Image Processing

OpenCV

Intel® OPEN SOURCE COMPUTER VISION LIBRARY

Based in part on slides by Victor Eruhimov, Itseez

Goals

Develop a universal toolbox for research and development in the field of Computer Vision

We will talk about:

- Algorithmic content
- Technical content
- Examples of usage
- Trainings

OpenCV algorithms

OpenCV Functionality (more than 350 algorithms)

Basic structures and operations

Image Analysis

Structural Analysis

Object Recognition

Motion Analysis and Object Tracking

3D Reconstruction

Basic Structures and Operations

File IO and capturing

Multidimensional array operations

Dynamic structures operations

Drawing primitives

Utility functions

Basic Structures and Operations

Multidimensional array operations include operations on images, matrices and histograms. In the future, when I talk about image operations, keep in mind that all operations are applicable to matrices and histograms as well. Dynamic structures operations concern all vector data storages. They will be discussed in detail in the Technical Section. Drawing primitives allows not only to draw primitives but to use the algorithms for pixel access. Utility functions, in particular, contain fast implementations of useful math functions

File IO and Capturing

Simple OpenCV example:

```
#include <stdio.h>
#include <opencv2/opencv.hpp>
using namespace cv;
int main(int argc, char** argv ) {
  if ( argc != 2 ) { printf("usage: DisplayImage.out
    <Image_Path>\n"); return -1; }
  Mat image;
  image = imread( argv[1], 1 );
  if ( !image.data ) { printf("No image data \n"); return -1; }
  namedWindow("Display Image", WINDOW_AUTOSIZE );
  imshow("Display Image", image);
  waitKey(0);
}
```

File IO and Capturing

CMake supports OpenCV as well so you can use a configuration file similar to using VTK:

```
cmake_minimum_required(VERSION 2.8)
project( DisplayImage )
find_package( OpenCV REQUIRED )
include_directories(
  ${OpenCV_INCLUDE_DIRS} )
add_executable( DisplayImage
  DisplayImage.cpp )
target_link_libraries( DisplayImage
  ${OpenCV_LIBS} )
```

File IO and Capturing

OpenCV supports a long list of file formats already that it is capable of loading directly. These include (via imdecode):

- Windows bitmaps - *.bmp, *.dib (always supported)
- JPEG files - *.jpeg, *.jpg, *.jpe (see the Notes section)
- JPEG 2000 files - *.jp2 (see the Notes section)
- Portable Network Graphics - *.png (see the Notes section)
- Portable image format - *.pbm, *.pgm, *.ppm (always supported)
- Sun rasters - *.sr, *.ras (always supported)
- TIFF files - *.tiff, *.tif (see the Notes section)

File IO and Capturing

As you saw from the example, images are typically represented as matrices, i.e. a 2x2 configuration of pixels, in OpenCV.

As data structure, OpenCV provides cv::Mat to store those images

File IO and Capturing

OpenCV also supports various video codecs. There is native support for:

AVI	'DIB '	RGB(A)	Uncompressed RGB, 24 or 32 bit
AVI	'I420'	RAW I420	Uncompressed YUV, 4:2:0 chroma subsampled
AVI	'IYUV'	RAW I420	identical to I420

Also, OpenCV can be compiled with support for ffmpg, which supports various different formats, including:

H.264, MJPG, MPEG, Quicktime, …

File IO and Capturing

OpenCV can also be used to capture images from recording devices, such as cameras, directly.

Both reading and capturing images are encapsulated in the VideoCapture class of OpenCV.

To open a file or get data from a capture devices use

```
bool VideoCapture::open(const string&
                              filename)
```

```
bool VideoCapture::open(int device)
```

You can release the device/close the file via

```
void VideoCapture::release()
```

File IO and Capturing

When recording from a capture device, you can grab and then retrieve the image:

```
bool VideoCapture::grab()
bool VideoCapture::retrieve(Mat& image,
                            int channel=0)
```

For reading the next image from an already opened file simply use the `read` method:

```
bool VideoCapture::read(Mat& image)
```

Alternatively, you can use the usual C++ stream operators.

File IO and Capturing

After that, you can simply apply any image processing filters that are needed and then show the image via

```
void imshow(const string& winname,
            InputArray mat)
```

Alternatively, you can convert the image and pass it onto VTK using the code fragment on the next slides.

File IO and Capturing

```
void fromMat2Vtk( cv::Mat src,
                  vtkImageData* dest ) {
  vtkImageImport *importer =
   vtkImageImport::New();
  Mat frame;
  cvtColor( src, frame, COLOR_BGR2RGB);
  if (dest) { importer->SetOutput( dest ); }
  importer->SetDataSpacing( 1, 1, 1 );
  importer->SetDataOrigin( 0, 0, 0 );
  importer->SetWholeExtent( 0, frame.size().width-
  1, 0, frame.size().height-1, 0, 0 );
```

File IO and Capturing

```
  importer->SetDataExtentToWholeExtent();
  importer->SetDataScalarTypeToUnsignedChar();
  importer->SetNumberOfScalarComponents(
    frame.channels() );
  importer->SetImportVoidPointer( frame.data );
  importer->Update();
}
```

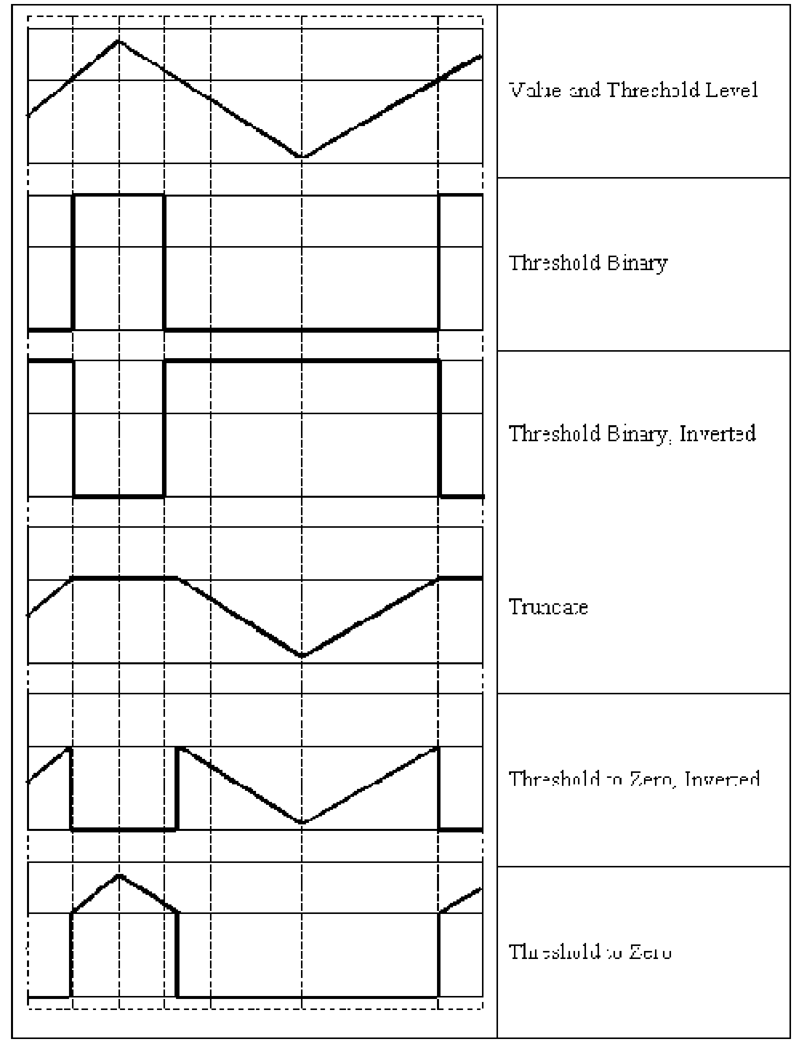
Image Analysis

- Thresholds
- Statistics
- Pyramids
- Morphology
- Distance transform
- Flood fill
- Feature detection
- Contours retrieving

Image Thresholding

Fixed threshold;

Adaptive threshold;



Adaptive Thesholding

Fixed thresholding may not work well where image has different lighting conditions in different areas. In that case, we go for adaptive thresholding. In this, the algorithm calculates the threshold for a small region of the image. So we get different thresholds for different regions of the same image and it gives us better results for images with varying illumination:

`cv2.ADAPTIVE_THRESH_MEAN_C`

threshold value is the mean of neighborhood area.

`cv2.ADAPTIVE_THRESH_GAUSSIAN_C`

threshold value is the weighted sum of neighborhood values where weights are a Gaussian window.

Adaptive Thesholding

Example:

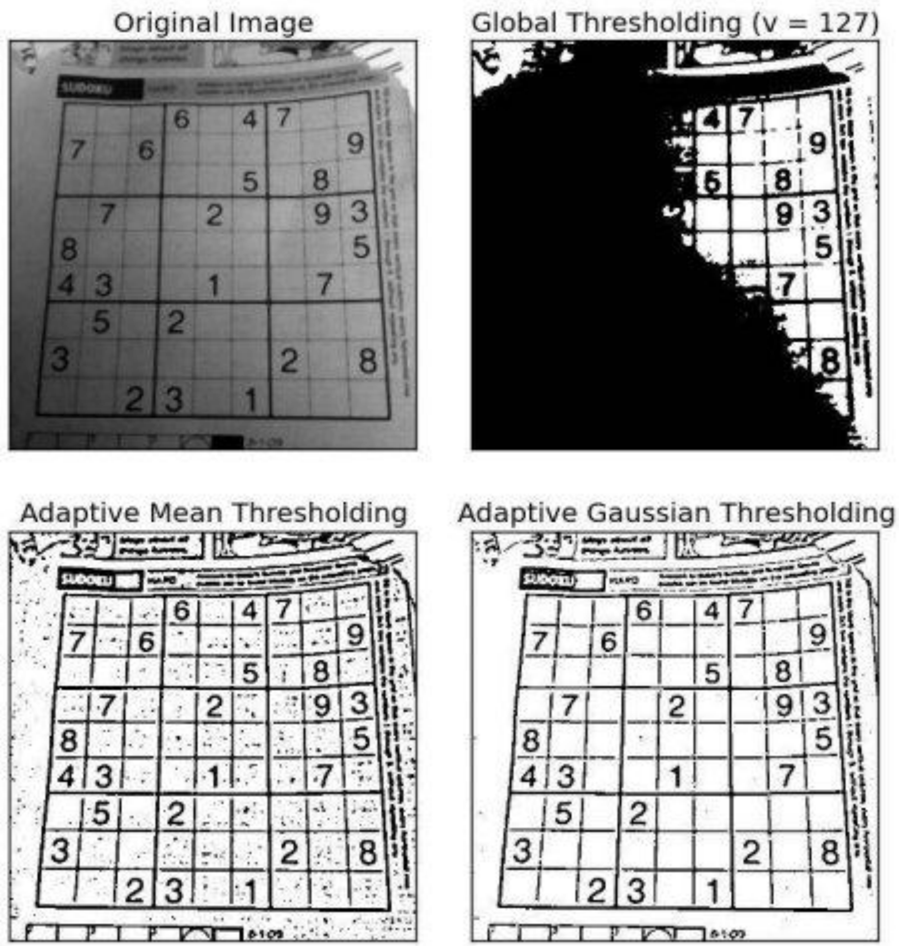
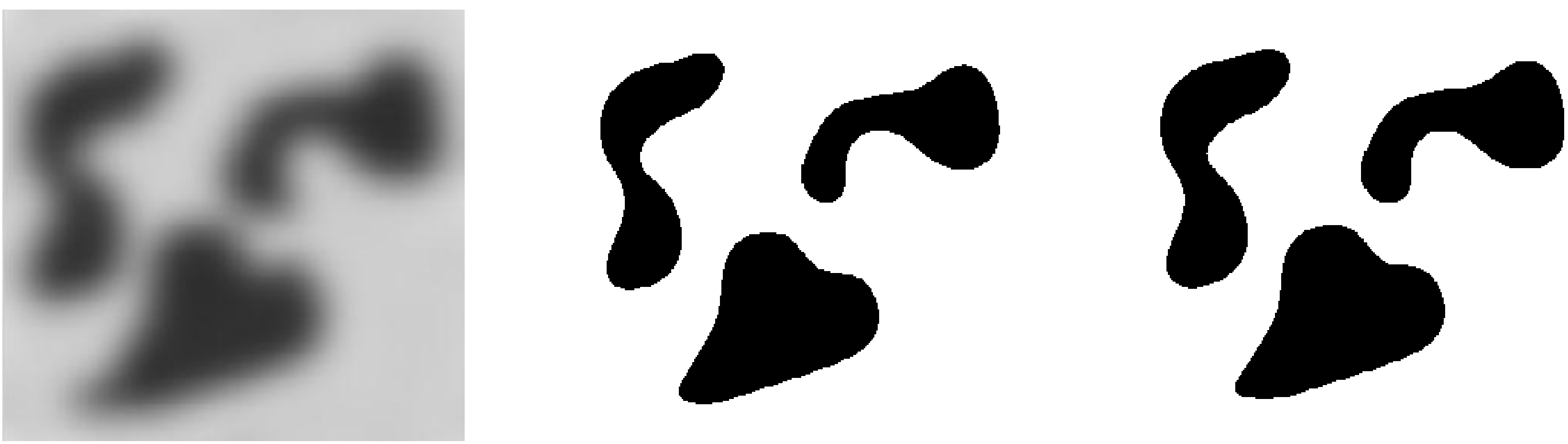


Image Thresholding Examples

Source picture ***Fixed threshold*** ***Adaptive threshold***



Statistics

min, max, mean value, standard deviation over the image

Norms C, L1, L2

Multidimensional histograms

Spatial moments up to order 3 (central, normalized, Hu)

In addition to simple norm calculation, there is a function that finds the norm of the difference between two images.

