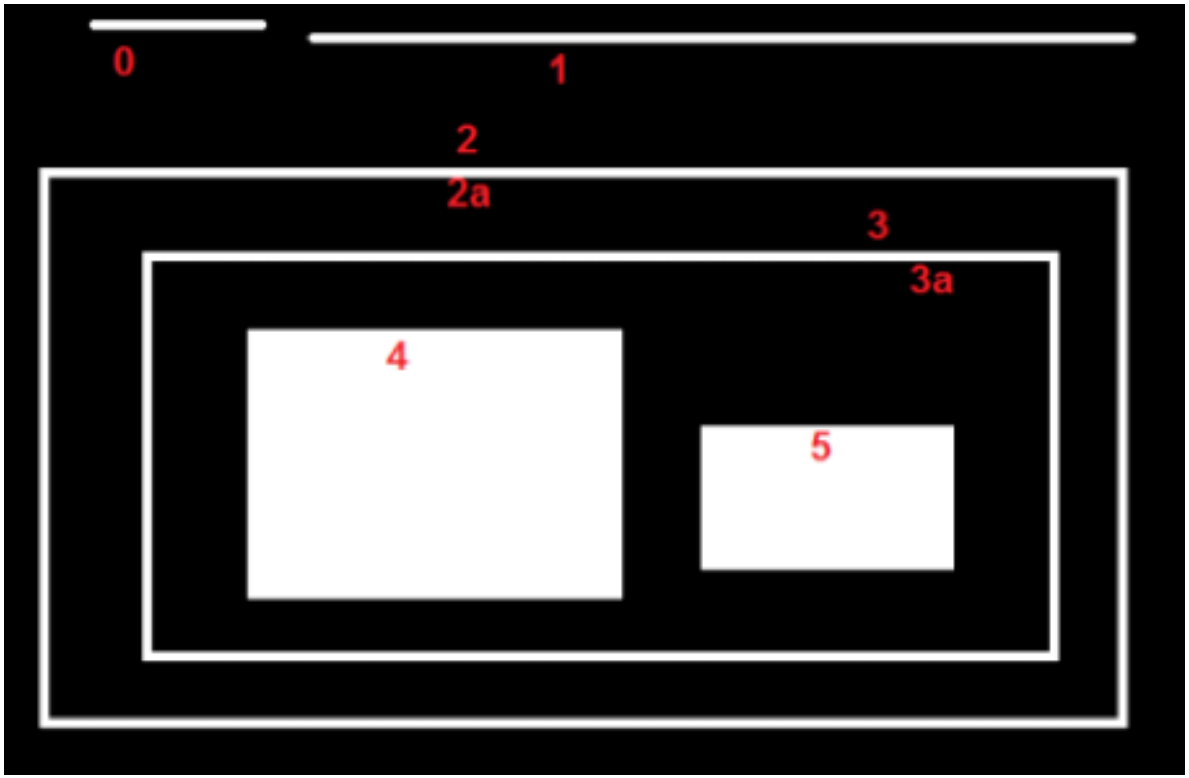
- Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity.

- The contours are a useful tool for shape analysis and object detection and recognition.



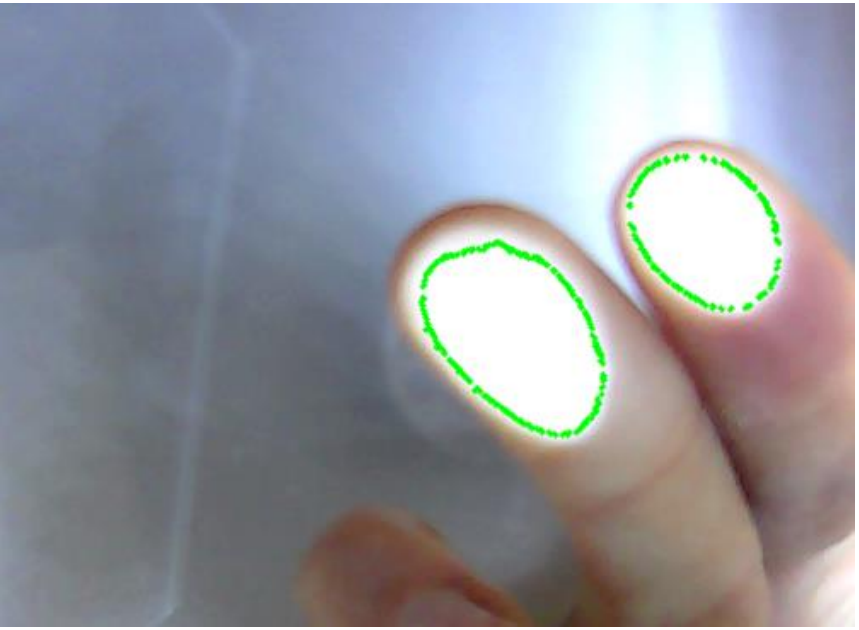
Connected components = find contours

- `contours, hierarchy = cv2.findContours()`

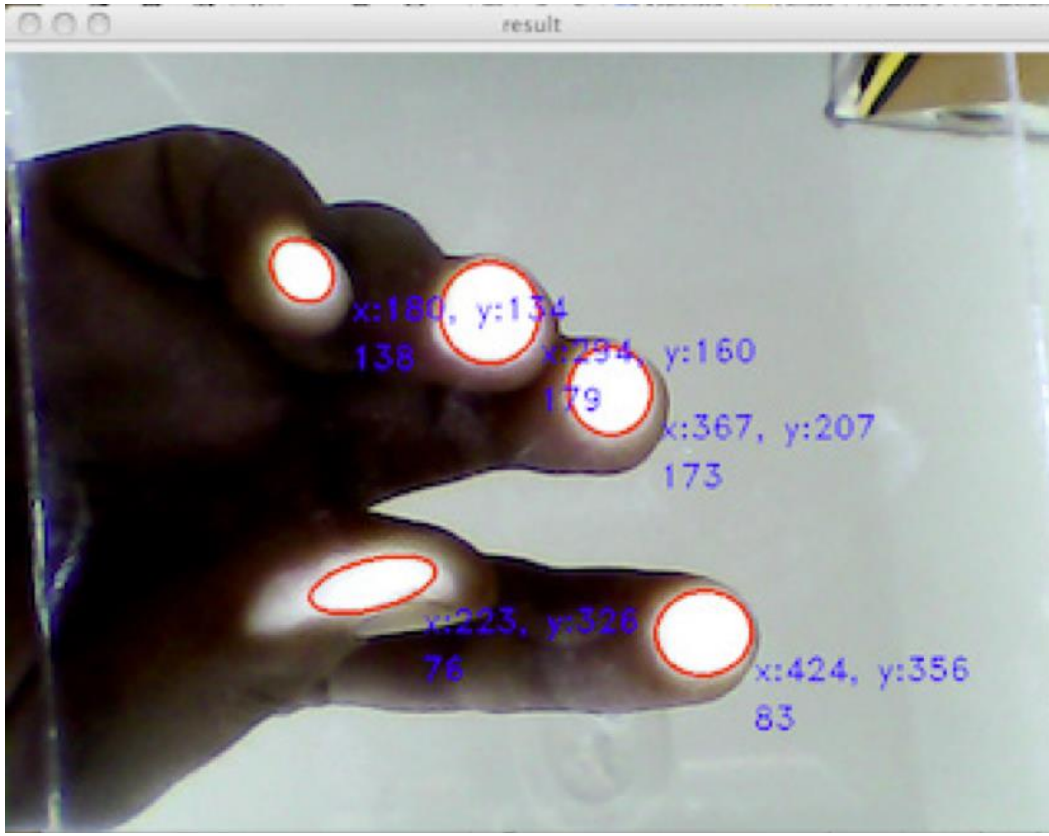


Find center of a contour:

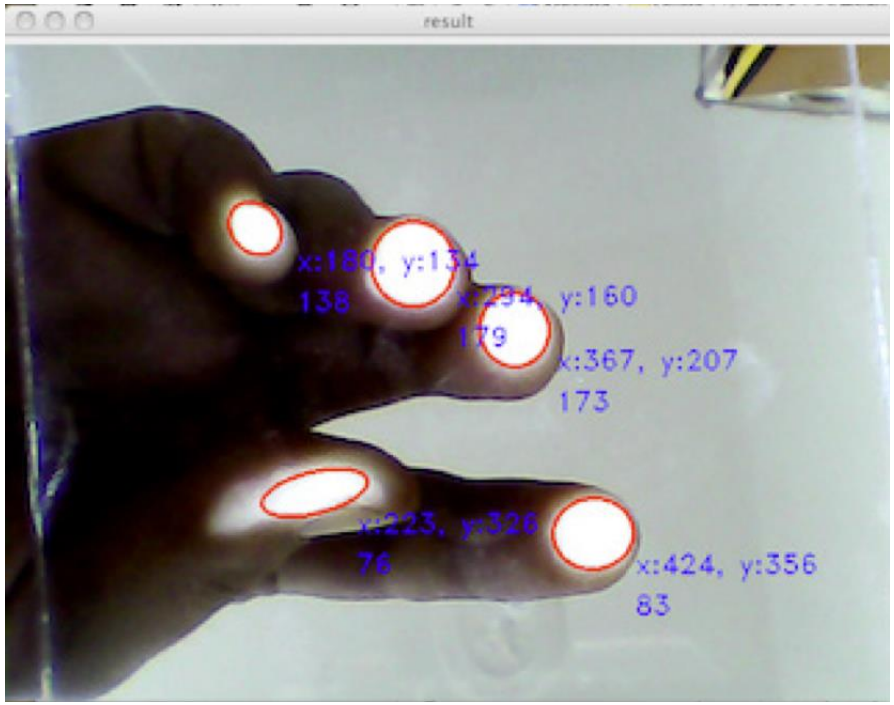
- `cv2.minEnclosingCircle(contour)`



contour = path of points



fit circle = center, radius

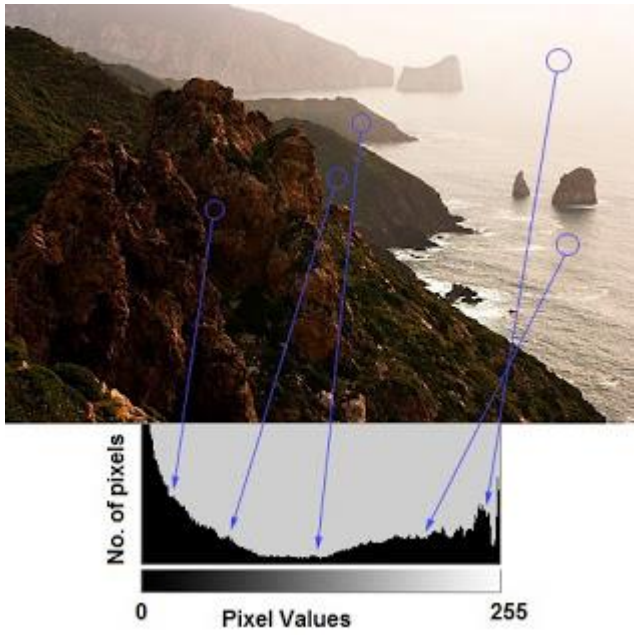



```
cv2.circle(frame,draw_center,draw_radius,(0,255,0),2)

cv2.putText(frame,textstring,(int(x+50),int(y)),
            cv2.FONT_HERSHEY_SIMPLEX, 0.5,(255,255,255),1)
```

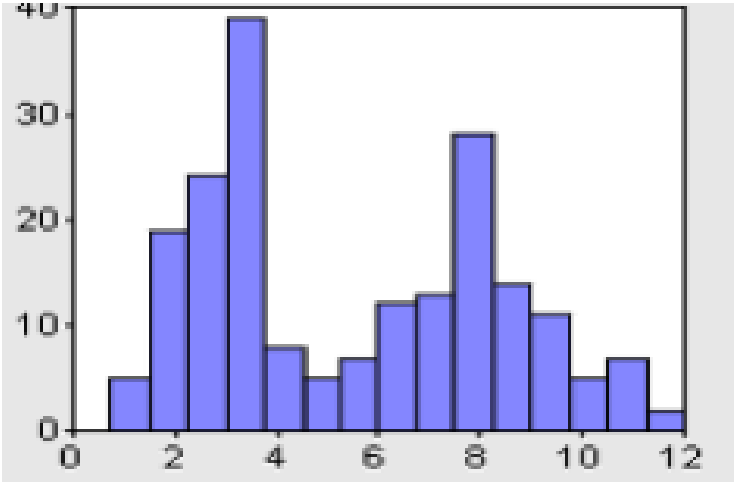
Histograms in OpenCV

- Histogram is a graph or plot, which gives you an overall idea about the intensity distribution of an image. It is a plot with pixel values (ranging from 0 to 255, not always) in X-axis and corresponding number of pixels in the image on Y-axis.



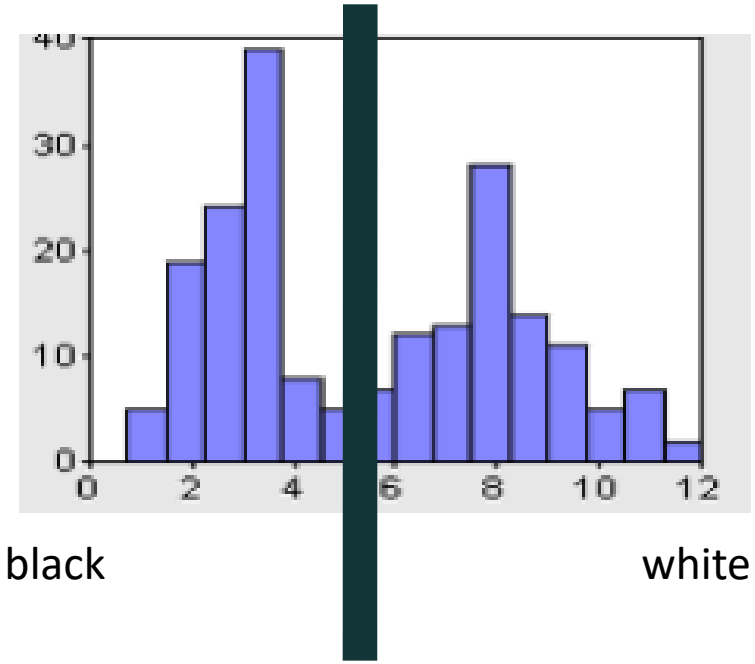
- `cv2.calcHist()`
- `np.histogram()`

let’s consider
this image



here’s how
the histogram works

black white



where would you place the threshold to find all trees?