Multidimensional Histograms

Histogram operations : calculation, normalization, comparison, back project

Histograms types:

- ✓ Dense histograms
- ✓ Signatures (balanced tree)

EMD (earth mover distance) algorithm:

The EMD computes the distance between two distributions (sets of weighted points), which are represented by signatures.

The signatures are sets of weighted features that capture the distributions. The features can be of any type and in any number of dimensions, and are defined by the user.

The EMD is defined as the minimum amount of work needed to change one signature into the other

EMD – a method for the histograms comparison

$$p_i \in P,\ 1 \le i \le |P|,\ q_j \in Q,\ 1 \le j \le |Q|, - \ two\ historams$$

$$EMD(P,Q) = \frac{\sum_{i,j} f_{ij} \cdot d(p_i, q_j)}{\sum_{i,j} f_{ij}},$$

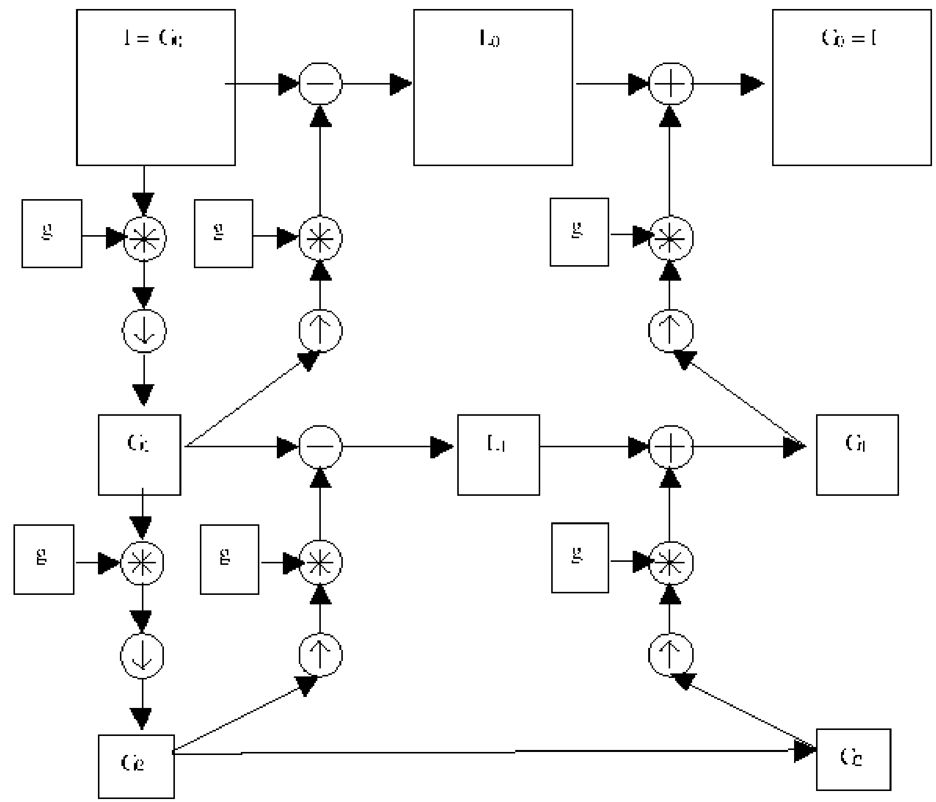
$$f_{ij} - weight\ \ coefficien\ \ ts,$$

$$d(p_i, q_j) - the\ \ distance\ \ between\ \ the\ elements\ \ p_i\ and\ \ q_j.$$

Image Pyramids

Gaussian and Laplacian pyramids

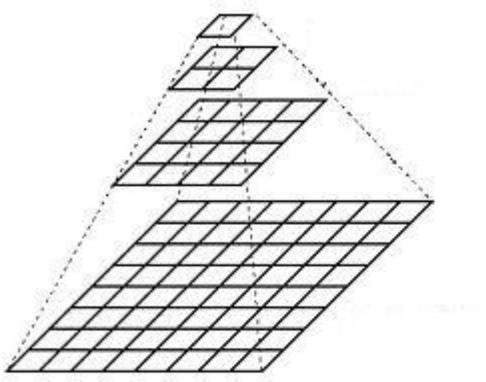
Image segmentation by pyramids



Gaussian

Use a Gaussian filter to blur image or down-sample it. A Gaussian filter simply uses the Gaussian distribution function to derive a filter matrix that describes how neighboring pixels are averaged.

$$\frac{1}{16}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$



Laplacian

Laplacian Pyramids are formed from the Gaussian Pyramids. There is no exclusive function for that. Laplacian pyramid images are like edge images only. Most of its elements are zeros. They are used in image compression. A level in Laplacian Pyramid is formed by the difference between that level in Gaussian Pyramid and expanded version of its upper level in Gaussian Pyramid.

Laplacian function:

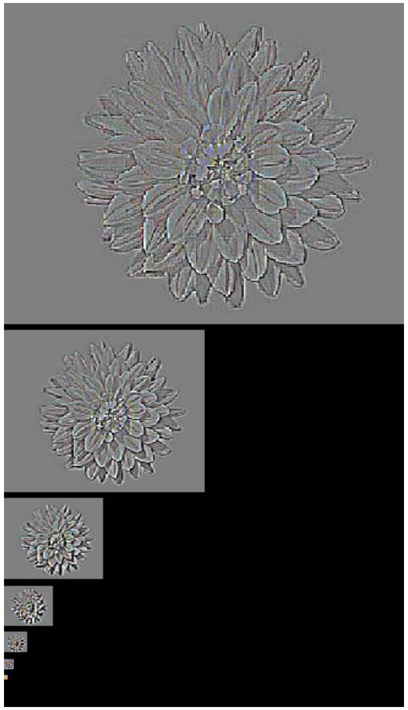
$$\mathrm{Laplace}(f) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Filter kernel for Laplacian:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

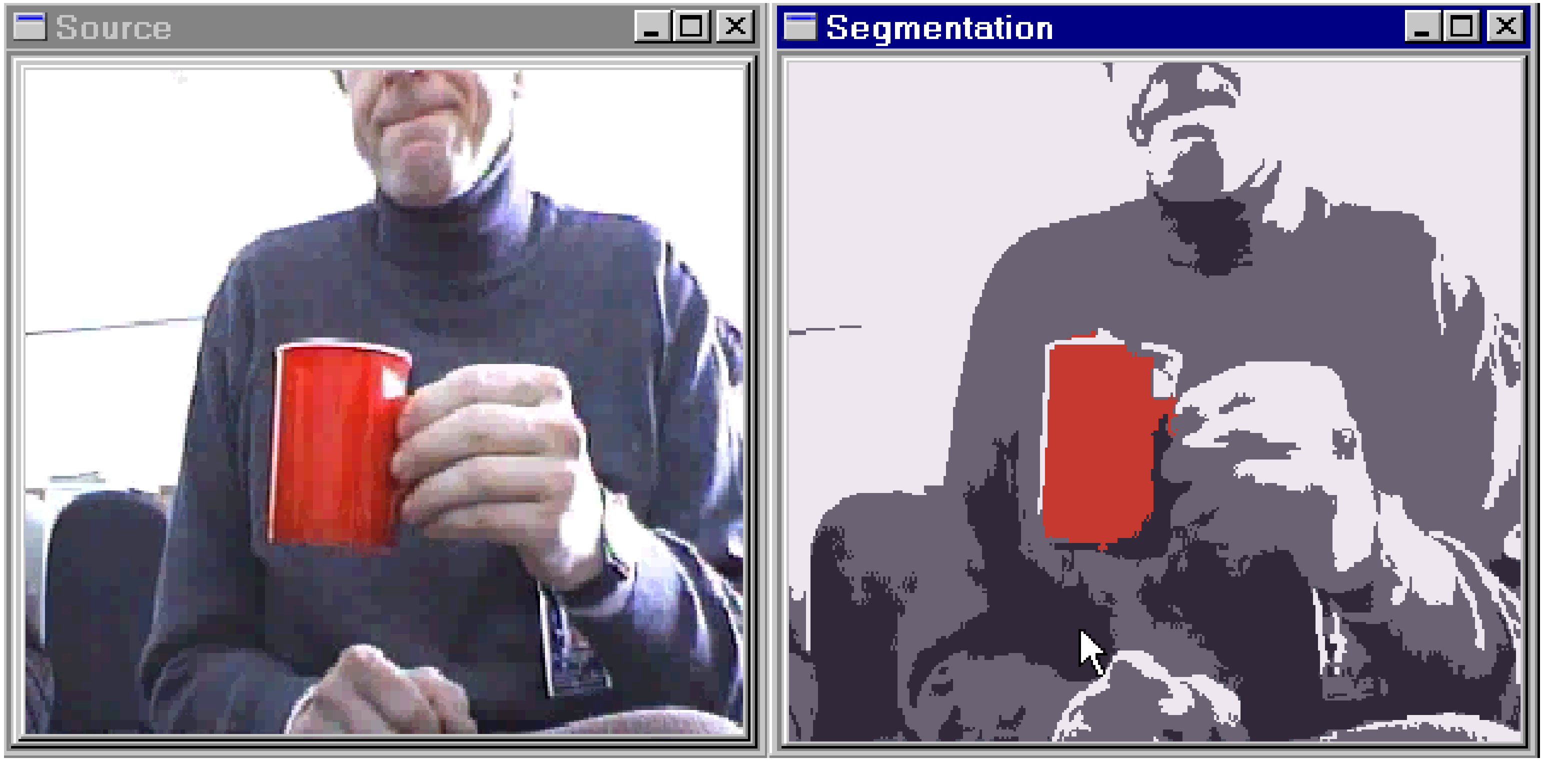
Image Pyramids

Gaussian and Laplacian



Pyramid-based color segmentation

On still pictures **And on movies**



Morphological Operations

Two basic morphology operations using structuring element:

- ✓ erosion
- ✓ dilation

More complex morphology operations:

- ✓ opening
- ✓ closing
- ✓ morphological gradient
- ✓ top hat
- ✓ black hat

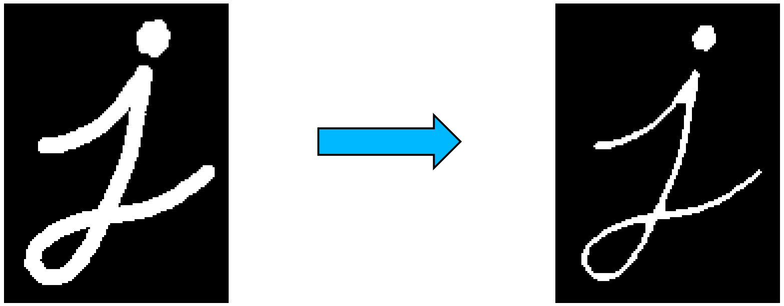
Morphological Operations

Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images. It needs two inputs, one is our original image, second one is called **structuring element** or **kernel** which decides the nature of operation. Two basic morphological operators are Erosion and Dilation. Then its variant forms like Opening, Closing, Gradient etc also comes into play. We will see them one-by-one with help of following image:

Morphological Operations

Erosion

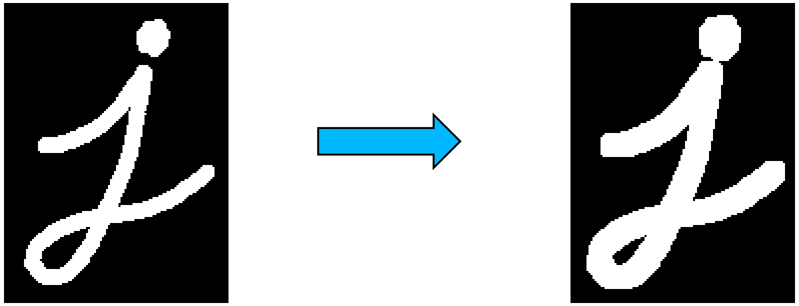
The basic idea of erosion is just like soil erosion only, it erodes away the boundaries of foreground object (Always try to keep foreground in white). So what it does? The kernel slides through the image (as in 2D convolution). A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero).