Image Transforms in OpenCV

- Fourier Transform of images using OpenCV
- `cv2.dft(), cv2.idft()`

- There are FFT functions available in Numpy

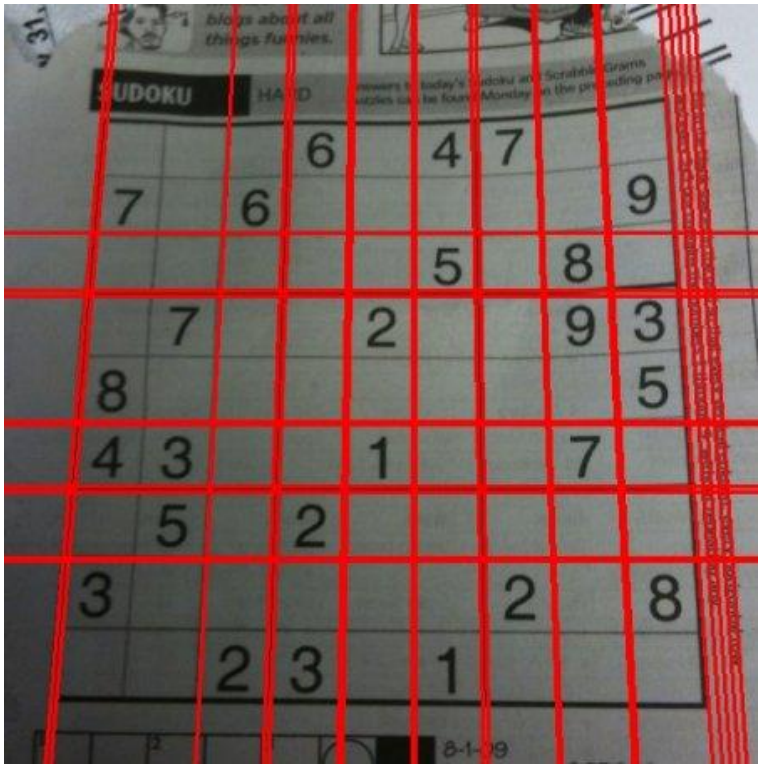
Template Matching

- Template Matching is a method for searching and finding the location of a template image in a larger image.

- OpenCV comes with the `cv2.matchTemplate()`

Hough Line Transform

- The Hough transform is an incredible tool that lets you identify lines and other shapes (circles, angles, etc).
- `cv2.HoughLines()`
- `cv2.HoughLinesP()`



Hough Circle Transform

- Use Hough Transform to find circles in an image
- `cv2.HoughCircles()`

Image Segmentation with Watershed Algorithm

- In geology, a watershed is a divide that separates adjacent catchment basins

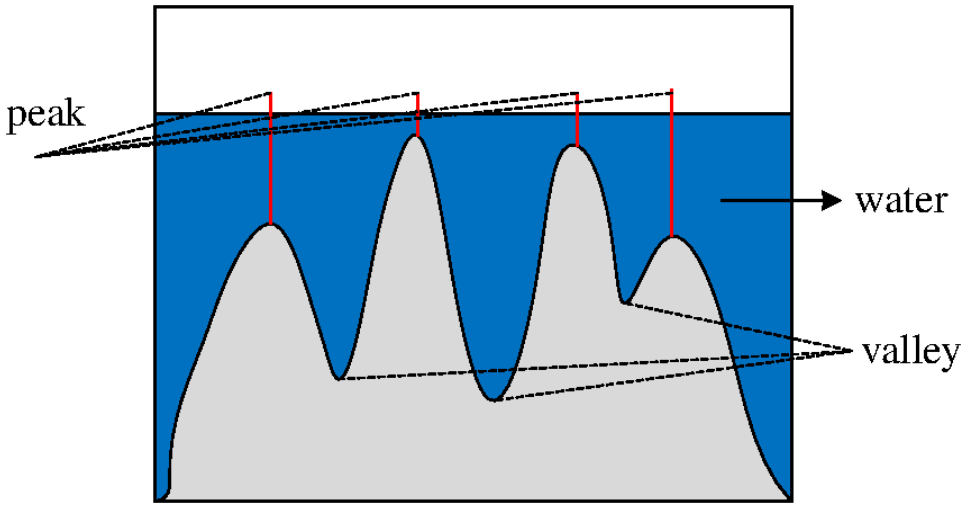
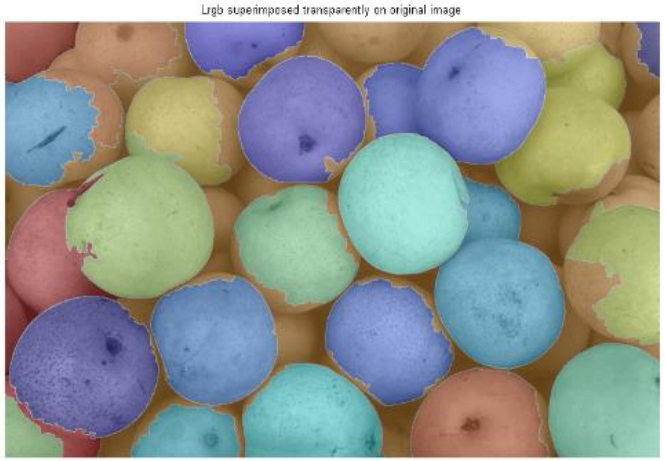


Image Segmentation with Watershed Algorithm

- `cv2.watershed()`



Or…

Original image

Mosaic-image

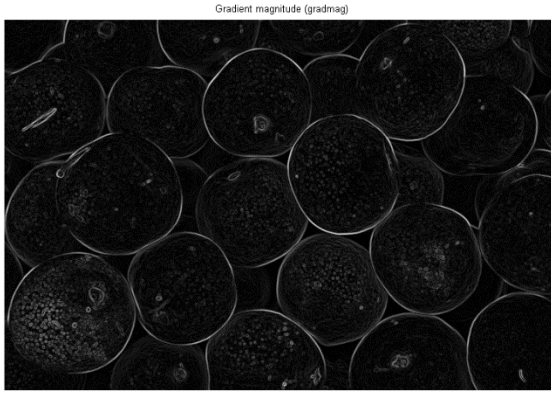
Watershed of mosaic-image

Lanes markers enhancement

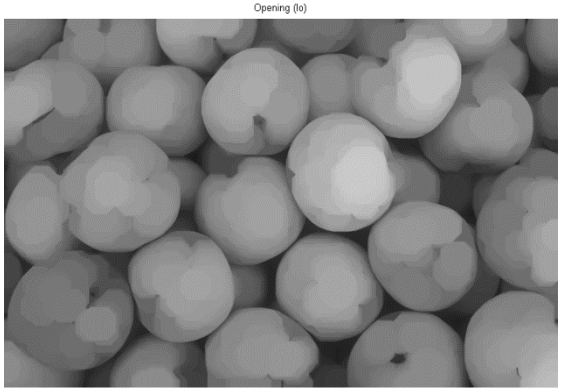
Final result

Watershed Transformation Process

Source: A gray scale image

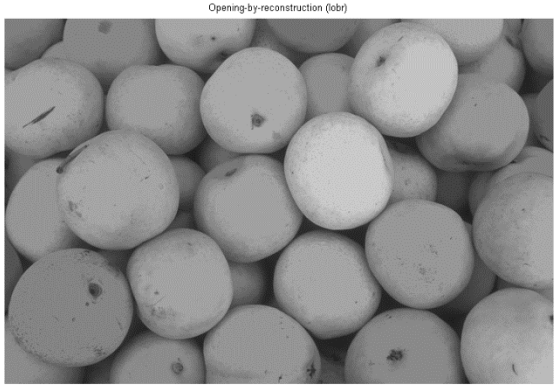


Step 1: Use the Gradient Magnitude as the Segmentation Function - The gradient is high at the borders of the objects and low (mostly) inside the objects.

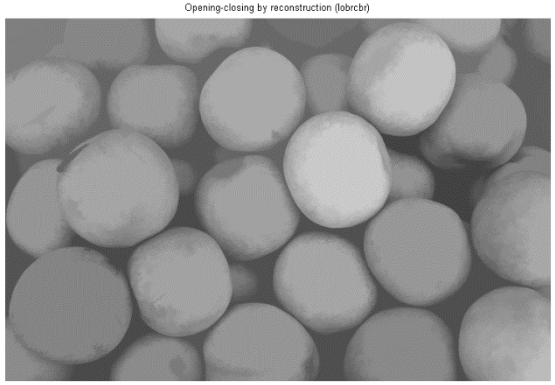


Step 2: Mark the foreground objects

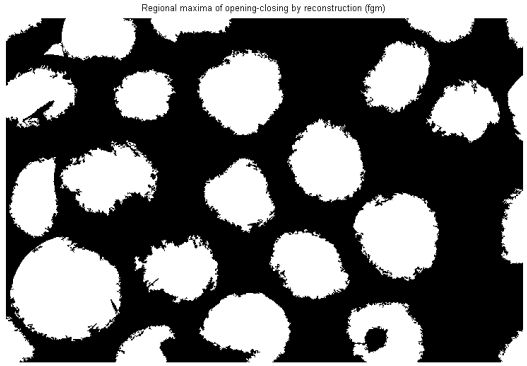
Watershed Transformation Process



Step 3: computing the opening-by-reconstruction of the image

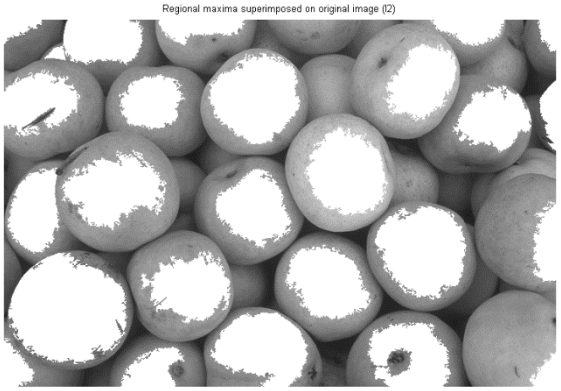


Step 4: Following the opening with a closing can remove the dark spots and stem marks.

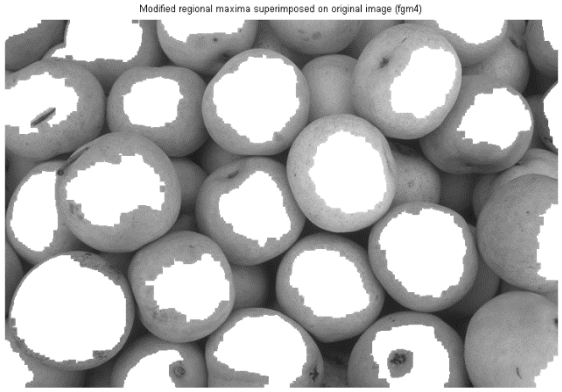


Step 5: Calculate the regional maxima to obtain good foreground markers.

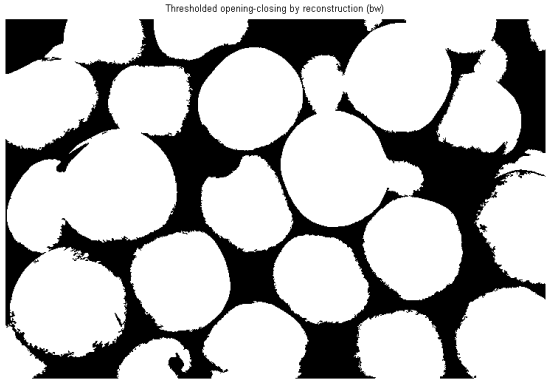
Watershed Transformation Process



Step 6: Superimpose the foreground marker image on the original image, Notice that the foreground markers in some objects go right up to the objects' edge

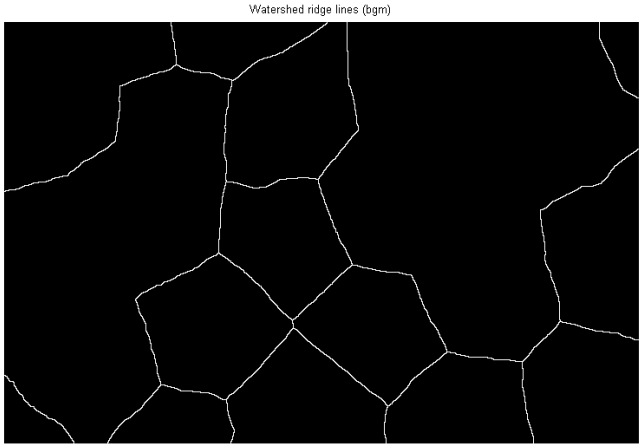


Step 7: cleaning the edges of the marker blobs and then shrinking them a bit

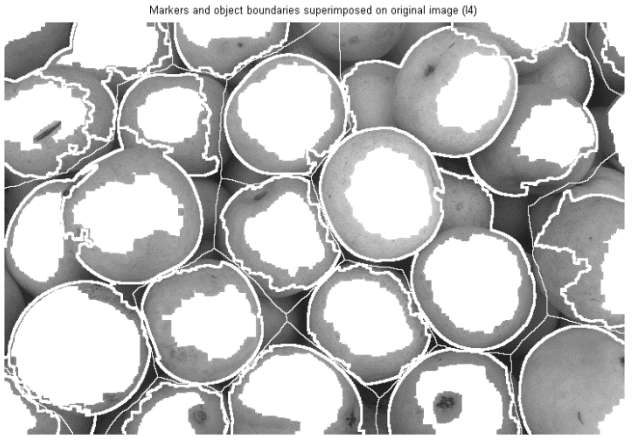


Step 8: Compute Background Markers, Starting with thresholding operation

Watershed Transformation Process

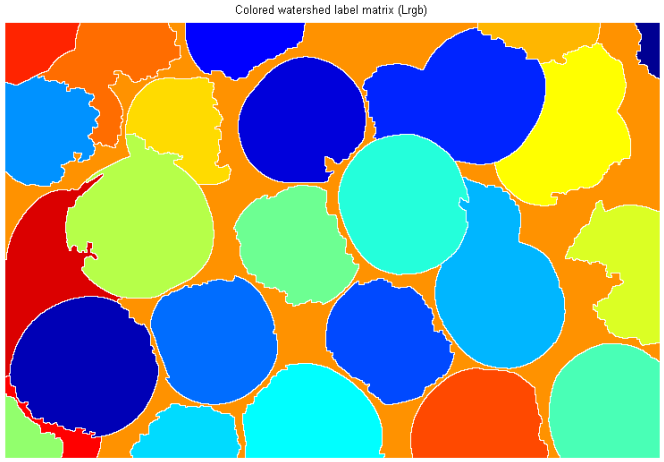


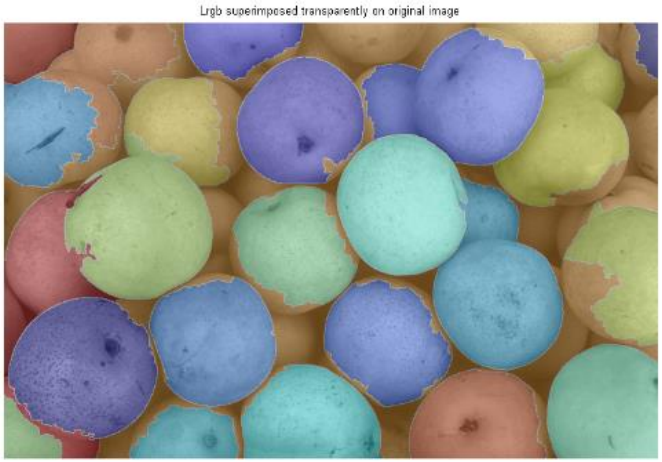
Step 9: Compute Background Markers, using the watershed transform of the distance transform and then looking for the watershed ridge lines of the result



Step 10: Visualize the Result, one of the techniques is to superimpose the foreground markers, background markers, and segmented object boundaries.

Watershed Transformation Process





*Another useful visualization technique is to display the label matrix as a color image

We can use transparency to superimpose this pseudo-color label matrix on top of the original intensity image.

Interactive Foreground Extraction using GrabCut Algorithm






Code Pack 28

1. ~~Introduction to OpenCV with Python~~
2. ~~Core Operations~~
3. Image Processing in OpenCV

Coding Bootcamp Code in Python

INTRODUCTION TO OPENCV: FEATURE DETECTION AND DESCRIPTION

Understanding Features

- Features Points or Interest Point are characteristics in an image which differentiates it.
- Like an ID

Feature points are used for

- Image alignment
- 3D reconstruction
- Motion tracking (robots, drones, AR)
- Indexing and database retrieval
- Object recognition
- …

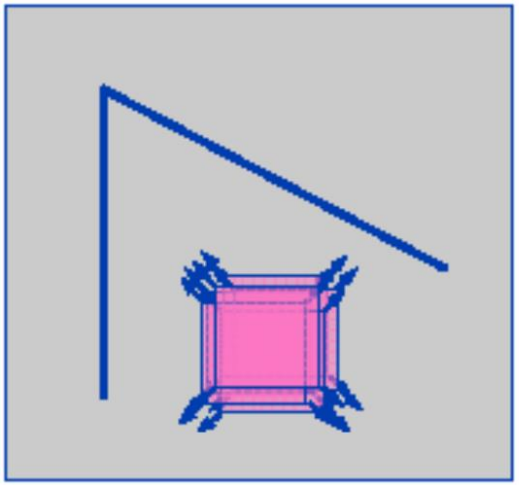
Features Invariance and covariance

Are locations *invariant* to photometric transformations and *covariant* to geometric transformations?

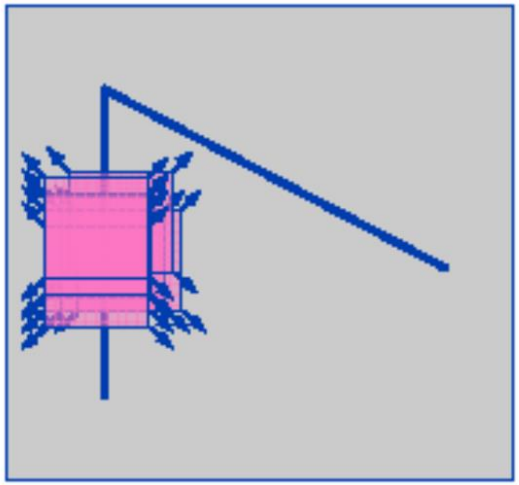
- **Invariance:** image is transformed and corner locations do not change
- **Covariance:** if we have two transformed versions of the same image, features should be detected in corresponding locations

Harris Corner Detection

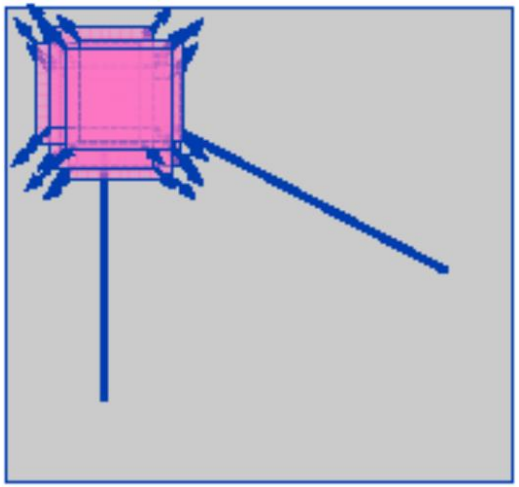
- Harris corner detector gives a mathematical approach for determining which are the corners in a image.
- `cv2.cornerHarris()`



Flat region:
No change in
all directions



Edge region:
No change along
the edge direction

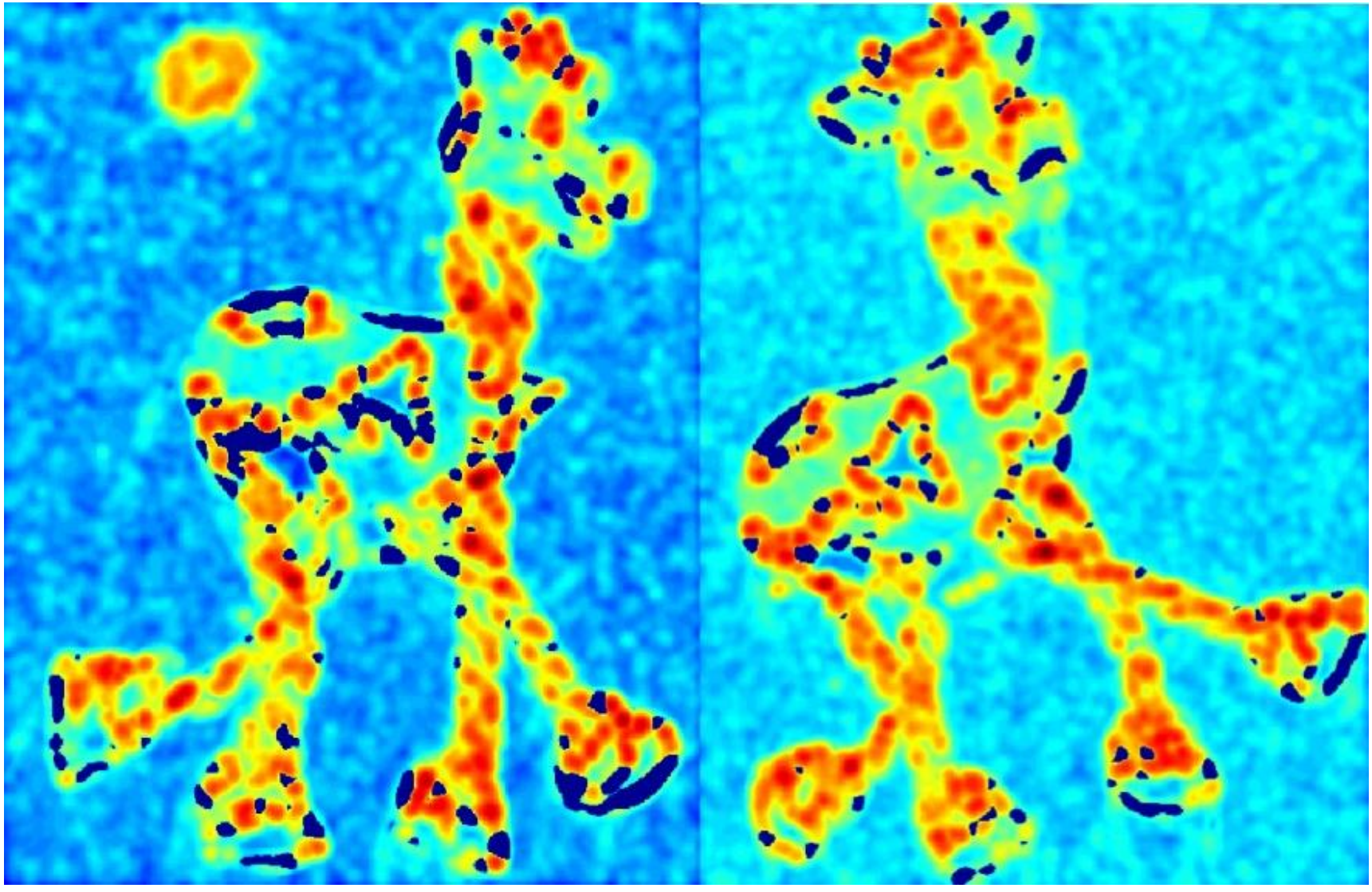


Corner region:
Significant change in
all directions

Harris Corner Detector Algorithm: Begin

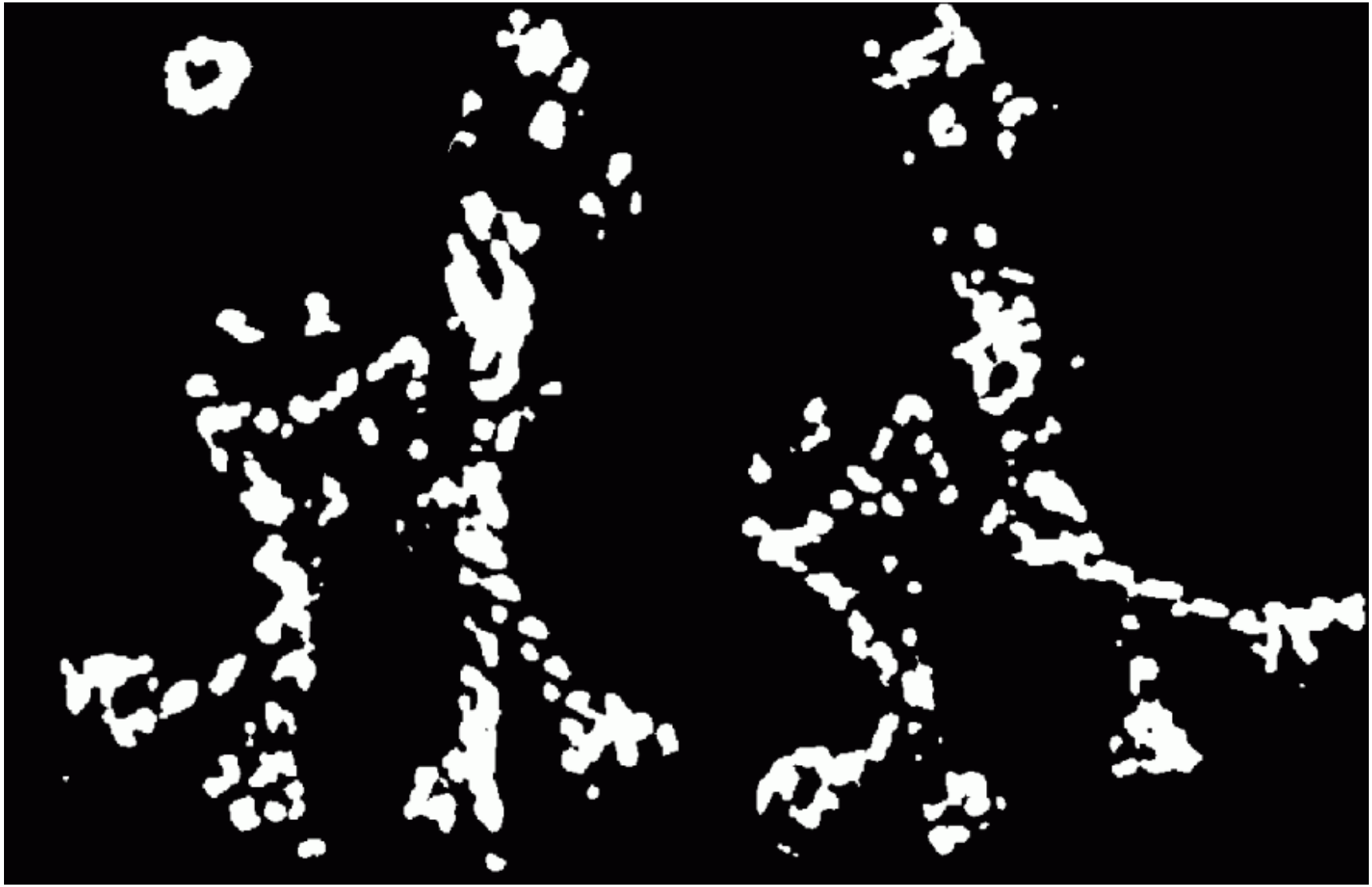
Harris Corner Detector Algorithm: Major Step 1

Compute corner response C



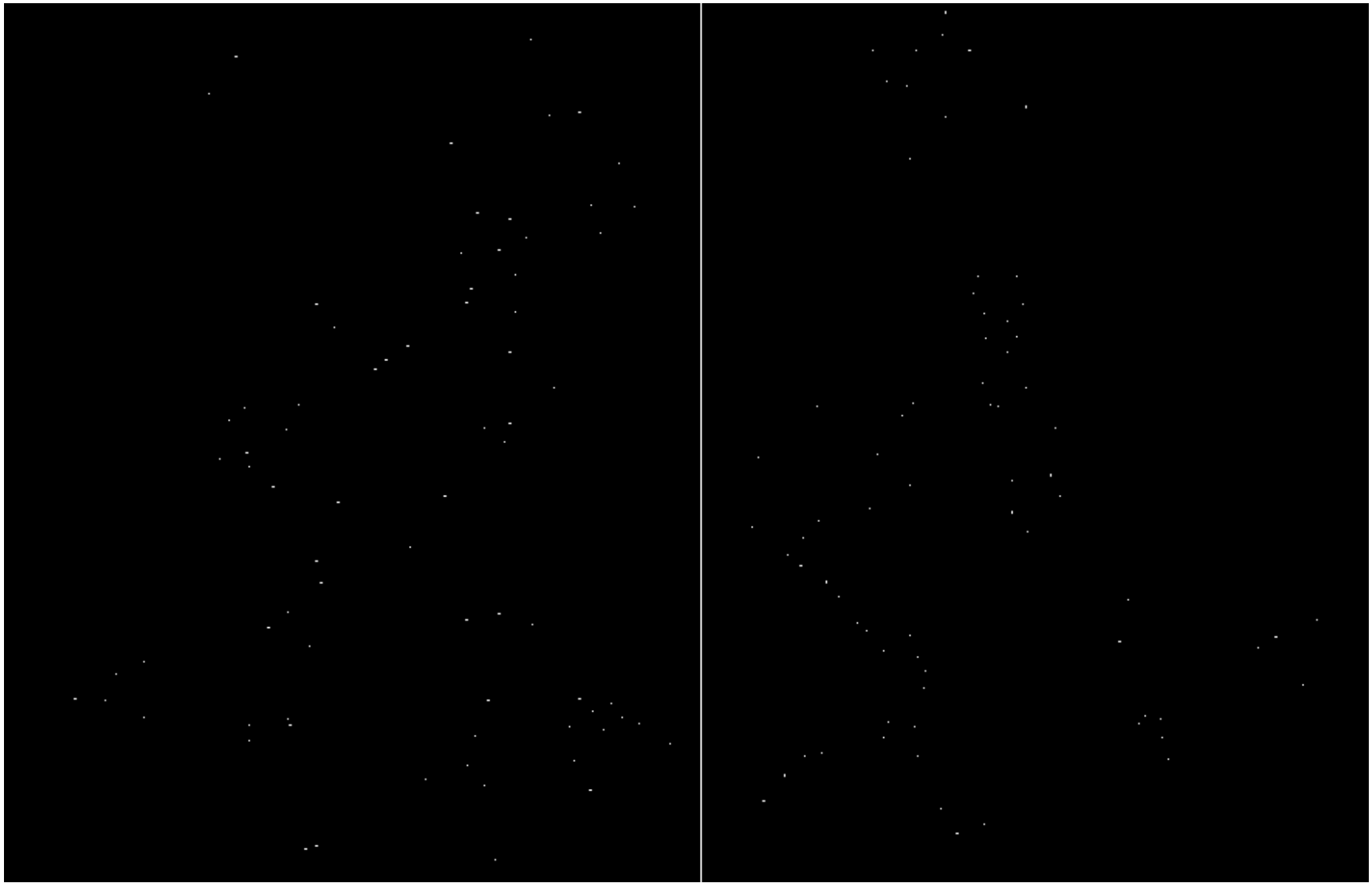
Harris Corner Detector Algorithm: Major Step 2