Morphological Operations

Dilation

It is just opposite of erosion. Here, a pixel element is '1' if atleast one pixel under the kernel is '1'. So it increases the white region in the image or size of foreground object increases. Normally, in cases like noise removal, erosion is followed by dilation. Because, erosion removes white noises, but it also shrinks our object. So we dilate it. Since noise is gone, they won't come back, but our object area increases. It is also useful in joining broken parts of an object.



Morphological Operations

Opening

Opening is just another name of **erosion followed by dilation**. It is useful in removing noise, as we explained above. Here we use the function,**cv2.morphologyEx()**

Morphological Operations

Closing

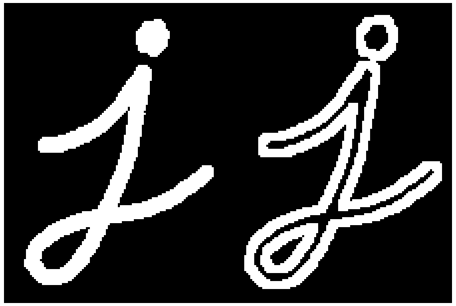
Closing is reverse of Opening, **Dilation followed by Erosion**. It is useful in closing small holes inside the foreground objects, or small black points on the object.

Morphological Operations

Morphological Gradient

It is the difference between dilation and erosion of an image.

The result will look like the outline of the object.



Morphological Operations

Top Hat

It is the difference between input image and Opening of the image. Below example is done for a 9x9 kernel.

Morphological Operations

Black Hat

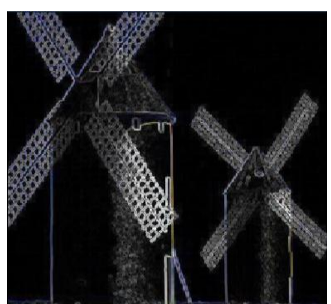
It is the difference between the closing of the input image and input image.

Morphological Operations Examples

Morphology - applying Min-Max. Filters and its combinations







Distance Transform

Calculate the distance for all non-feature points to the closest feature point

Two-pass algorithm, 3x3 and 5x5 masks, various metrics predefined

Flood Filling

Simple

Gradient

Original image   Tolerance interval ± 5   Tolerance interval ± 6

Feature Detection

Fixed filters (Sobel operator, Laplacian);

Optimal filter kernels with floating point coefficients (first, second derivatives, Laplacian)

Special feature detection (corners)

Canny operator

Hough transform (find lines and line segments)

Gradient runs

Sobel filter

Edges in an image become apparent when looking at the change between neighboring pixels. The Sobel filter is designed to detect just that. It approximates the image gradient, i.e. change of pixels, by applying a filter kernel in horizontal and vertical direction and then combining the results:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * I$$

$$G = \sqrt{G_x^2 + G_y^2}$$

Sobel filter: result

Canny Edge Detector

Multi-stage process:

Noise Reduction

Since edge detection is susceptible to noise in the image, first step is to remove the noise in the image with a 5x5 Gaussian filter.

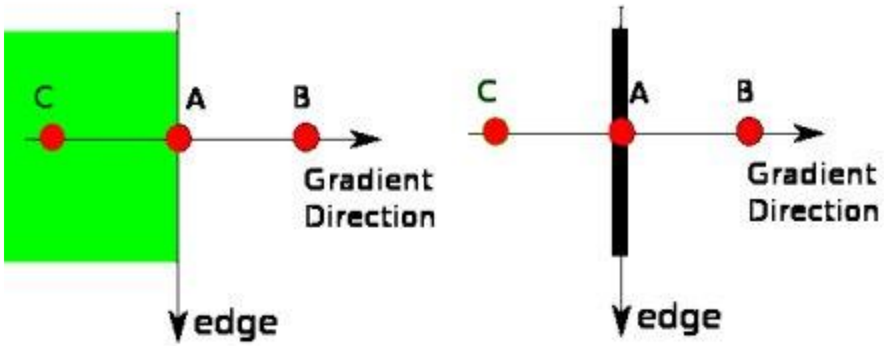
Finding Intensity Gradient of the Image

Smoothened image is then filtered with a Sobel kernel in both horizontal and vertical direction to get first derivative in horizontal direction (G_x) and vertical direction (G_y). From these two images, we can find edge gradient and direction for each pixel as $\sqrt{G_x^2 + G_y^2}$

Canny Edge Detector

Non-maximum Suppression

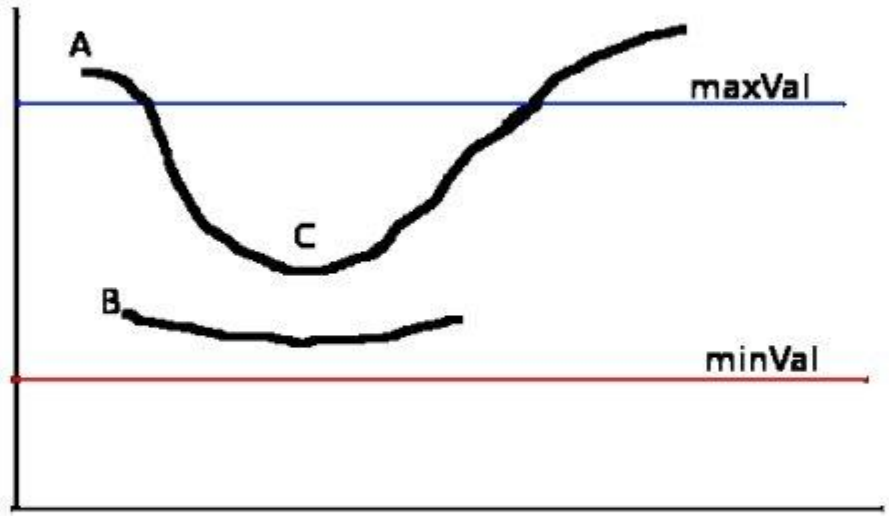
After getting gradient magnitude and direction, a full scan of image is done to remove any unwanted pixels which may not constitute the edge. For this, at every pixel, pixel is checked if it is a local maximum in its neighborhood in the direction of gradient.



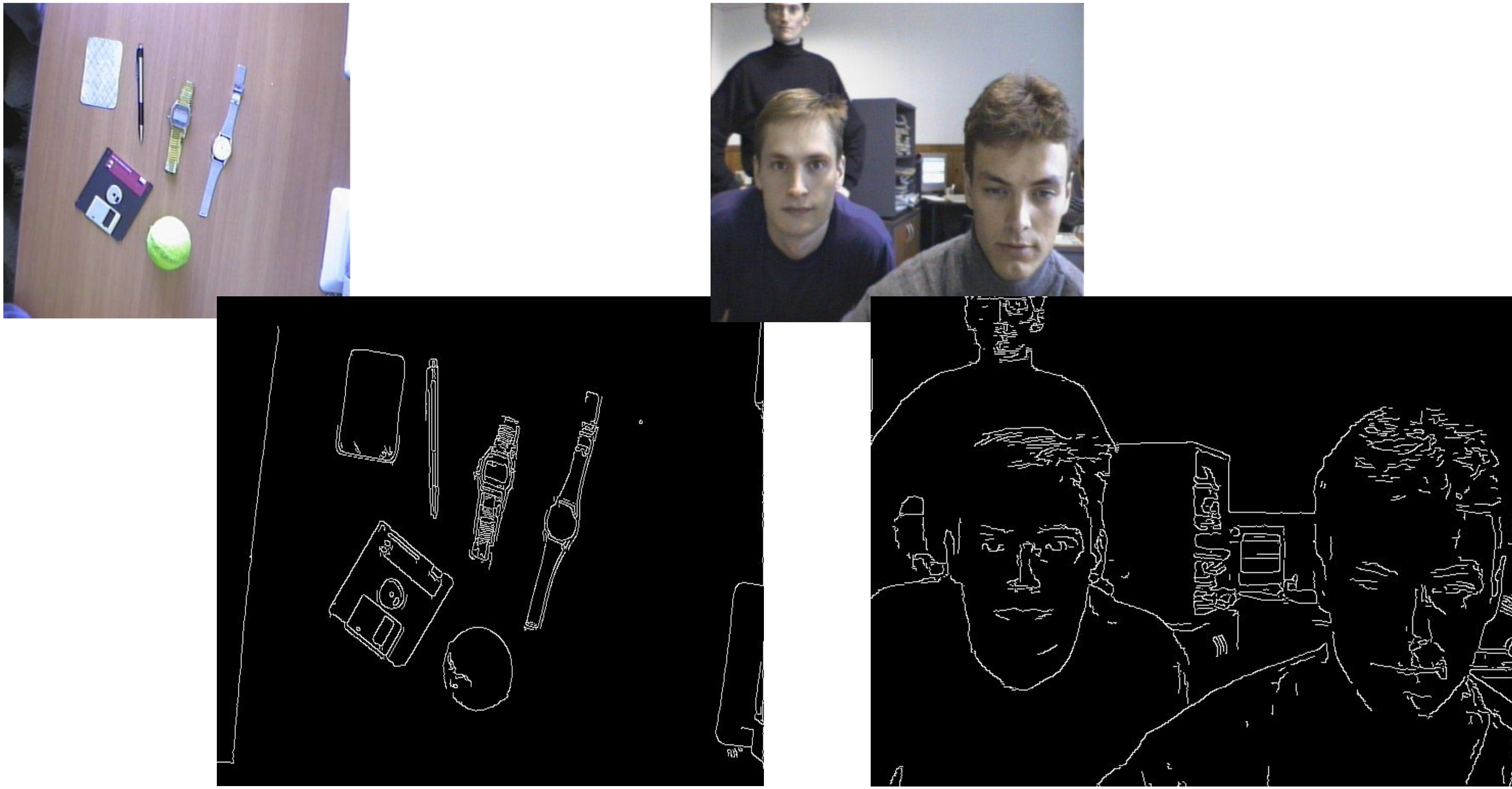
Canny Edge Detector

Hysteresis Thresholding

This stage decides which are all edges are really edges and which are not. For this, we need two threshold values, minVal and maxVal. Any edges with intensity gradient more than maxVal are sure to be edges and those below minVal are sure to be non-edges, so discarded. Those who lie between these two thresholds are classified edges or non-edges based on their connectivity. If they are connected to "sure-edge" pixels, they are considered to be part of edges. Otherwise, they are also discarded.



Canny Edge Detector



Harris Corner Detection

This algorithm basically finds the difference in intensity for a displacement of (u,v) in all directions:

$$E(u,v) = \sum_{x,y} \underbrace{w(x,y)}_{\text{window function}} \, [\underbrace{I(x+u,y+v)}_{\text{shifted intensity}} - \underbrace{I(x,y)}_{\text{intensity}}]^2$$

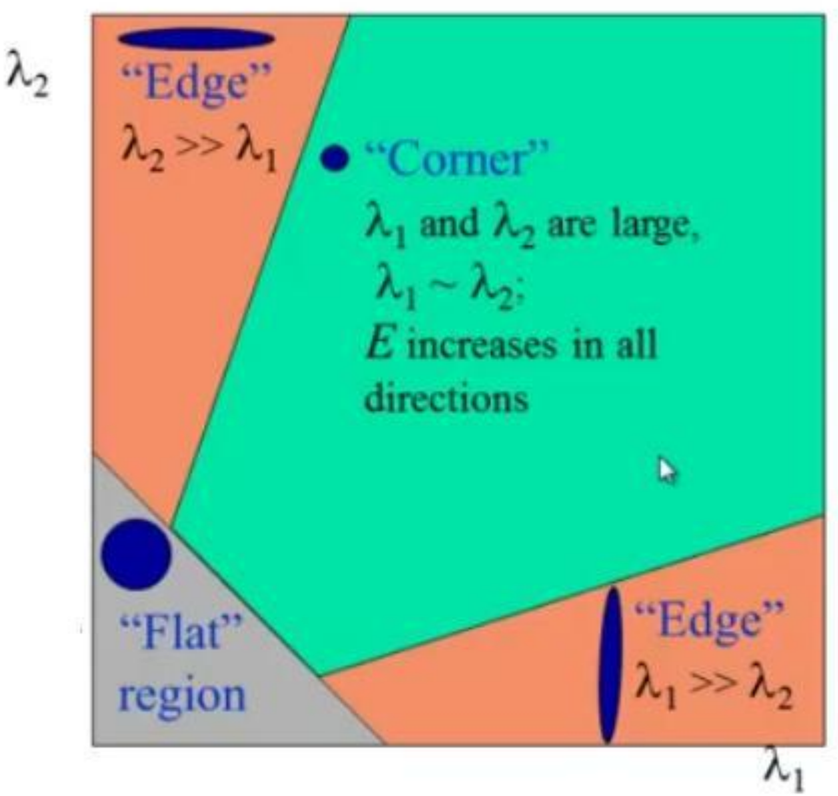
We have to maximize this function E(u,v) for corner detection. That means, we have to maximize the second term. Applying Taylor Expansion to above equation and using some mathematical steps, we get the final equation as:

$$E(u,v) \approx [u \quad v] \, M \begin{bmatrix} u \\ v \end{bmatrix} \qquad M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$$

Here, Ix and Iy are image derivatives in x and y directions respectively, which can be computed via Sobel.