Find points with large corner response: C > threshold



Harris Corner Detector Algorithm: Major Step 3

Take only the points of local maxima of C



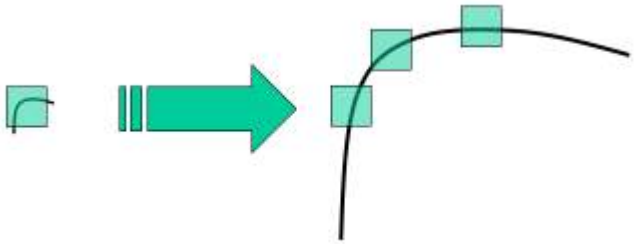
Harris Corner Detector Algorithm: End

Shi-Tomasi Corner Detector & Good Features to Track

- Another more recent corner detector (1994)
- `cv2.goodFeaturesToTrack()`

Introduction to SIFT (Scale Invariant Feature Transform)

- Corner Detectors work if image rotate but fail at scalling. X



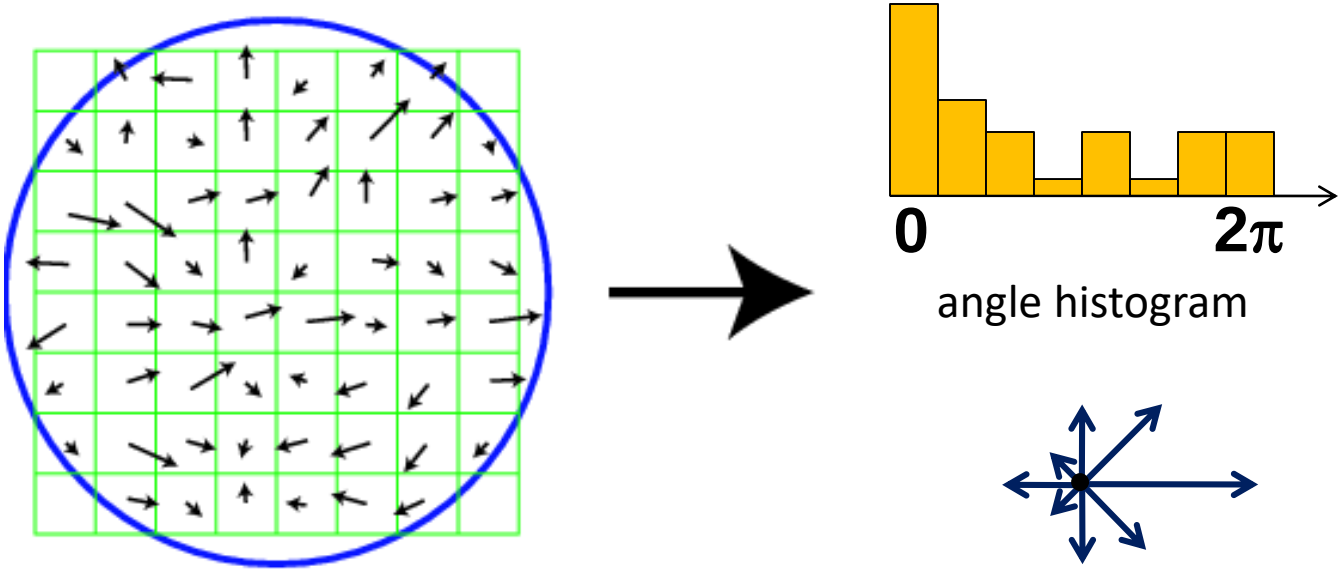
- SIFT (Scale Invariant Feature Transform) solve that. V

- But brings new Concepts:
- **Keypoints** - Features
- **Descriptors** – Encoding of the Features, computed from the keypoint

Scale Invariant Feature Transform

Basic idea:

- Take 16x16 square window around detected feature
- Compute gradient orientation for each pixel
- Create histogram over edge orientations weighted by magnitude



SIFT descriptor

Full version

- Divide the 16x16 window into a 4x4 grid of cells (2x2 case shown below)
- Compute an orientation histogram for each cell
- 16 cells * 8 orientations = 128 dimensional descriptor

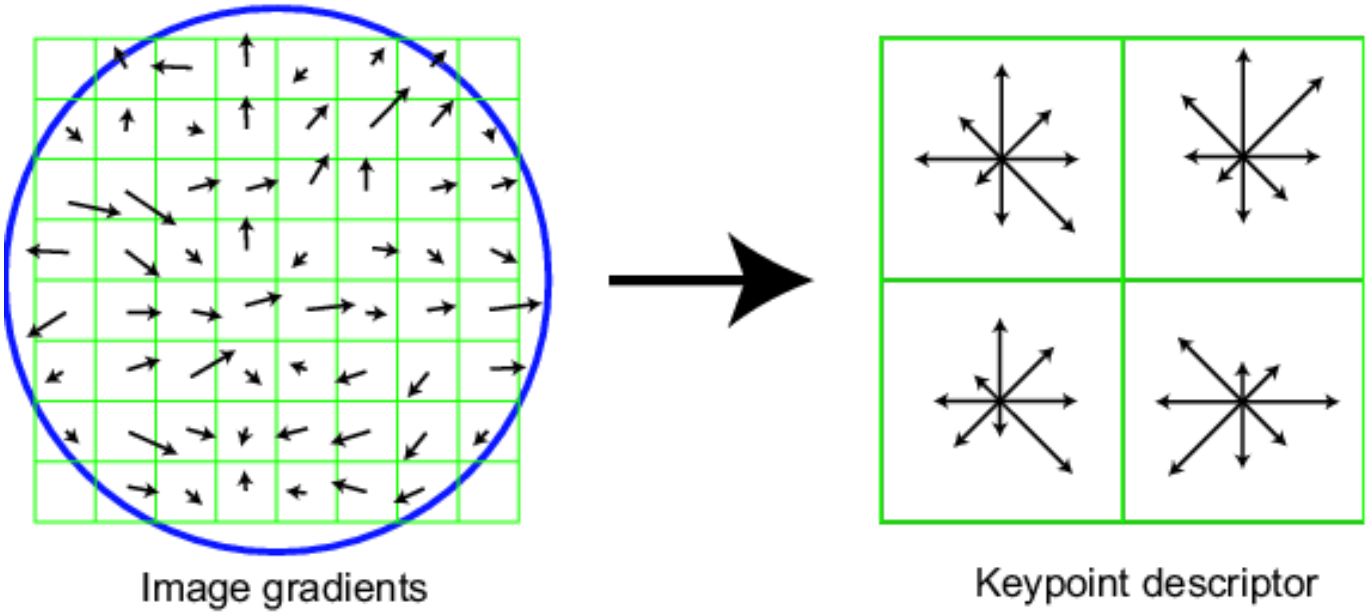


Image gradients

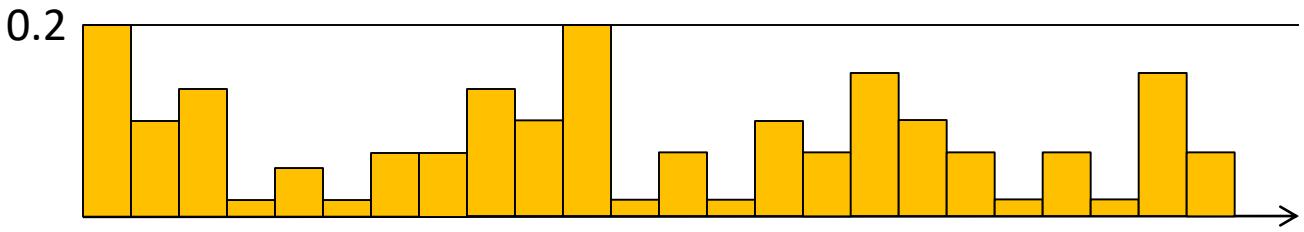
Keypoint descriptor

SIFT descriptor

Full version

- Divide the 16x16 window into a 4x4 grid of cells (2x2 case shown below)
- Compute an orientation histogram for each cell
- 16 cells * 8 orientations = 128 dimensional descriptor
- Threshold normalize the descriptor:

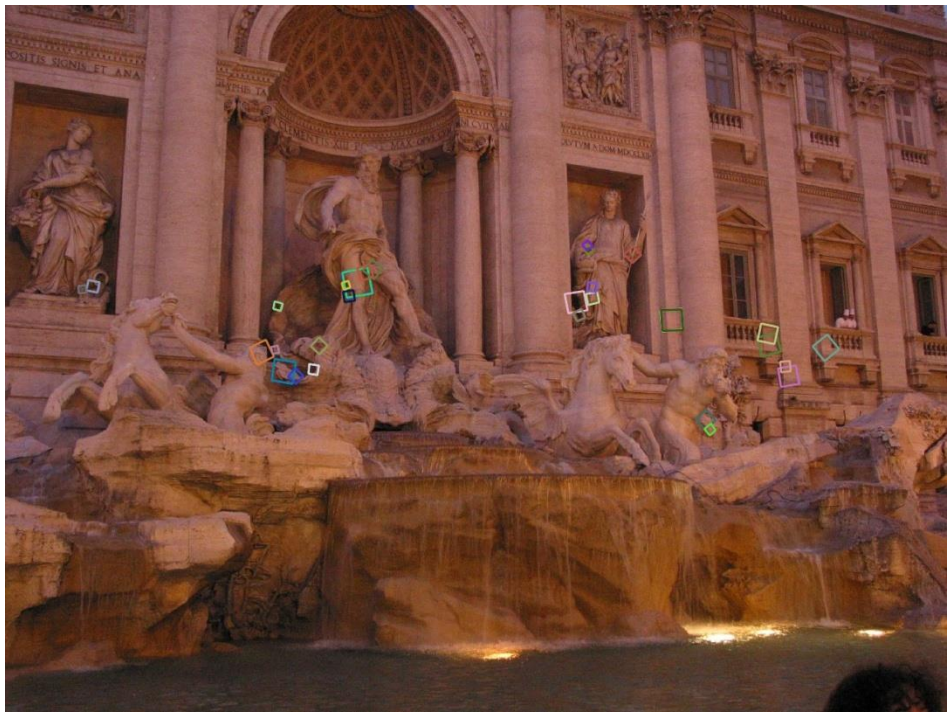
$$\sum_i d_i^2 = 1 \quad \text{such that:} \quad d_i < 0.2$$



Properties of SIFT

Extraordinarily robust matching technique

- Can handle changes in viewpoint
  - Up to about 30 degree out of plane rotation
- Can handle significant changes in illumination
  - Sometimes even day vs. night (below)
- Fast and efficient—can run in real time
- Lots of code available
  - http://people.csail.mit.edu/albert/ladypack/wiki/index.php/Known_implementations_of_SIFT



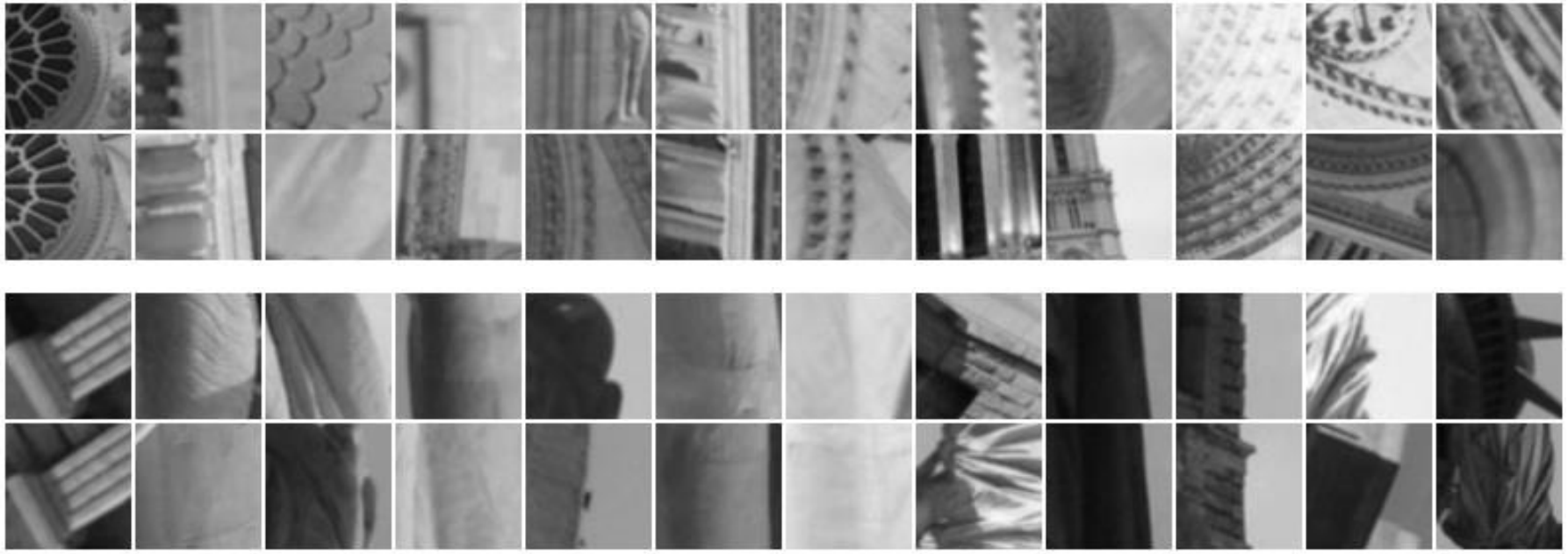
SIFT in OpenCV

- ~~`sift = cv2.SIFT()`~~
- ~~`kp = sift.detect(…)`~~

- `sift = cv2.xfeatures2d.SIFT_create()`
- `kp = sift.detect(…)`

When does SIFT fail?