Harris Corner Detection

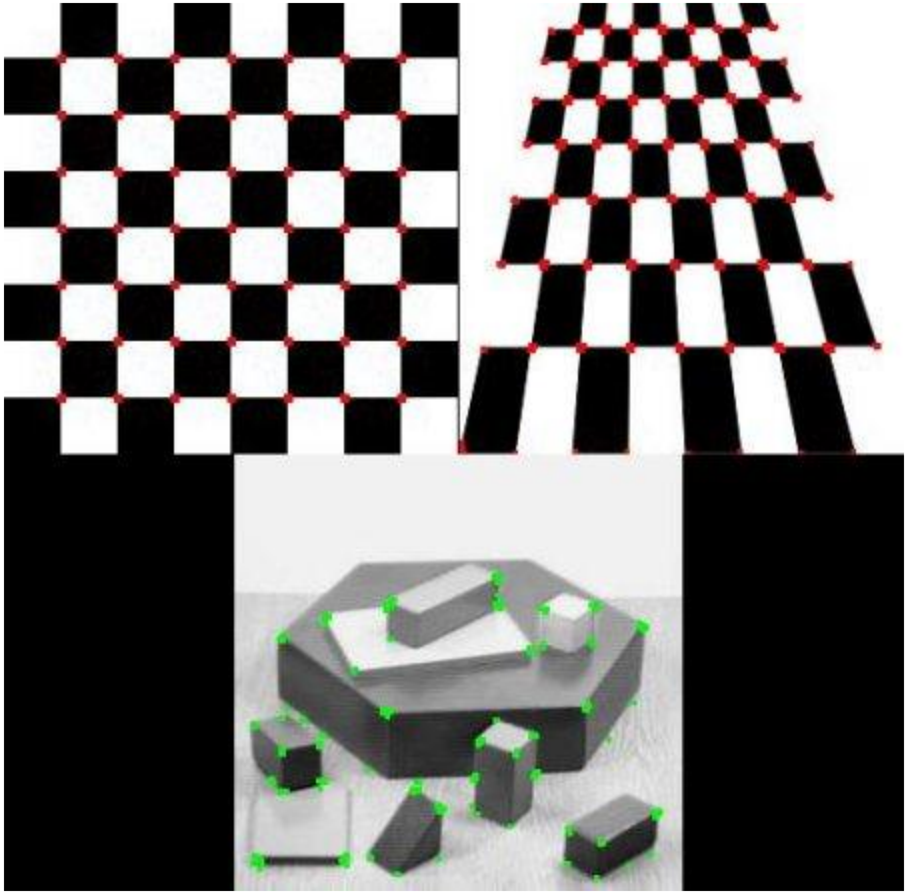
We can then look at the eigenvalues λ1 and λ2, which decide whether a region is corner, edge or flat.



- When |R| is small, which happens when λ1 and λ2 are small, the region is flat.
- When R<0, which happens when λ1 >> λ2 or vice versa, the region is edge.
- When R is large, which happens when λ1 and λ2 are large and λ1 ~ λ2, the region is a corner.

Harris Corner Detection

Result:



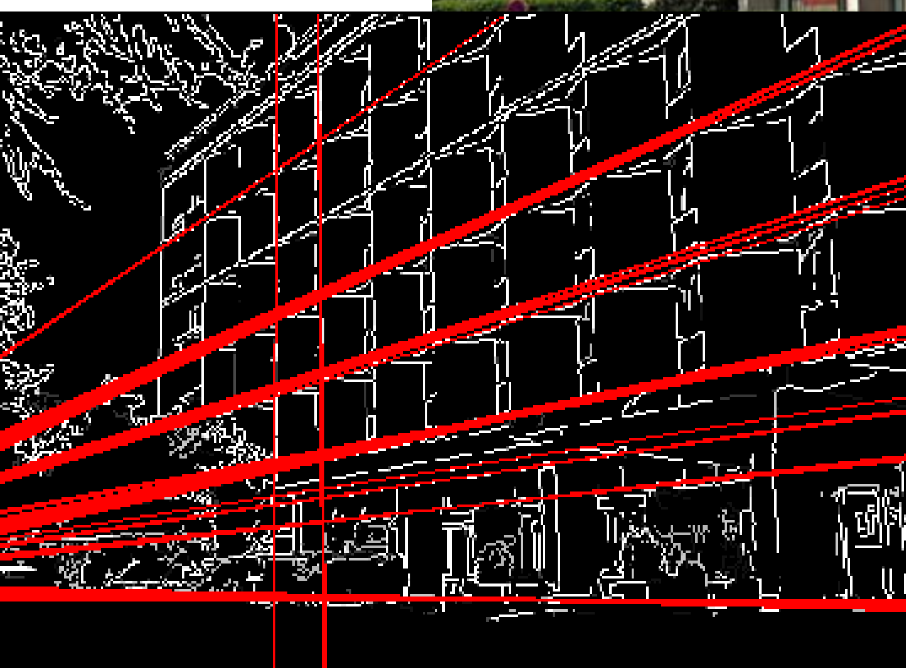
Hough Transform

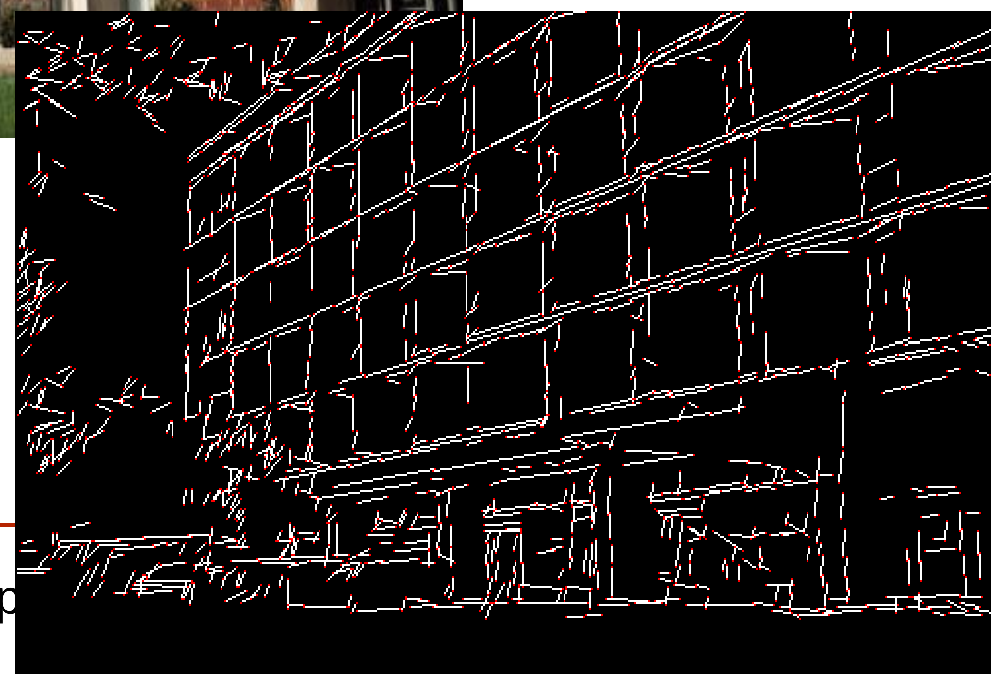
Any line can be represented in two terms, (ρ, θ), where ρ is the perpendicular distance from origin to the line, and θ is the angle formed by this perpendicular line and horizontal axis measured in counter-clockwise. So first it creates a 2D array or accumulator (to hold values of two parameters) and it is set to 0 initially. Let rows denote the ρ and columns denote the θ. Size of array depends on the accuracy you need. Suppose you want the accuracy of angles to be 1 degree, you need 180 columns. For ρ, the maximum distance possible is the diagonal length of the image. So taking one pixel accuracy, number of rows can be diagonal length of the image.

Hough Transform

Standard Hough Transform

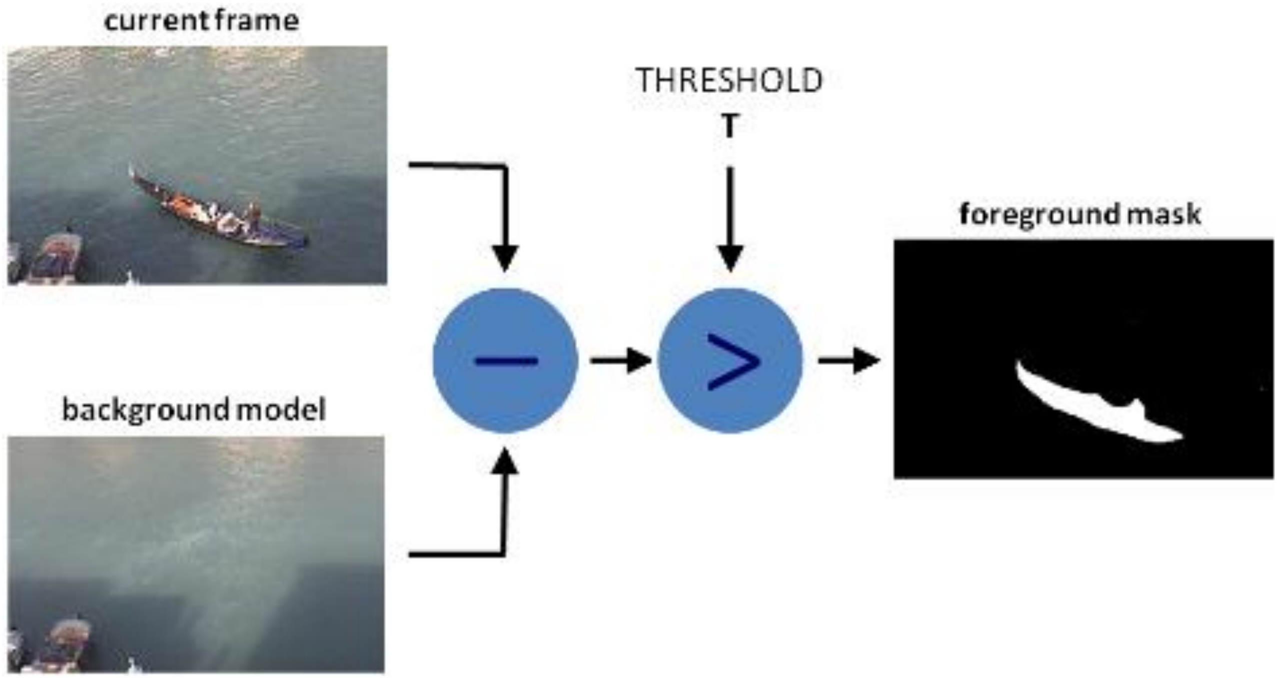
Probabilistic Hough Transform





Background Subtraction

Background subtraction is a common and widely used technique for generating a foreground mask (namely, a binary image containing the pixels belonging to moving objects in the scene) by using static cameras.



Background Subtraction

The threshold parameter is important as two images taken with even the same camera will likely not be identical. Thus, a threshold parameter allows for some variance. Often times, blurring the images, e.g. with a Gaussian filter, is used to make this approach work with similar but not identical images.

OpenCV provides different approaches for such a background subtraction algorithm.

Background Subtraction

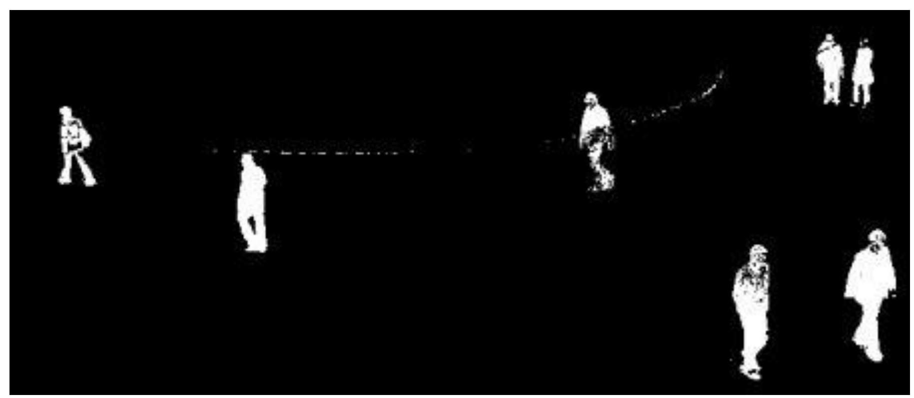
BackgroundSubtractorMOG

It is a Gaussian Mixture-based Background/Foreground Segmentation Algorithm. It was introduced in the paper "An improved adaptive background mixture model for real-time tracking with shadow detection" by P. KadewTraKuPong and R. Bowden in 2001. It uses a method to model each background pixel by a mixture of K Gaussian distributions (K = 3 to 5). The weights of the mixture represent the time proportions that those colors stay in the scene. The probable background colors are the ones which stay longer and more static.