Patches SIFT thought were the same but aren't:



Introduction to SURF (Speeded-Up Robust Features)

- For computational efficiency only compute gradient histogram with 4 bins.

```
surf = cv2.SURF(…)
cv2.xfeatures2d.SURF_create()
kp, des = surf.detectAndCompute(…)
```

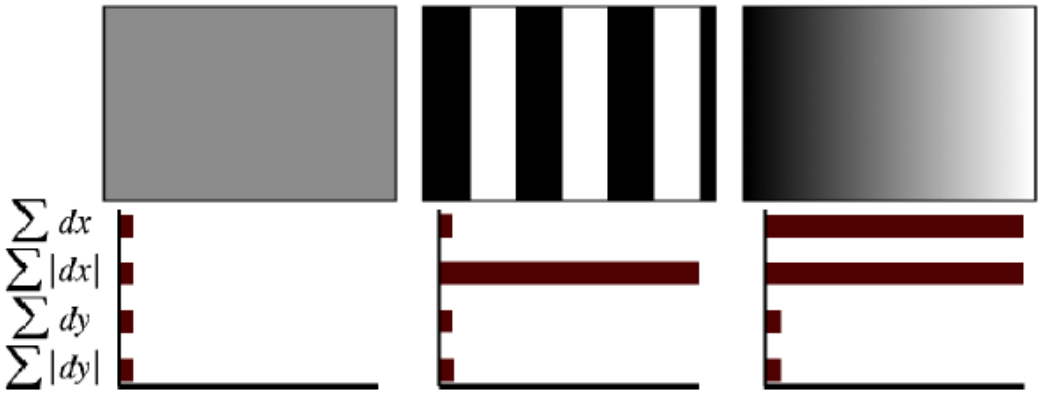


Fig. 3. The descriptor entries of a sub-region represent the nature of the underlying intensity pattern. Left: In case of a homogeneous region, all values are relatively low. Middle: In presence of frequencies in x direction, the value of $\sum |d_x|$ is high, but all others remain low. If the intensity is gradually increasing in x direction, both values $\sum d_x$ and $\sum |d_x|$ are high.

FAST Algorithm for Corner Detection

- Like SURF tries to be a quicker SIFT.

- FAST (Features from Accelerated Segment Test) tries to be a quicker Corner Detection

```
fast = cv2.FastFeatureDetector_create()
kp = fast.detect(…)
```

BRIEF (Binary Robust Independent Elementary Features)

- SURF → quicker SIFT.
- FAST → quicker Harris

- BRIEF is a easier descriptor to compare and store.
- Binary-string descriptors

```
brief =
 cv.xfeatures2d.BriefDescriptorExtractor_create()

brief.compute(…)
```

ORB (Oriented FAST and Rotated BRIEF)

- First that came from “OpenCV Labs”

- An efficient alternative to SIFT or SURF in 2011.
- SIFT and SURF are patented and its supposed to pay them for its use.
- ORB is not !!!

```
orb = cv2.ORB()
kp = orb.detect(…)
kp, des = orb.compute(…)
```

Feature Matching

- Identify if it's the same object (in less than a error)

- Brute-Force Matching with SIFT Descriptors and Ratio Test

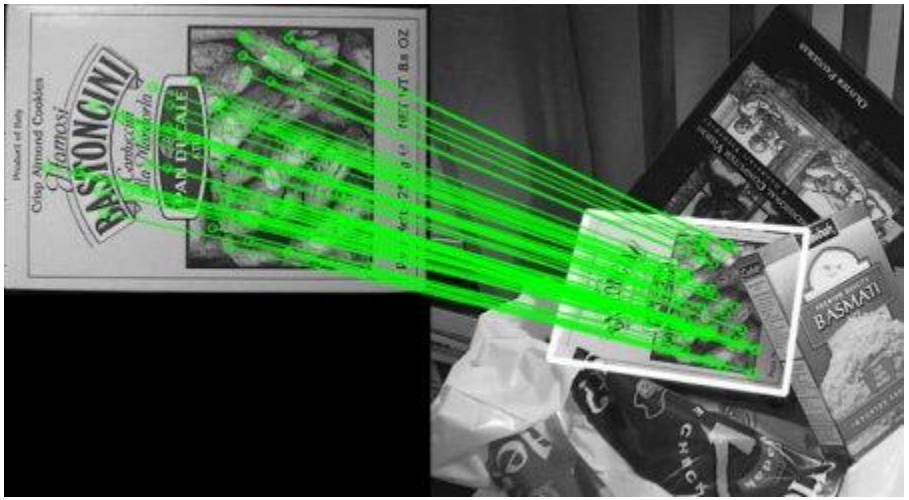
`cv2.BFMatcher()`

- FLANN (Fast Library for Approximate Nearest Neighbors)

`cv2.FlannBasedMatcher`

Feature Matching + Homography to find Objects

- Join Techniques and other OpenCV tools to work with complex images



Code Pack 28

1. ~~Introduction to OpenCV with Python~~
2. ~~Core Operations~~
3. ~~Image Processing in OpenCV~~
4. Feature Detection and Description

Coding Bootcamp Code in Python

INTRODUCTION TO OPENCV: VIDEO ANALYSIS

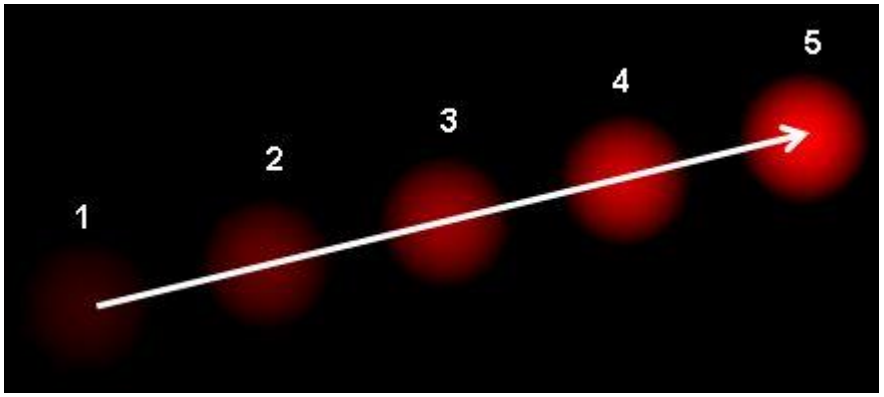
Meanshift and Camshift

- Meanshift and Camshift algorithms are used to find and track objects in videos.

- `cv2.meanShift()` – only used if the object has the same size
- `cv2.CamShif()` – used more if the object resizes in the image

Optical Flow

- Pattern of apparent motion of image objects between two consecutive frames caused by the movemement of object or camera.



- Optical flow has many applications in areas like :
- Structure from Motion
- Video Compression
- Video Stabilization

- `cv2.calcOpticalFlowPyrLK()`
- (Lucas-Kanade method)

Background Subtraction

Code Pack 28

1. ~~Introduction to OpenCV with Python~~
2. ~~Core Operations~~
3. ~~Image Processing in OpenCV~~
4. ~~Feature Detection and Description~~
5. Video Analysis

Coding Bootcamp Code in Python

INTRODUCTION TO OPENCV: CAMERA CALIBRATION AND 3D RECONSTRUCTION

Camera Calibration

- Cheap cameras introduces a lot of distortion to images. Two major distortions are radial distortion and tangential distortion

It his necessary to remap the reading Matrix

- `cv2.findChessboardCorners` – Dummy Test
- `cv2.calibrateCamera()`
- `cv2.getOptimalNewCameraMatrix()`

Pinhole Camera Model: Camera Matrix

- **Camera matrix**: a 3x4 matrix that transforms from 3D scene points (x, y, z) to 2D screen points $(x'/w', y'/w')$:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- How many free parameters in the camera matrix?

Pinhole Camera Model

- Can rewrite the camera matrix in terms of **intrinsic** and **extrinsic** parameters.

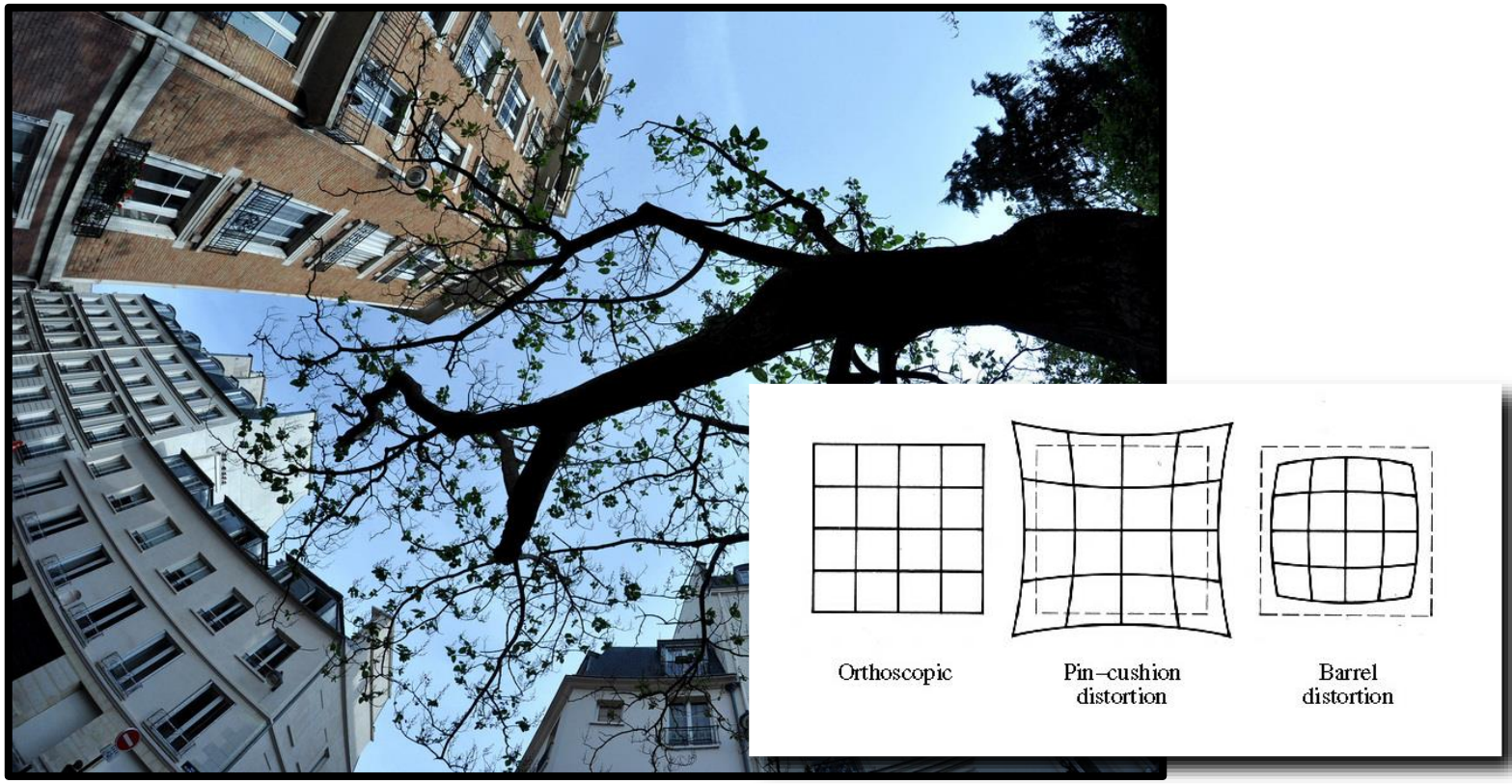
$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\mathbf{K} = \begin{bmatrix} \alpha_x & \gamma & u_0 \\ 0 & \alpha_x & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

Intrinsic matrix

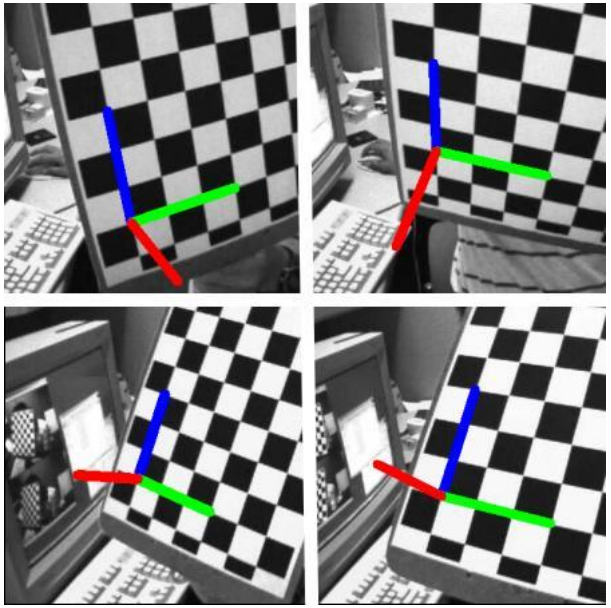
- How many intrinsic parameters?
- How many extrinsic parameters?
- How many parameters in total?

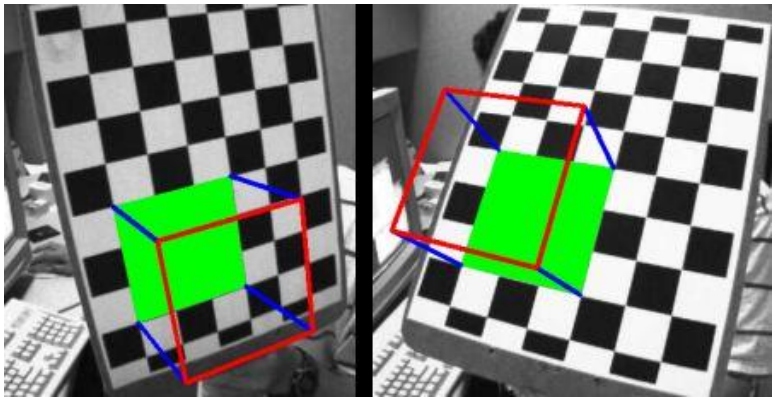
Real Camera Model: Radial Distortion



Pose Estimation

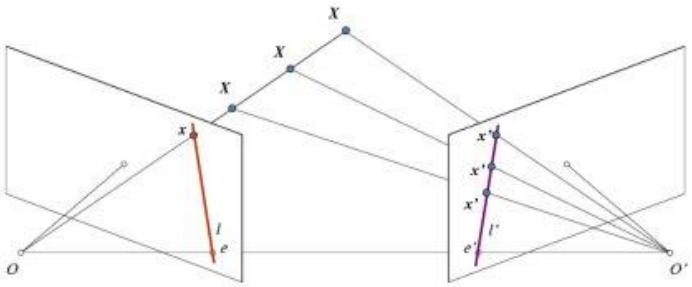
- Use the camera matrix and the calib3d module to create some 3D effects in image