Background Subtraction

BackgroundSubtractorMOG - Result

```
import numpy as np
import cv2
cap = cv2.VideoCapture('vtest.avi')
fgbg = cv2.createBackgroundSubtractorMOG()
while(1):
    ret, frame = cap.read()
    fgmask = fgbg.apply(frame)
    cv2.imshow('frame',fgmask)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
         break
cap.release()
cv2.destroyAllWindows()
```



Background Subtraction

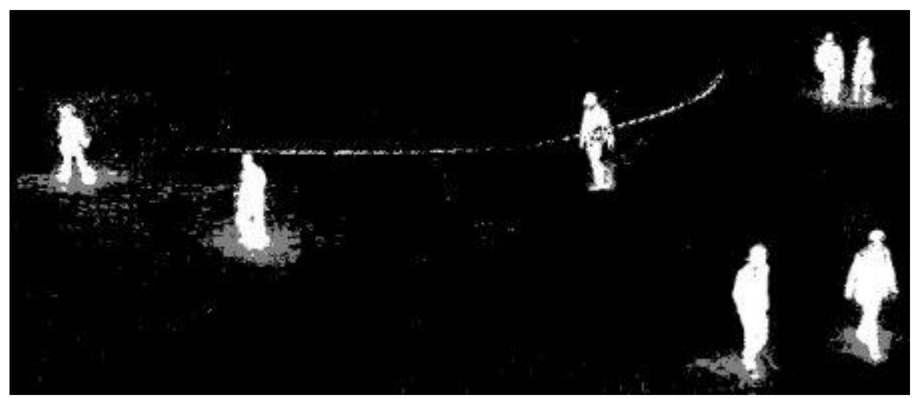
BackgroundSubtractorMOG2

It is also a Gaussian Mixture-based Background/Foreground Segmentation Algorithm. It is based on two papers by Z.Zivkovic, "Improved adaptive Gaussian mixture model for background subtraction" in 2004 and "Efficient Adaptive Density Estimation per Image Pixel for the Task of Background Subtraction" in 2006. One important feature of this algorithm is that it selects the appropriate number of Gaussian distribution for each pixel. (Remember, in last case, we took a K Gaussian distributions throughout the algorithm). It provides better adaptability to varying scenes due illumination changes etc.

Background Subtraction

BackgroundSubtractorMOG2 - Results

```
import numpy as np
import cv2
cap = cv2.VideoCapture('vtest.avi')
fgbg = cv2.createBackgroundSubtractorMOG2()
while(1):
    ret, frame = cap.read()
    fgmask = fgbg.apply(frame)
    cv2.imshow('frame',fgmask)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break
cap.release()
cv2.destroyAllWindows()
```



Background Subtraction

BackgroundSubtractorGMG

This algorithm combines statistical background image estimation and per-pixel Bayesian segmentation. It was introduced by Andrew B. Godbehere, Akihiro Matsukawa, Ken Goldberg in their paper "Visual Tracking of Human Visitors under Variable-Lighting Conditions for a Responsive Audio Art Installation" in 2012. As per the paper, the system ran a successful interactive audio art installation called “Are We There Yet?” from March 31 - July 31 2011 at the Contemporary Jewish Museum in San Francisco, California.

Background Subtraction

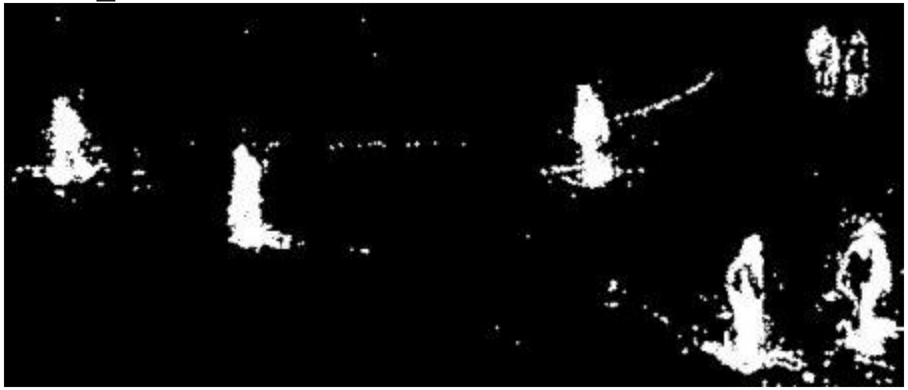
BackgroundSubtractorGMG (continued)

It uses first few (120 by default) frames for background modelling. It employs a probabilistic foreground segmentation algorithm that identifies possible foreground objects using Bayesian inference. The estimates are adaptive; newer observations are more heavily weighted than old observations to accommodate variable illumination. Several morphological filtering operations like closing and opening are done to remove unwanted noise. You will get a black window during first few frames.

Background Subtraction

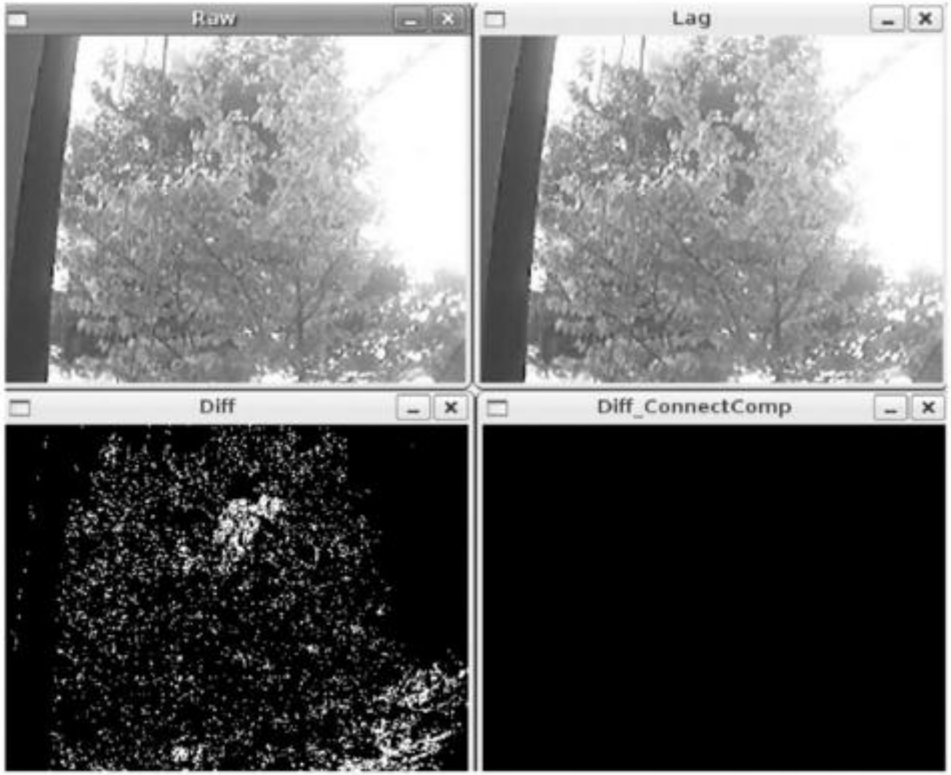
BackgroundSubtractorGMG - Results

```
import numpy as np
import cv2
cap = cv2.VideoCapture('vtest.avi')
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(3,3))
fgbg = cv2.createBackgroundSubtractorGMG()
while(1):
ret, frame = cap.read()
fgmask = fgbg.apply(frame)
fgmask = cv2.morphologyEx(fgmask, cv2.MORPH_OPEN, kernel)
cv2.imshow('frame',fgmask)
k = cv2.waitKey(30) & 0xff
if k == 27:
break
cap.release()
cv2.destroyAllWindows()
```



Background Subtraction

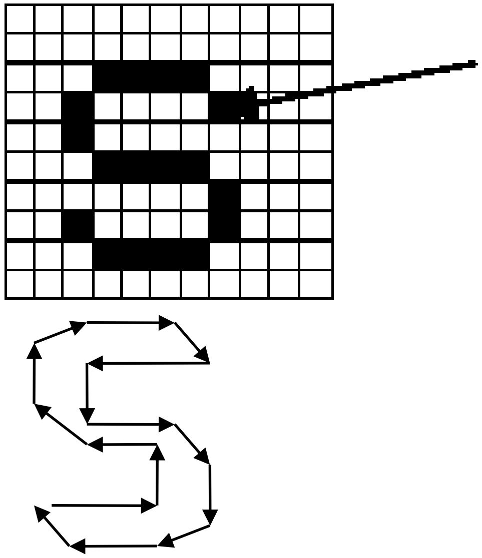
Further clean-up of the image may be necessary. For example, a tree waiving in the wind will likely leave residue in the image after background connection. This type of noise can be cleaned up by despeckle filters or the connected-components algorithm.



Contour Retrieving

The contour representation:

- ✓ Chain code (Freeman code)
- ✓ Polygonal representation

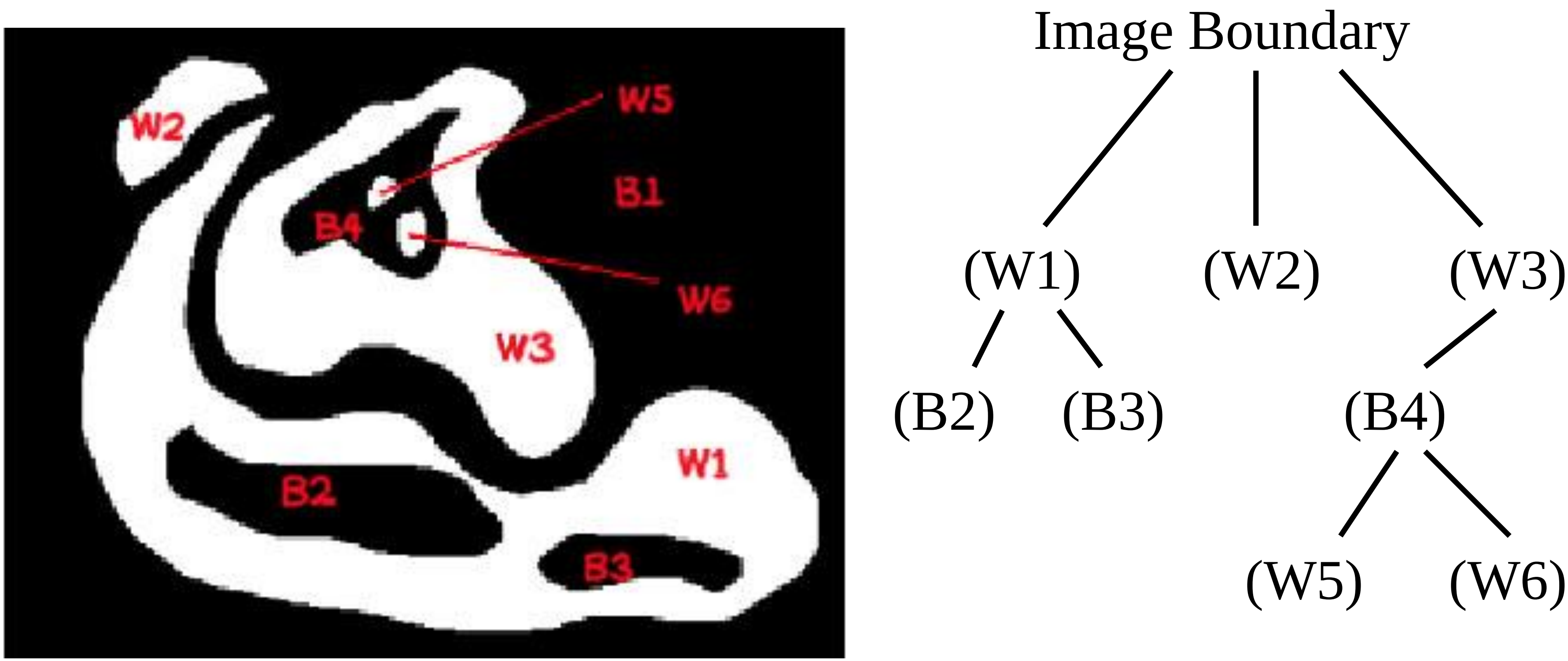


Initial Point

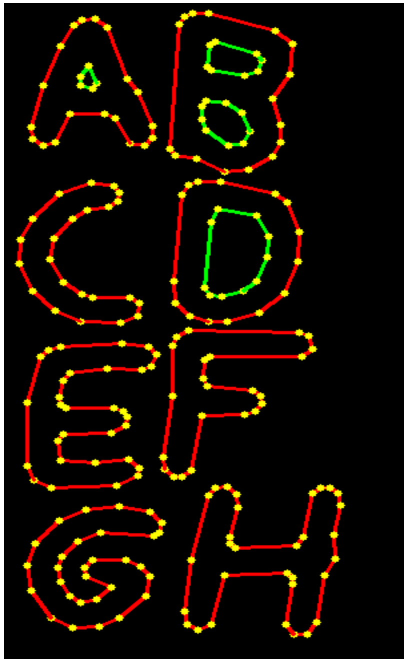
Chain code for the curve:
3444567000765443

Contour representation

Hierarchical representation of contours



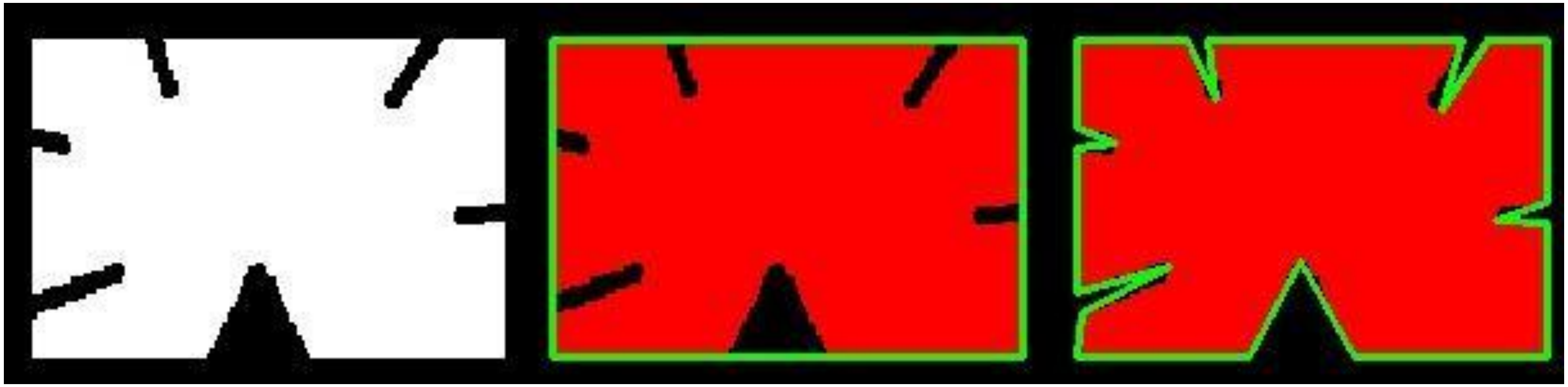
Contours Examples



Contour algorithms

OpenCV implements different types of contour algorithms. A polynomial contour can be retrieved like this:

```
epsilon = 0.1*cv2.arcLength(cnt,True)
approx = cv2.approxPolyDP(cnt,epsilon,True)
epsilon = 0.01*cv2.arcLength(cnt,True)
approx = cv2.approxPolyDP(cnt,epsilon,True)
```



0.1% 0.01%

Contour algorithms

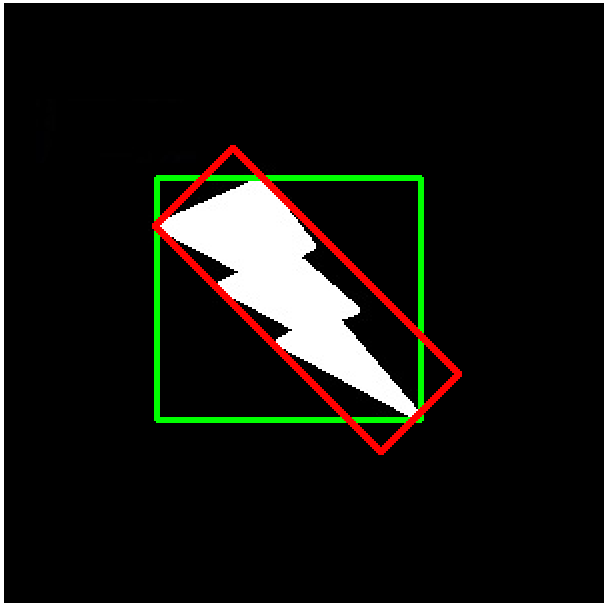
Convex Hull:

```
hull = cv2.convexHull(points, hull,
                      clockwise, returnPoints))
```

Contour algorithms

Bounding Rectangle (straight or rotated):

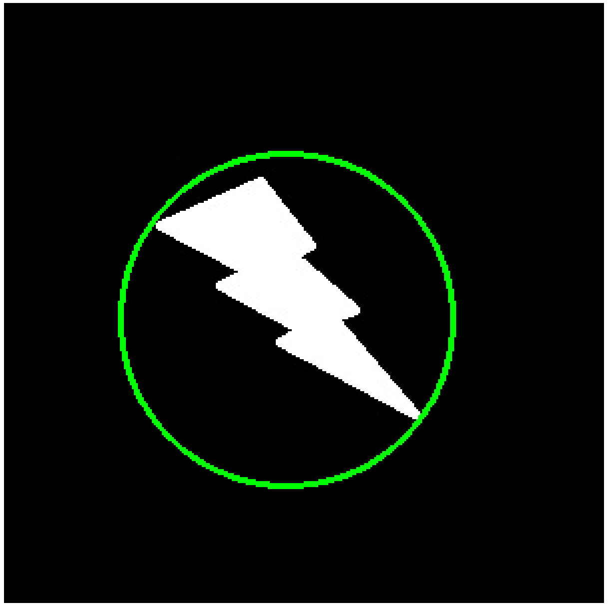
```
x,y,w,h = cv2.boundingRect(cnt)
cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)
rect = cv2.minAreaRect(cnt)
box = cv2.boxPoints(rect)
box = np.int0(box)
cv2.drawContours(img,[box],
                          0,(0,0,255),
                          2)
```



Contour algorithms

Minimum Enclosing Circle:

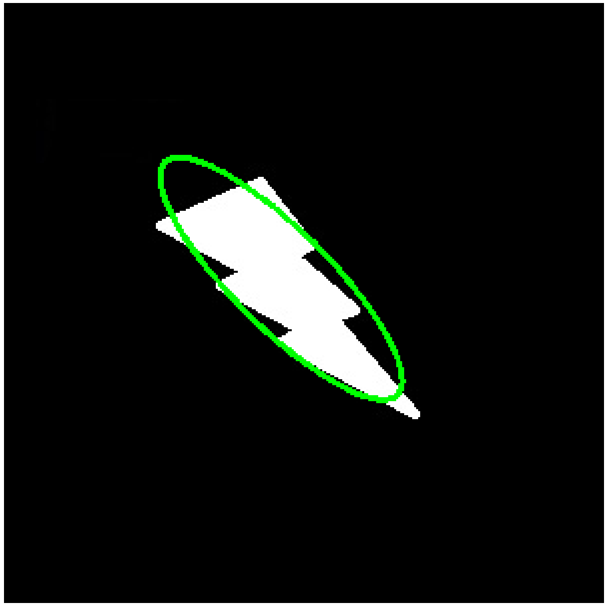
```
(x,y),radius = cv2.minEnclosingCircle(cnt)
center = (int(x),int(y))
radius = int(radius)
cv2.circle(img,center,
           radius,(0,255,0),2)
```



Contour algorithms

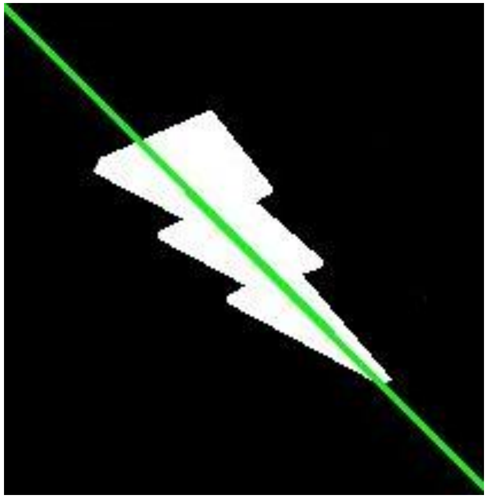
Fitting an Ellipse:

```
ellipse = cv2.fitEllipse(cnt)
cv2.ellipse(img,ellipse,(0,255,0),2)
```



Contour algorithms

Fitting a Line:



```
rows,cols = img.shape[:2]
[vx,vy,x,y] = cv2.fitLine(cnt,
  cv2.DIST_L2,0,0.01,0.01)
lefty = int((-x*vy/vx) + y)
righty = int(((cols-x)*vy/vx)+y)
cv2.line(img,(cols-
  1,righty),(0,lefty),(0,255,0),2)
```

OpenCV Functionality

- ✓ Basic structures and operations
- ✓ Image Analysis
- Structural Analysis

Object Recognition

Motion Analysis and Object Tracking

3D Reconstruction

Structural Analysis

- Contours processing
  - Approximation
  - Hierarchical representation
  - Shape characteristics
  - Matching
- Geometry
  - Contour properties
  - Fitting with primitives
  - PGH: pair-wise geometrical histogram for the contour.

Contour Processing

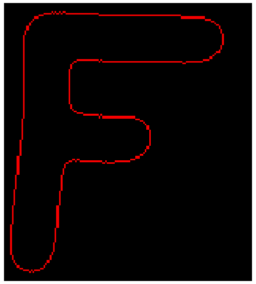
Approximation:

- ✓ RLE algorithm (chain code)
- ✓ Teh-Chin approximation (polygonal)
- ✓ Douglas-Peucker approximation (polygonal);

Contour moments (central and normalized up to order 3)