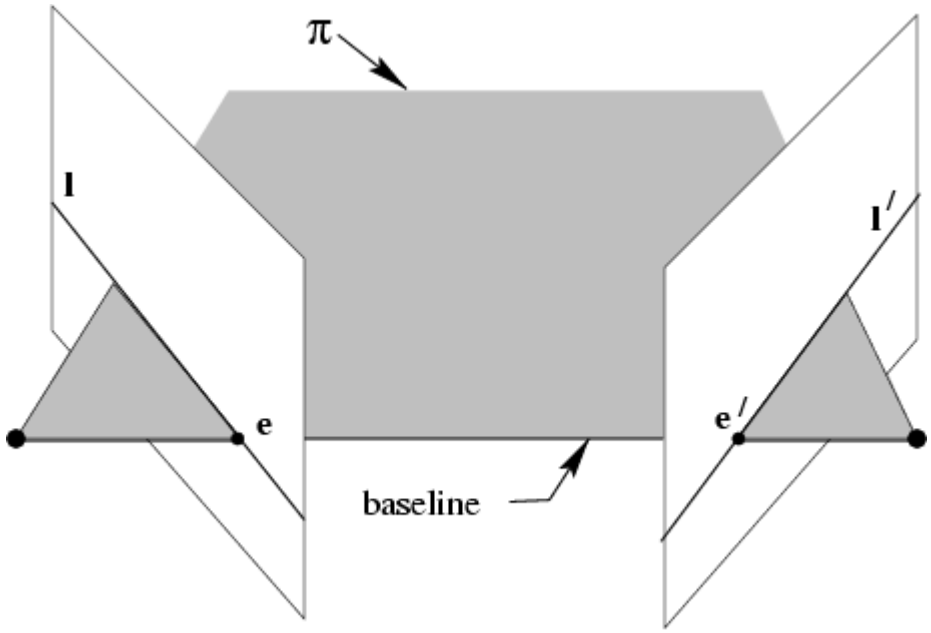




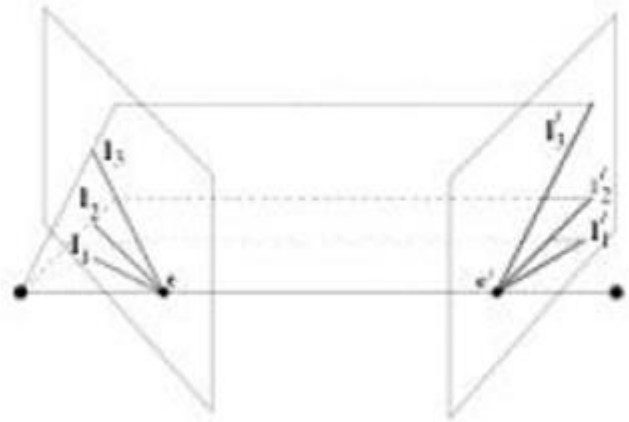
Epipolar Geometry

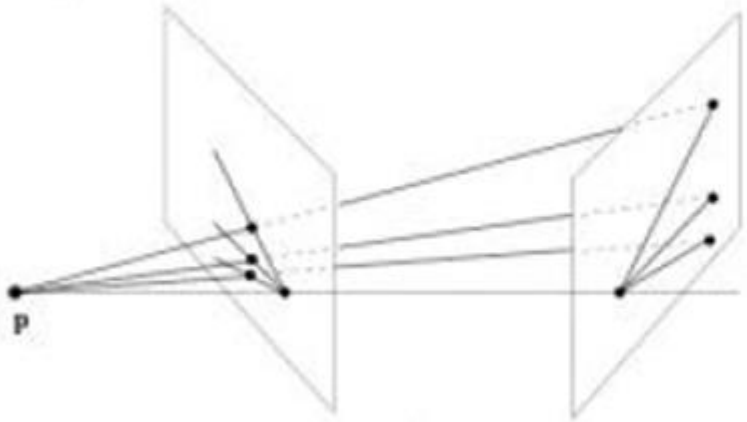
- Baseline: connects two camera centers
- Epipole: point of intersection of baseline with image plane
- Epipole: image in one view of the camera center of the other view.

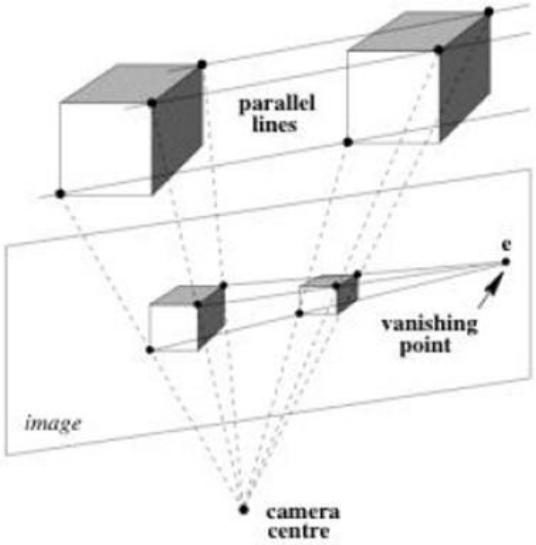




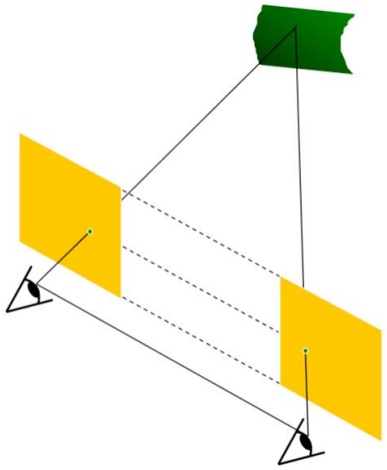
Epipolar Geometry – More examples





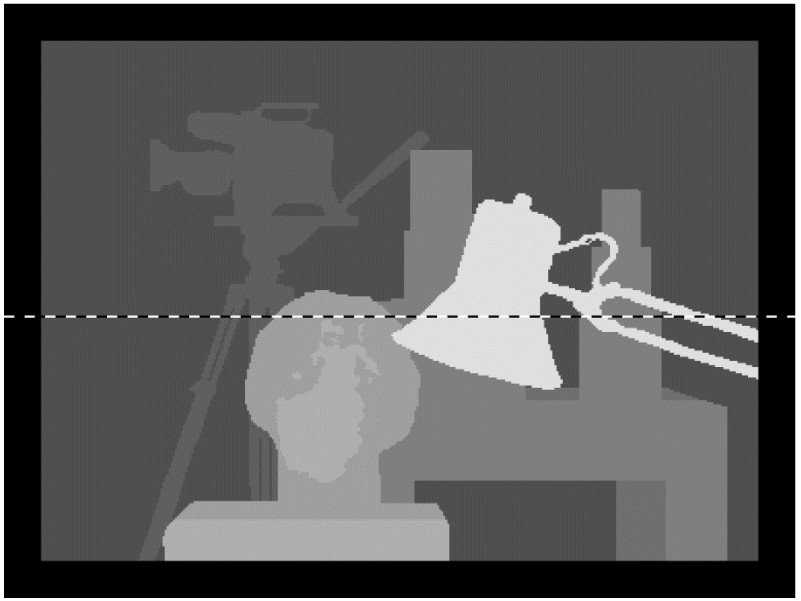


Stereo Images



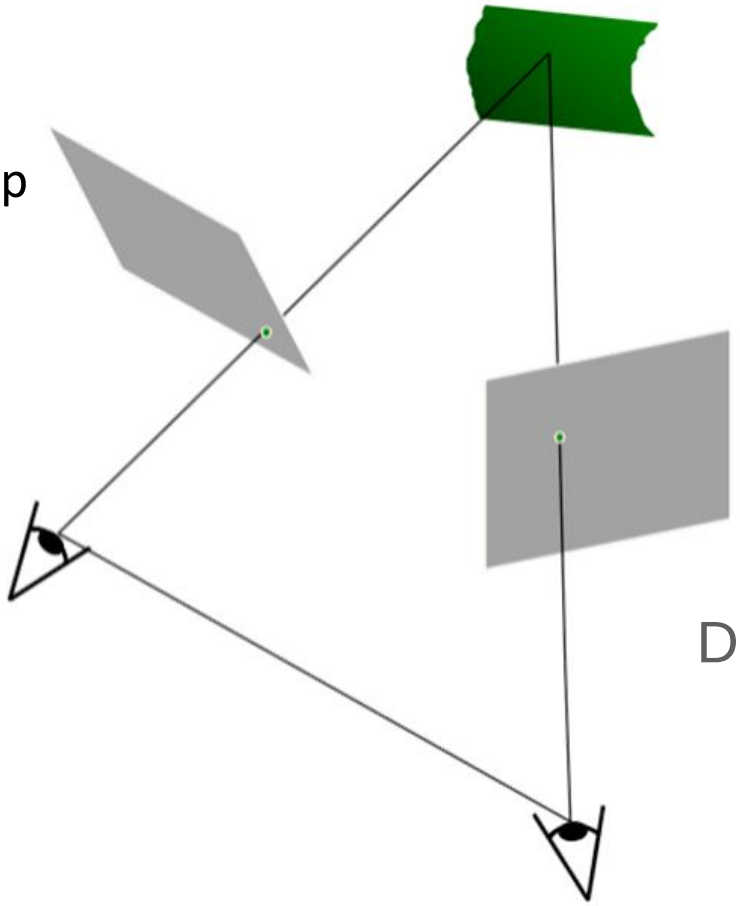
Depth Map from Stereo Images

- Below image contains the original image (left) and its disparity map (right). As you can see, result is contaminated with high degree of noise.
- By adjusting the values of numDisparities and blockSize, you can get better results.



Stereo Rectification

- Depth Map



Depth Map

Code Pack 28

1. ~~Introduction to OpenCV with Python~~
2. ~~Core Operations~~
3. ~~Image Processing in OpenCV~~
4. ~~Feature Detection and Description~~
5. ~~Video Analysis~~
6. Camera Calibration and 3D Reconstruction

Coding Bootcamp Code in Python

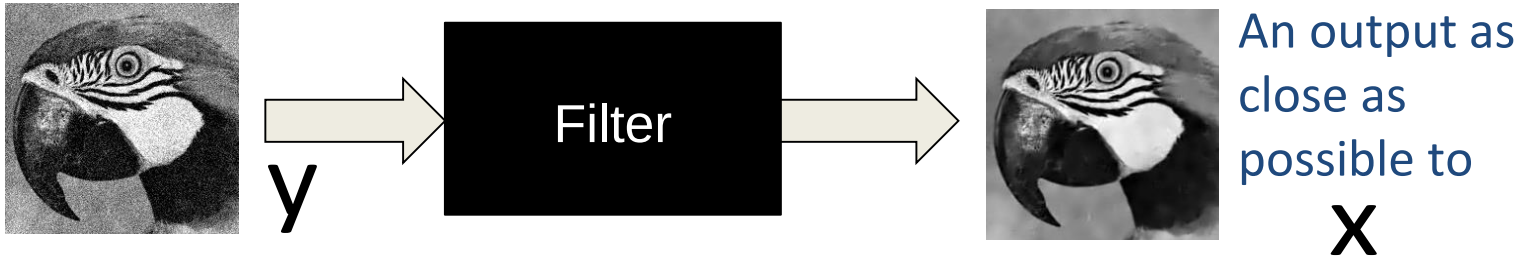
INTRODUCTION TO OPENCV: COMPUTATIONAL PHOTOGRAPHY

Image Denoising

This is the “simplest” inverse problem

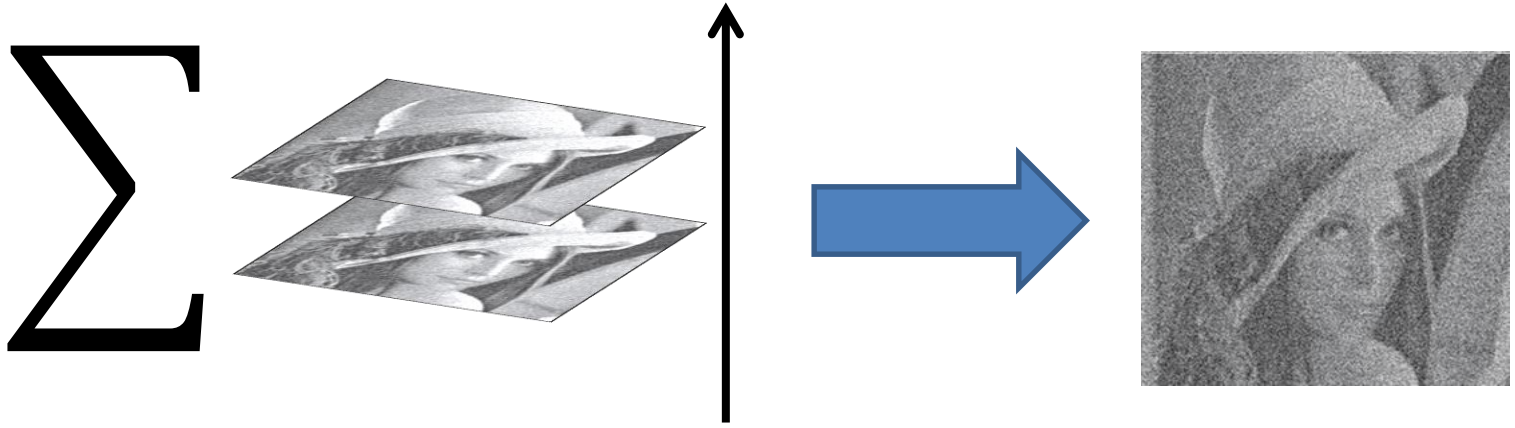
$$y = x + e$$

Measured Clean Noise (AWGN)