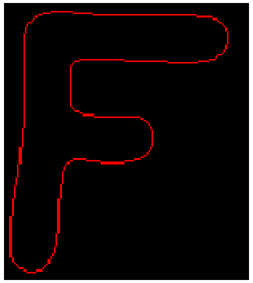
Hierarchical representation of contours

Matching of contours



Source Image

Rosenfeld-Johnston Algorithm Output



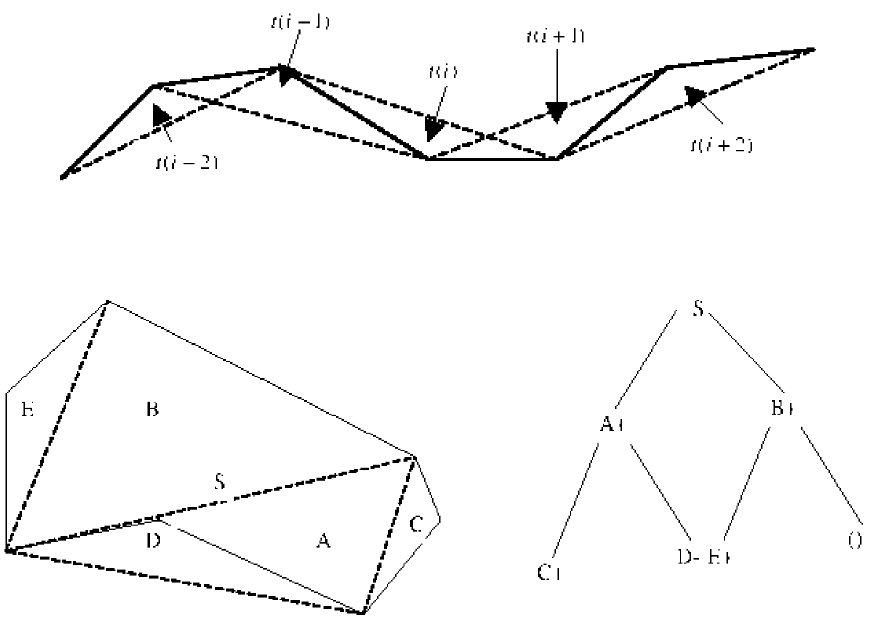
Teh-Chin Algorithm Output

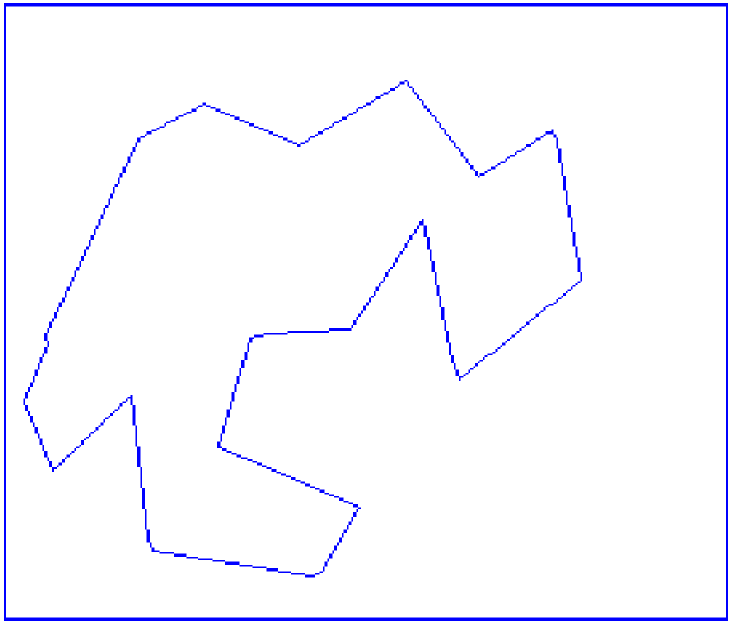
Hierarchical Representation of Contours

A contour is represented with a binary tree

Given the binary tree, the contour can be retrieved with arbitrary precision

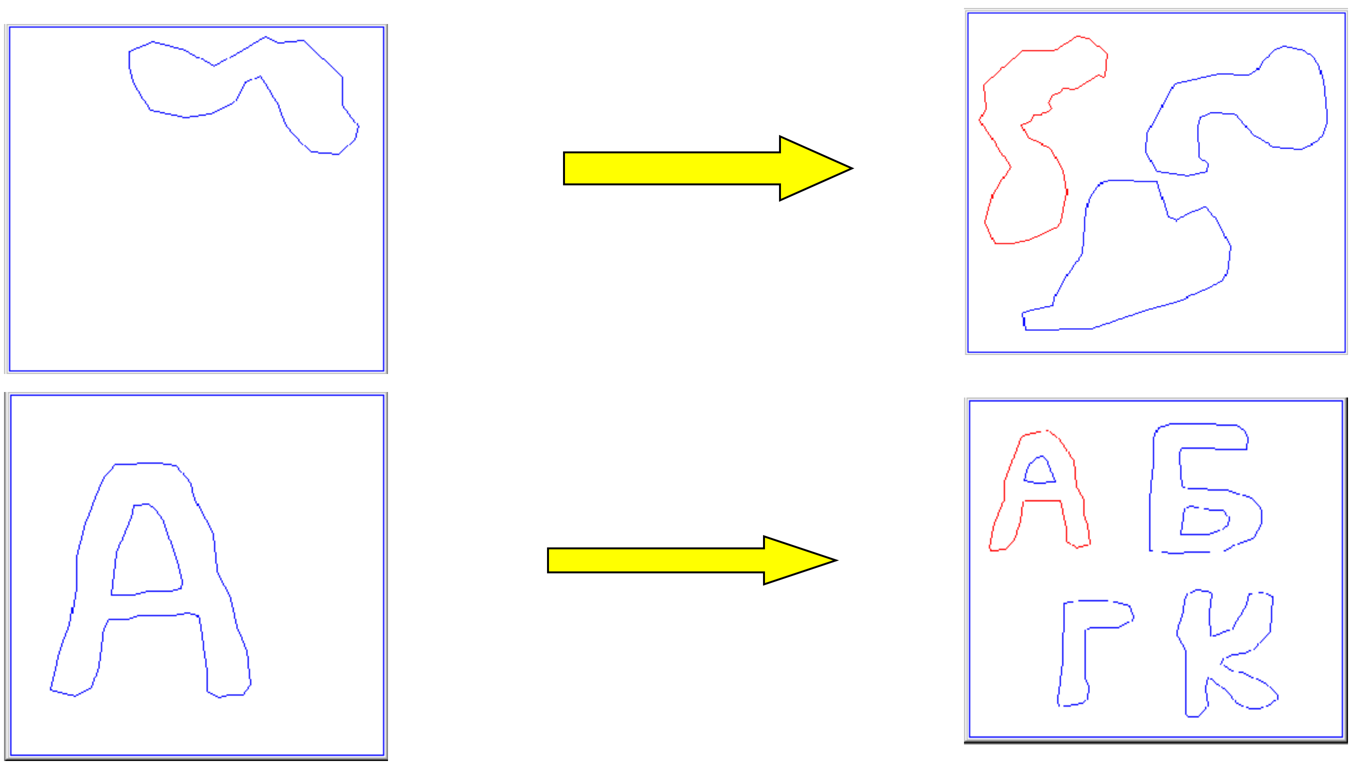
The binary tree is quasi invariant to translations, rotations and scaling





Contours matching

Matching based on hierarchical representation of contours

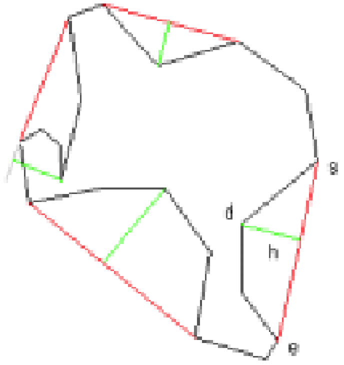


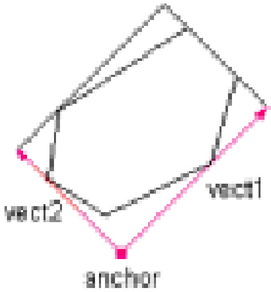
Geometry

Properties of contours: (perimeter, area, convex hull, convexity defects, rectangle of minimum area)

Fitting: (2D line, 3D line, circle, ellipse)

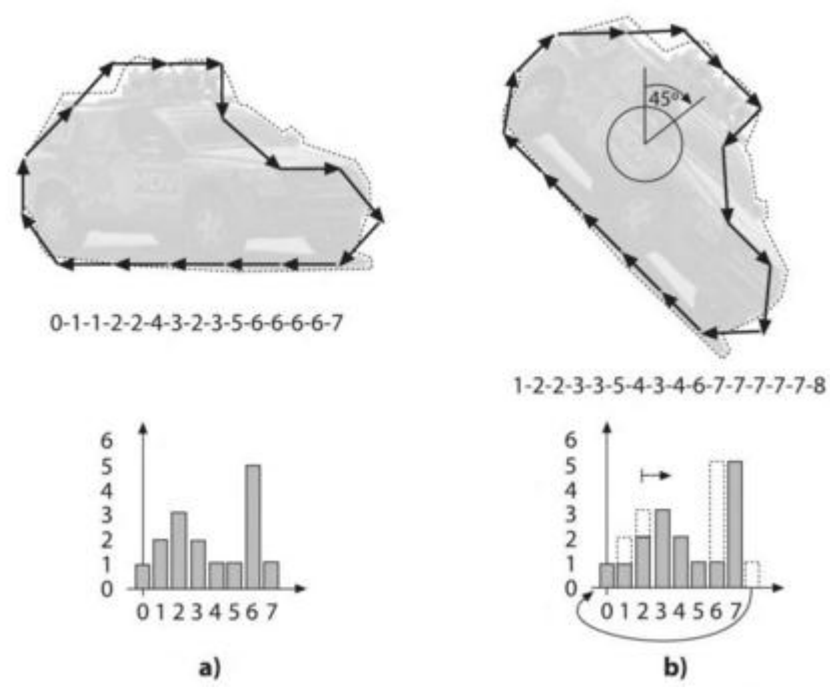
Pair-wise geometrical histogram





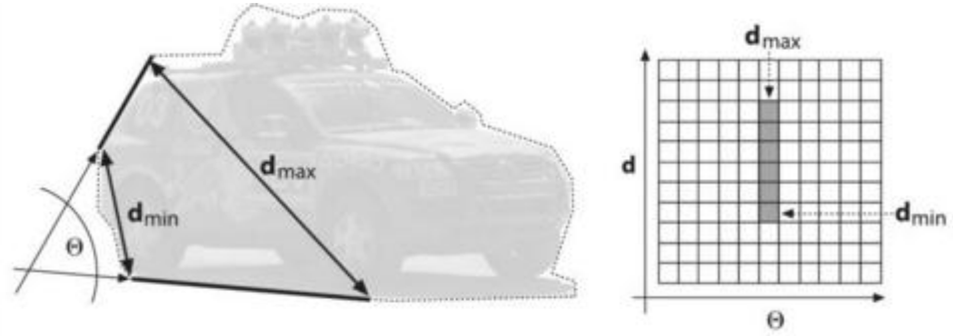
Pair-wise geometrical histogram (PGH)

PGH can measure similarity between objects. It is a generalization of the chain code histogram (CCH):
Count the number of each kind of steps in the Freeman chain code representation of the contour

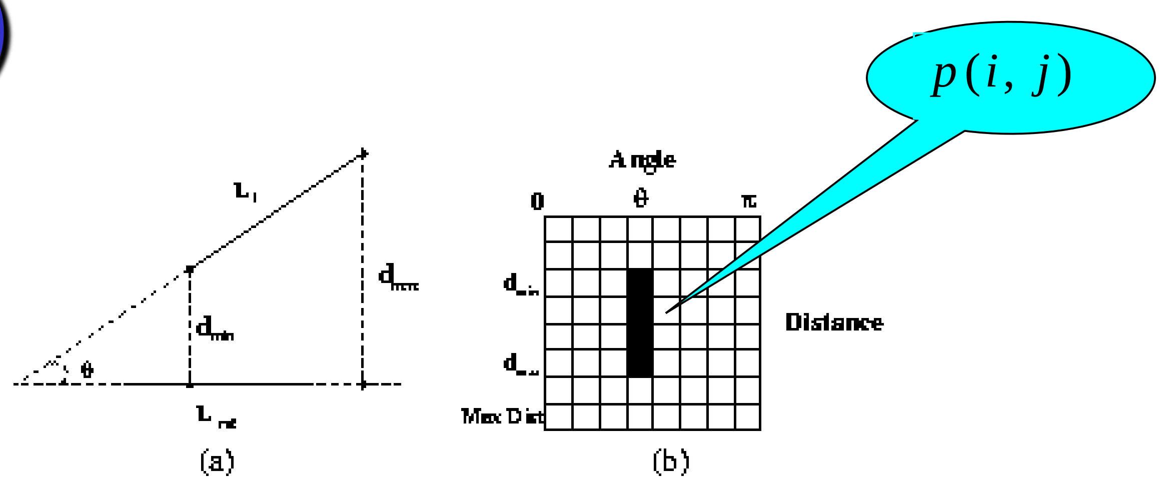


Pair-wise geometrical histogram (PGH)

The PGH is constructed as follows: Each of the edges of the polygon is successively chosen to be the "base edge". Then each oif the other edges is considered relative to that base edge and three values are computed: d_{min}, d_{max}, and θ. D_{min} is the smallest distance between the two edges, d_{max} is the largest, and θ is the angle between them. The PGH is the 2D histogram whose dimensions are the angle and the distance.



Pair-wise geometrical histogram (PGH)



$$f_{PGH} = [E_r(1), E_r(2), \ldots E_r(N), E_c(1), E_c(2), \ldots E_c(M)]^T,$$

$$E_r(i) = \sum_j j \cdot p(i, j) / \sum_j p(i, j),$$

$$E_c(j) = \sum_i i \cdot p(i, j) / \sum_i p(i, j).$$

OpenCV Functionality

- ✓ Basic structures and operations
- ✓ Image Analysis
- ✓ Structural Analysis
- Object Recognition

Motion Analysis and Object Tracking

3D Reconstruction

Object Recognition

Eigen objects

Hidden Markov Models

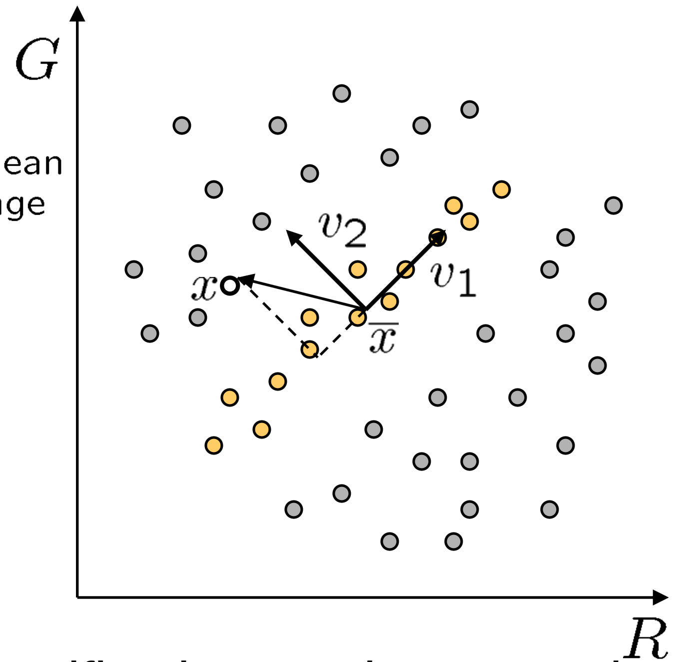
Eigenfaces for recognition

Matthew Turk and Alex Pentland

J. Cognitive Neuroscience

1991

Linear subspaces



$\overline{x}$ is the mean of the orange points

convert **x** into $\mathbf{v_1}$, $\mathbf{v_2}$ coordinates

$$\mathbf{x} \rightarrow ((\mathbf{x} - \overline{x}) \cdot \mathbf{v_1}, (\mathbf{x} - \overline{x}) \cdot \mathbf{v_2})$$

What does the $\mathbf{v_2}$ coordinate measure?