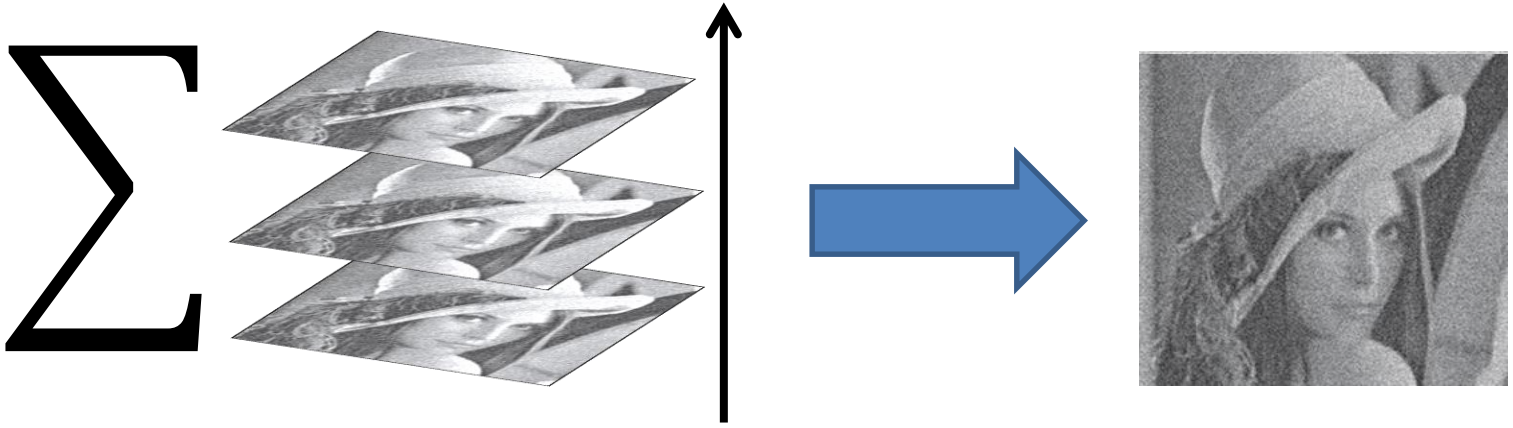
Temporal Denoising

Average multiple images over time



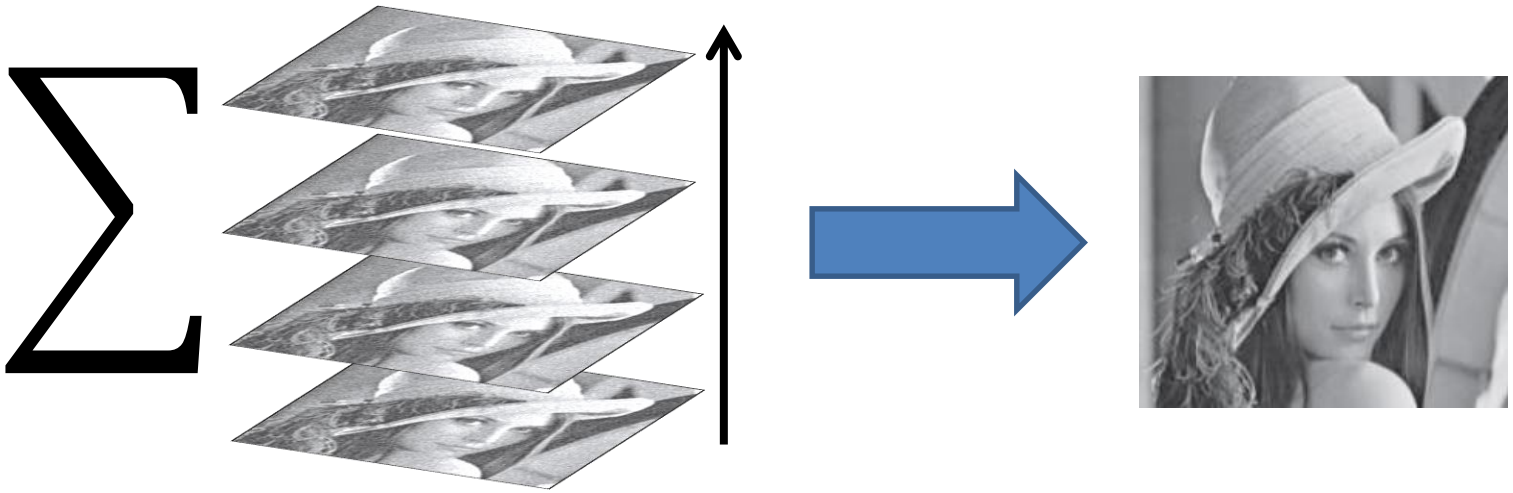
Temporal Denoising

Average multiple images over time

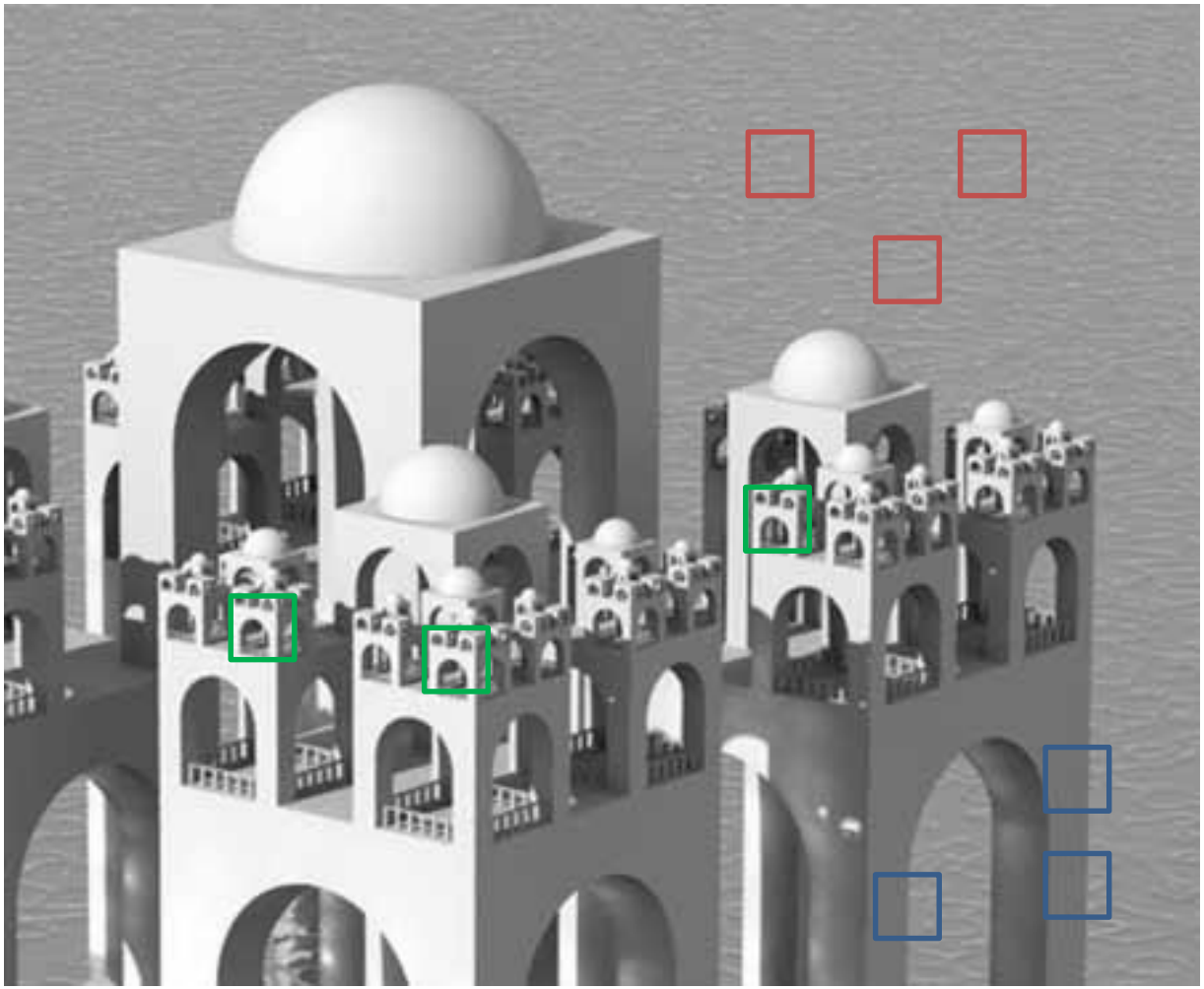


Temporal Denoising

Average multiple images over time



Redundancy in natural images



Stronger Techniques

- Classical Denoising
  - Spatial Methods
  - Transform Methods

- State-of-the-art Methods
  - GSM – Gaussian Scale Mixture
  - NLM – Non-local means
  - BM3D – Block Matching 3D collaborative filtering

Image Denoising in OpenCV

OpenCV provides 4 variations of this technique.

1. `cv2.fastNlMeansDenoising()` - works with a single grayscale images
2. `cv2.fastNlMeansDenoisingColored()` - works with a color image.
3. `cv2.fastNlMeansDenoisingMulti()` - works with image sequence captured in short period of time (grayscale images)
4. `cv2.fastNlMeansDenoisingColoredMulti()` - same as above, but for color images.

Common arguments are:

- `h` : parameter deciding filter strength. Higher h value removes noise better, but removes details of image also. (**10 is ok**)
- `hForColorComponents` : same as h, but for color images only. (**normally same as h**)
- `templateWindowSize` : should be odd. (**recommended 7**)
- `searchWindowSize` : should be odd. (**recommended 21**)

Image Inpainting

- Inpainting is the process of reconstructing lost or deteriorated parts of images and videos.

Other example Image Inpanting

Image Inpainting in OpenCV

- Several algorithms were designed for this purpose and OpenCV provides 2 of them.
- Both can be accessed by the same function:

```
cv2.inpaint()
```

They are:

- `cv2.INPAINT_TELEA` – based on the paper “An Image Inpainting Technique Based on the Fast Marching Method” by Alexandru Telea in 2004
- `cv2.INPAINT_NS`. – based on the paper “Navier-Stokes, Fluid Dynamics, and Image and Video Inpainting” by Bertalmio, Marcelo, Andrea L. Bertozzi, and Guillermo Sapiro in 2001

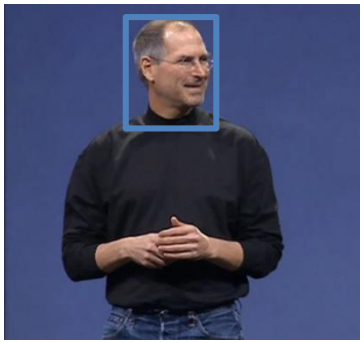
Code Pack 28

1. ~~Introduction to OpenCV with Python~~
2. ~~Core Operations~~
3. ~~Image Processing in OpenCV~~
4. ~~Feature Detection and Description~~
5. ~~Video Analysis~~
6. ~~Camera Calibration and 3D Reconstruction~~
7. Computational Photography

Coding Bootcamp Code in Python

INTRODUCTION TO OPENCV: OBJECT DETECTION

Face detection



Face Detection ≠ Face recognition

•Who is this?→

•←Where is the face?

Alfréd Haar

Face Detection - detect that is a human face

- Haar Classifier
- LBP Classifier

Face Recognition - detect that is that human

- Fisher
- Eigen
- LBPHF

Object Recognition Classifier

- A program that decides whether an image is a positive image (face image) or negative image (non-face image).

- A classifier is trained on hundreds of thousands of face and non-face images to learn how to classify a new image correctly.

Haar Cascade

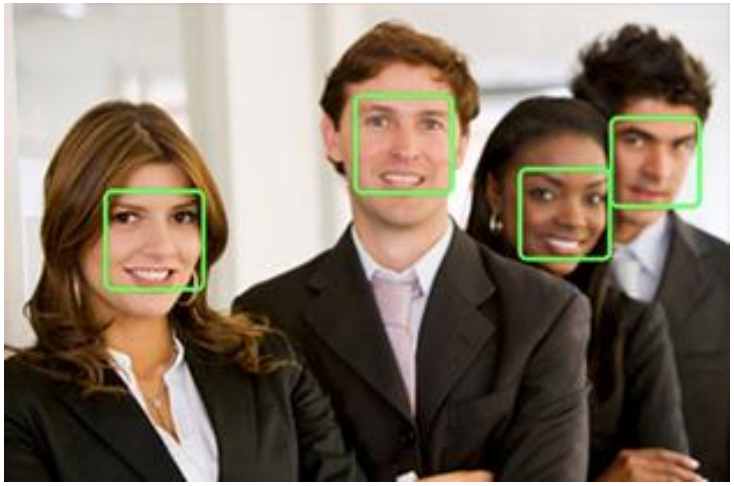
- A "cascade" is a series of "Haar-like features" that are combined to form a classifier

- Haar Cascade = Classifier

Face Detection

- The problem of face detection is:
  Given an image, say if it contains faces or not.

- The idea of face detection in computer vision is to let the computer learn to detect faces in images, just as a human can do.



Difficulties - Changing lightening

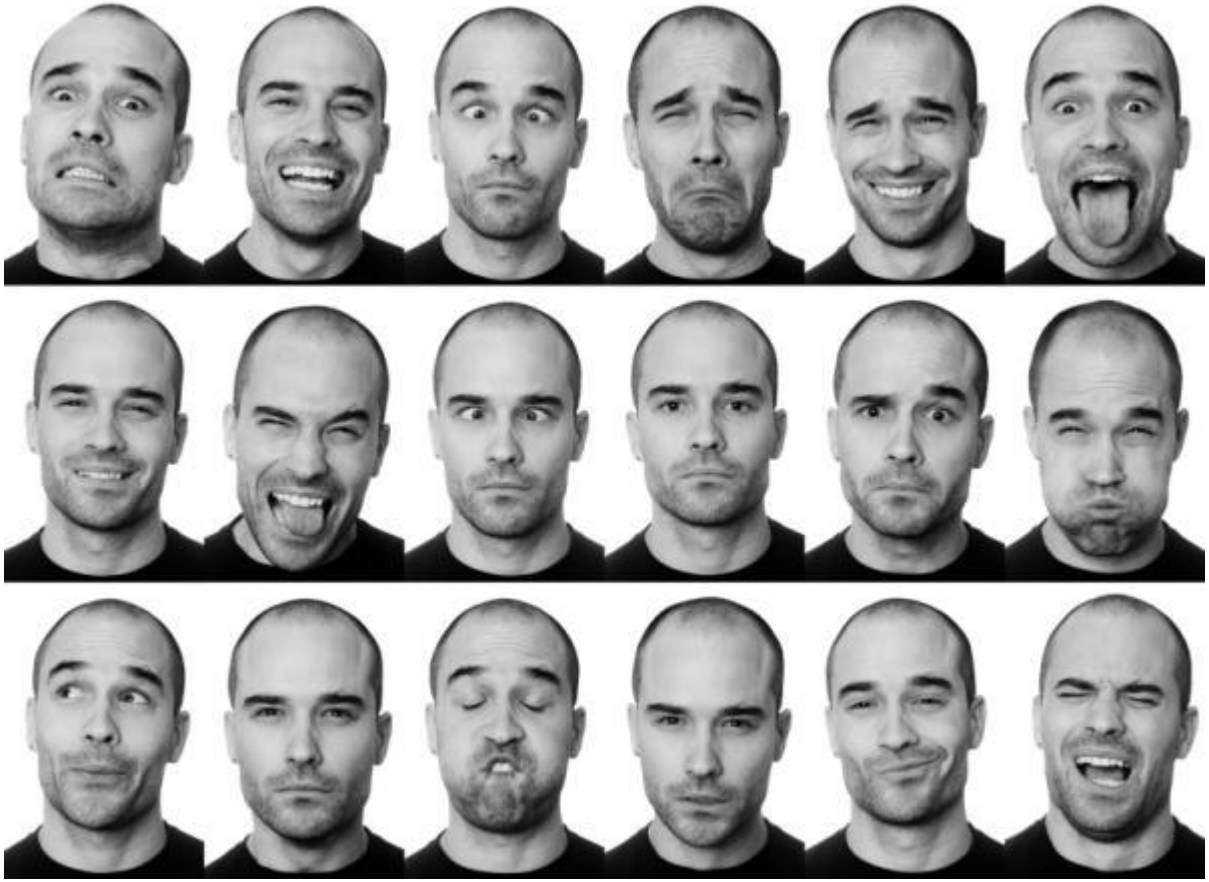
- Affects color, facial feature

Difficulties - Skin Tone

- Large variety of skin tones.

Difficulties - Facial Expressions

- Affects shape of face and its features



Difficulties – Scaling and Angles

Difficulties - Obstructions

- Obstruction of facial features







Paul Viola and Michael Jones - 2001

- The Viola–Jones object detection algorithm is the first object detection framework to provide competitive object detection rates in real-time proposed.
- First with speed!

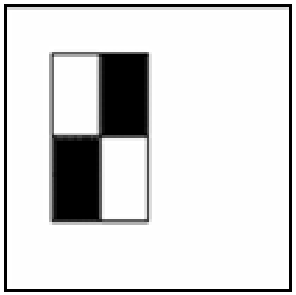
- Although it can be trained to detect a variety of object classes, it was motivated primarily by the problem of face detection.

- The algorithm has 4 stages:

1. Haar Feature Selection
2. Creating an Integral Image
3. Adaboost Training
4. Final Classifier = Cascading Classifiers

Rectangle Features (or Haar-like Features)

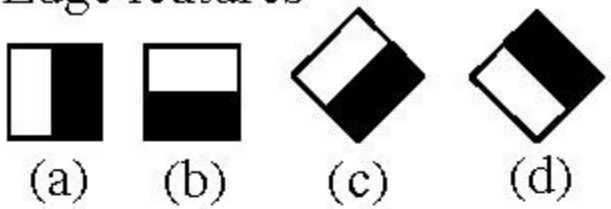
- It look for features inside 24x24 pixels window
- Each feature contains black & white rectangles, the feature value is defined by:
  - $\sum$ *(pixels in white area)* – $\sum$ *(pixels in black area)*
  - Other explanation: correlation of the image with a mask with 1 in pixels of white areas, and -1 in pixels of black areas



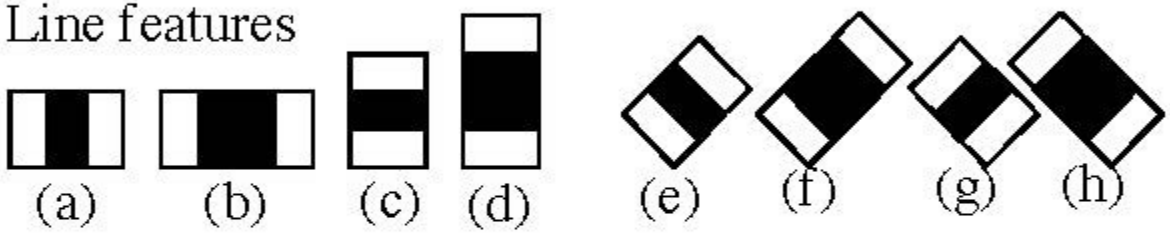
0	0	0	0	0
0	**1**	**-1**	0	0
0	**-1**	**1**	0	0
0	0	0	0	0

Haar-Like Features

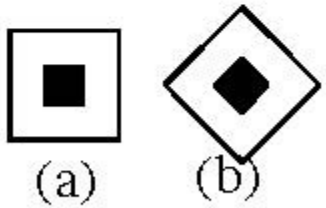
1. Edge features



2. Line features



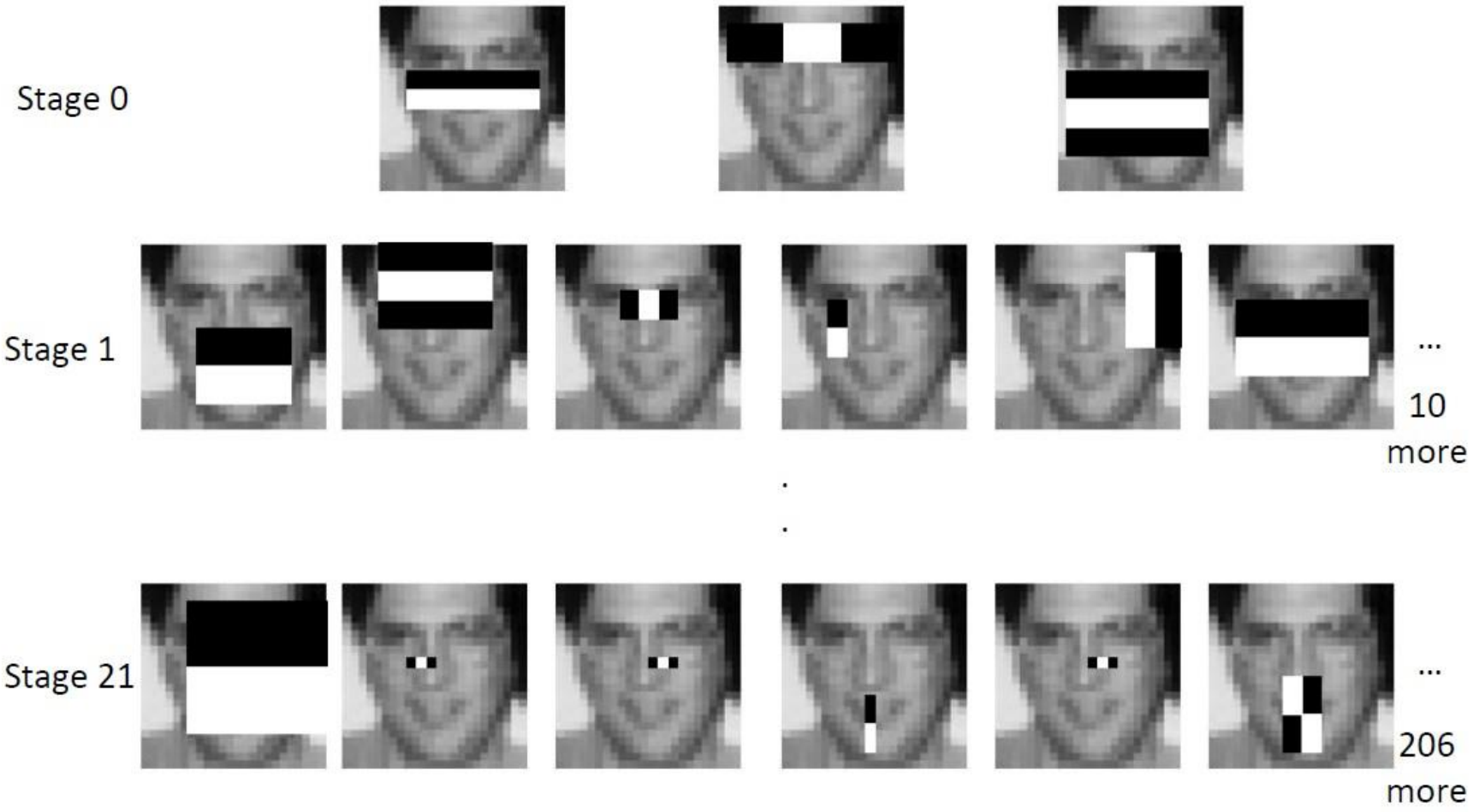
3. Center-surround features



4. Special diagonal line feature used in [3,4,5]

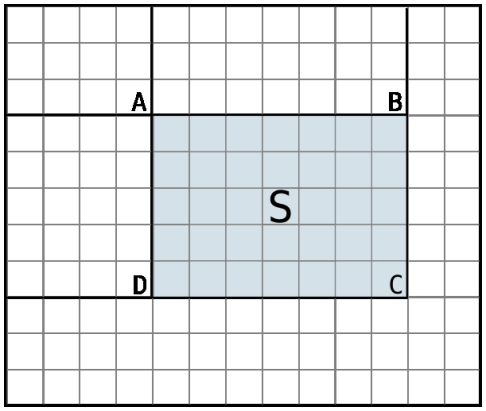
Examples of how the Haar-like Features correspond to common facial features.

1. Haar Feature Selection



2. Creating an Integral Image

- A "summed area table" is created in a single pass across the image.
- The sum of any region in the image can be computed by a single formula.



Original

5	**2**	**3**	4	1
1	**5**	**4**	2	3
2	2	1	3	4
3	5	6	4	5
4	1	3	2	6

Integral

5	7	10	14	15
6	13	**20**	26	30
8	17	25	34	42
11	25	39	52	65
15	30	47	62	81

$5 + 2 + 3 + 1 + 5 + 4 = 20$

Original

5	2	3	4	1
1	**5**	**4**	**2**	3
2	**2**	**1**	**3**	4
3	5	6	4	5
4	1	3	2	6

$5 + 4 + 2 + 2 + 1 + 3 = 17$

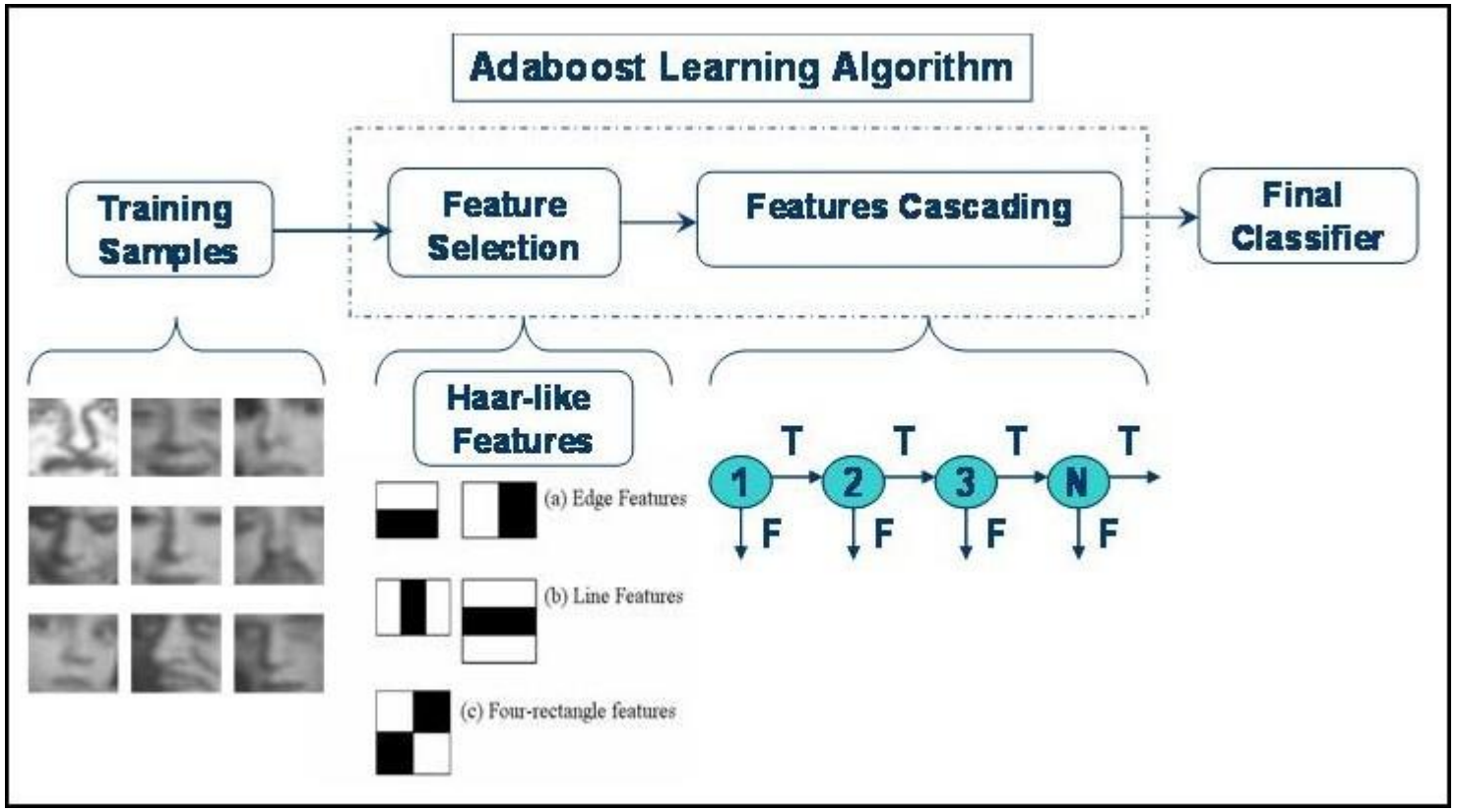
Integral

5	7	10	**14**	15
6	13	20	26	30
8	17	25	**34**	42
11	25	39	52	65
15	30	47	62	81

$34 - 14 - 8 + 5 = 17$

3. AdaBoost (Adaptive Boosting) Training

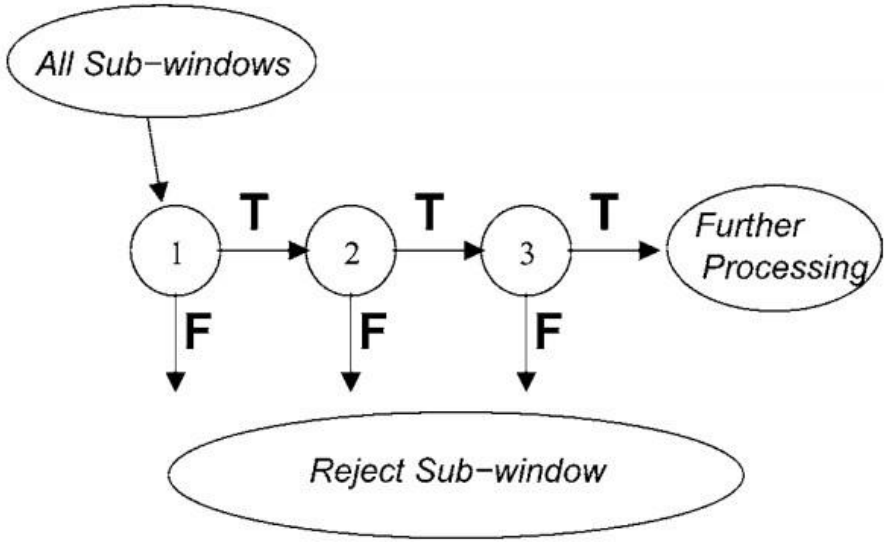
- Adaboost tries out multiple weak classifiers over several rounds, selecting the best weak classifier in each round and combining the best weak classifiers to create a strong classifier



4. Final Classifier = Cascading Classifiers

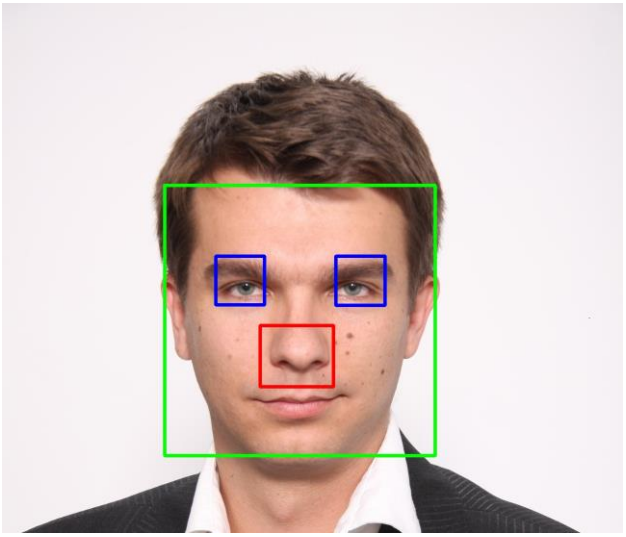
- Haar cascades consists of a series of weak classifiers - those barely better than 50% correct

- If an area passes a single classifier, go to the next classifier; otherwise, area doesn't match



OpenCV already contains many pre-trained classifiers

- Eyes
- Nose
- Mouth
- Face
- Smile
- Body
- Profile face
- Frontal face
- ……



Code Pack 28

1. ~~Introduction to OpenCV with Python~~
2. ~~Core Operations~~
3. ~~Image Processing in OpenCV~~
4. ~~Feature Detection and Description~~
5. ~~Video Analysis~~
6. ~~Camera Calibration and 3D Reconstruction~~
7. ~~Computational Photography~~
8. Object Detection