- distance to line
- use it for classification—near 0 for orange pts

What does the $\mathbf{v_1}$ coordinate measure?

- position along line
- use it to specify which orange point it is

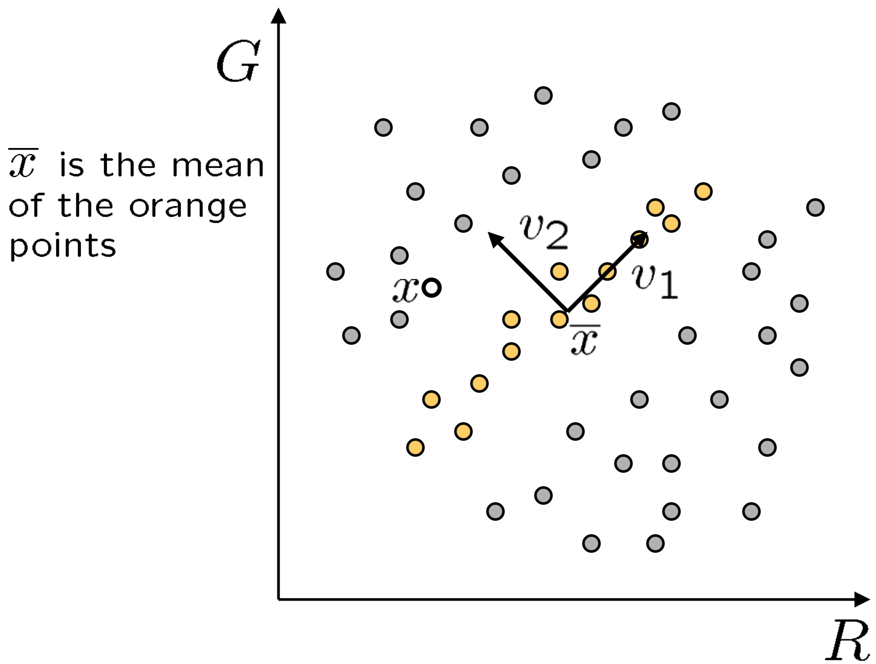
Classification can be expensive:

Big search prob (e.g., nearest neighbors) or store large PDF's

Suppose the data points are arranged as above

Idea—fit a line, classifier measures distance to line

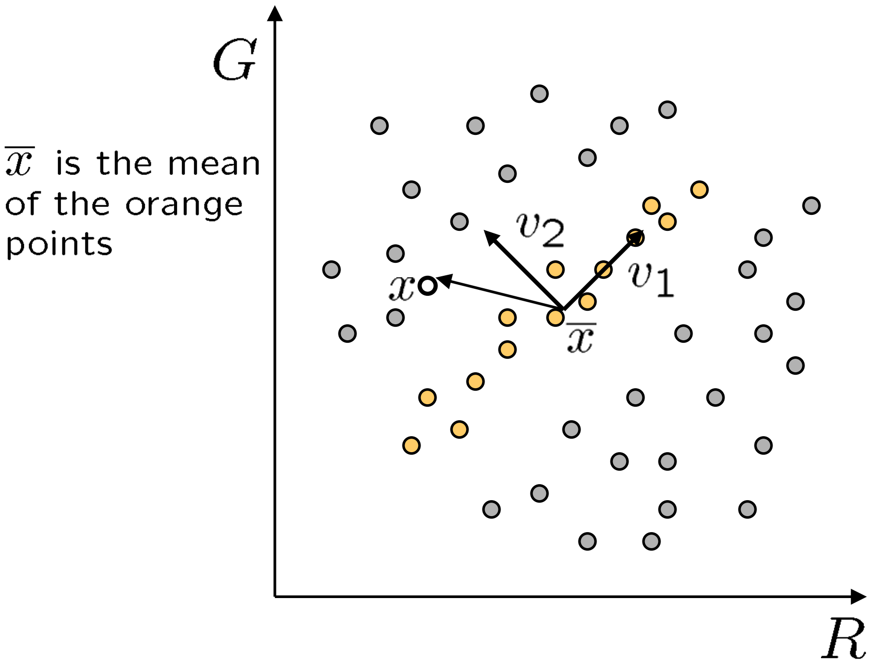
Dimensionality reduction



Dimensionality reduction

- We can represent the orange points with *only* their $\mathbf{v_1}$ coordinates (since $\mathbf{v_2}$ coordinates are all essentially 0)
- This makes it much cheaper to store and compare points
- A bigger deal for higher dimensional problems

Linear subspaces



Consider the variation along direction **v** among all of the orange points:

$$var(\mathbf{v}) = \sum_{\text{orange point } \mathbf{x}} \|(\mathbf{x} - \overline{\mathbf{x}})^{\mathbf{T}} \cdot \mathbf{v}\|^{2}$$

What unit vector **v** minimizes *var*?

$$\mathbf{v}_2 = min_{\mathbf{v}} \; \{var(\mathbf{v})\}$$

What unit vector **v** maximizes *var*?

$$\mathbf{v}_1 = max_{\mathbf{v}} \; \{var(\mathbf{v})\}$$

$$\begin{aligned} var(\mathbf{v}) &= \sum_{\mathbf{x}} \|(\mathbf{x} - \overline{\mathbf{x}})^{\mathbf{T}} \cdot \mathbf{v}\| \\ &= \sum_{\mathbf{x}} \mathbf{v}^{\mathbf{T}} (\mathbf{x} - \overline{\mathbf{x}})(\mathbf{x} - \overline{\mathbf{x}})^{\mathbf{T}} \mathbf{v} \\ &= \mathbf{v}^{\mathbf{T}} \left[\sum_{\mathbf{x}} (\mathbf{x} - \overline{\mathbf{x}})(\mathbf{x} - \overline{\mathbf{x}})^{\mathbf{T}} \right] \mathbf{v} \\ &= \mathbf{v}^{\mathbf{T}} \mathbf{A} \mathbf{v} \quad \text{where } \mathbf{A} = \sum_{\mathbf{x}} (\mathbf{x} - \overline{\mathbf{x}})(\mathbf{x} - \overline{\mathbf{x}})^{\mathbf{T}} \end{aligned}$$

Solution: $\mathbf{v}_1$ is eigenvector of **A** with *largest* eigenvalue
$\mathbf{v}_2$ is eigenvector of **A** with *smallest* eigenvalue

Principal component analysis

Suppose each data point is N-dimensional

Same procedure applies:

$$\begin{aligned} var(\mathbf{v}) &= \sum_{\mathbf{x}} \|(\mathbf{x} - \overline{\mathbf{x}})^{\mathbf{T}} \cdot \mathbf{v}\| \\ &= \mathbf{v}^{\mathbf{T}} \mathbf{A} \mathbf{v} \quad \text{where } \mathbf{A} = \sum_{\mathbf{x}} (\mathbf{x} - \overline{\mathbf{x}})(\mathbf{x} - \overline{\mathbf{x}})^{\mathbf{T}} \end{aligned}$$

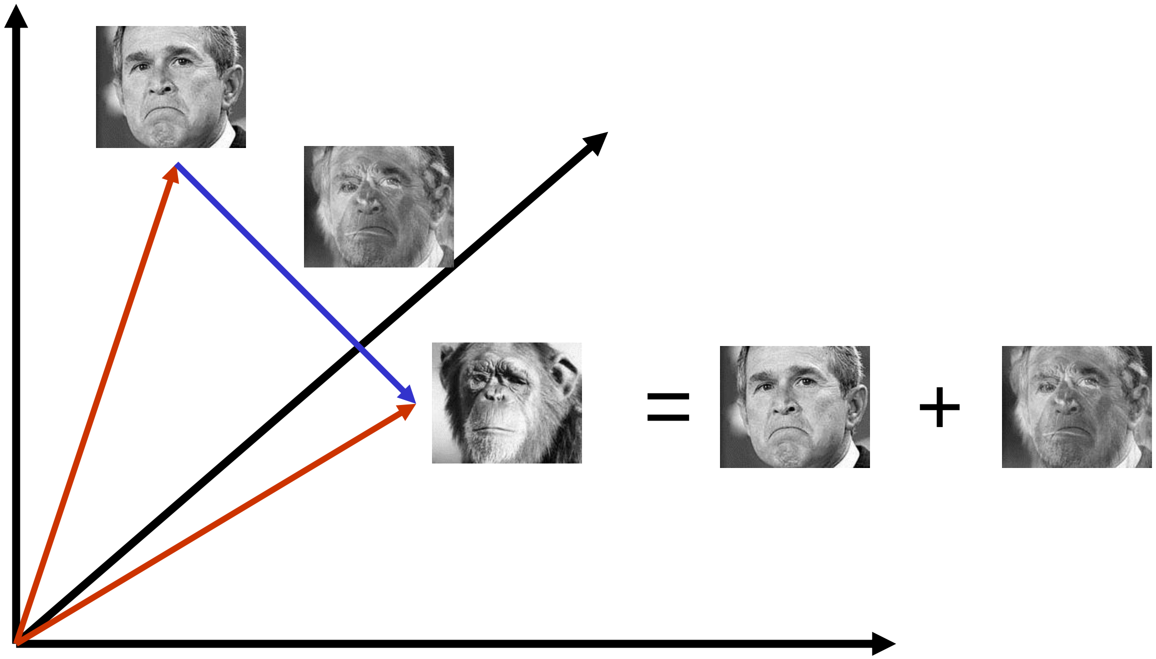
The eigenvectors of **A** define a new coordinate system

- eigenvector with largest eigenvalue captures the most variation among training vectors **x**
- eigenvector with smallest eigenvalue has least variation

We can compress the data using the top few eigenvectors

- corresponds to choosing a “linear subspace”
  - represent points on a line, plane, or “hyper-plane”
- these eigenvectors are known as the ***principal components***

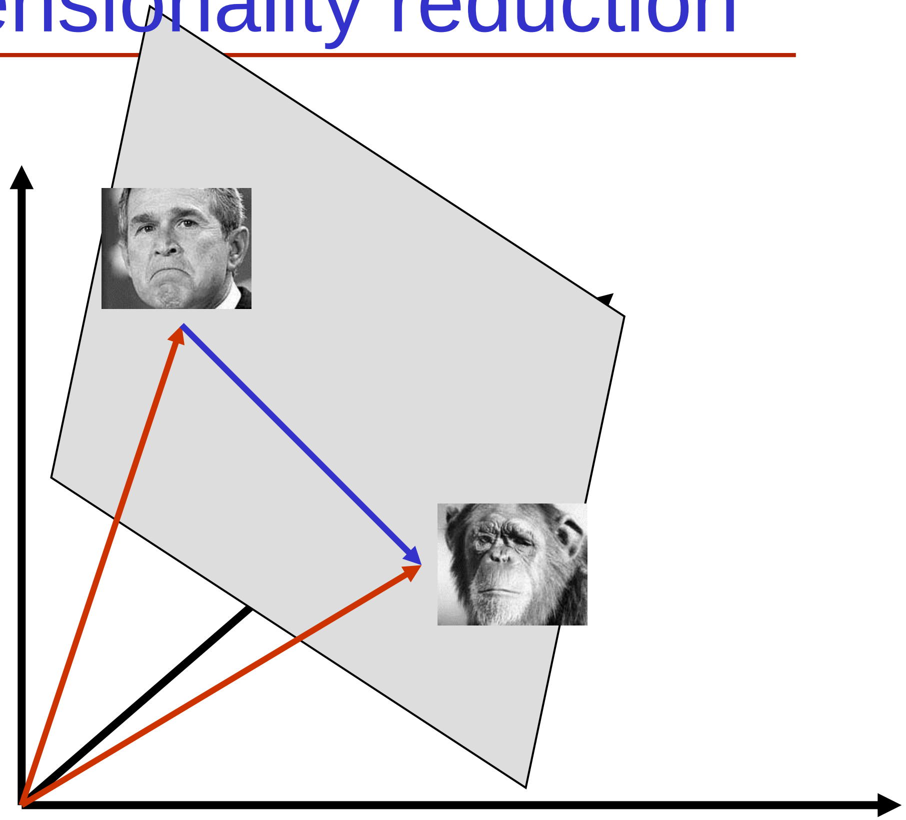
The space of faces



An image is a point in a high dimensional space

- An N x M image is a point in R^{NM}
- We can define vectors in this space as we did in the 2D case

Dimensionality reduction



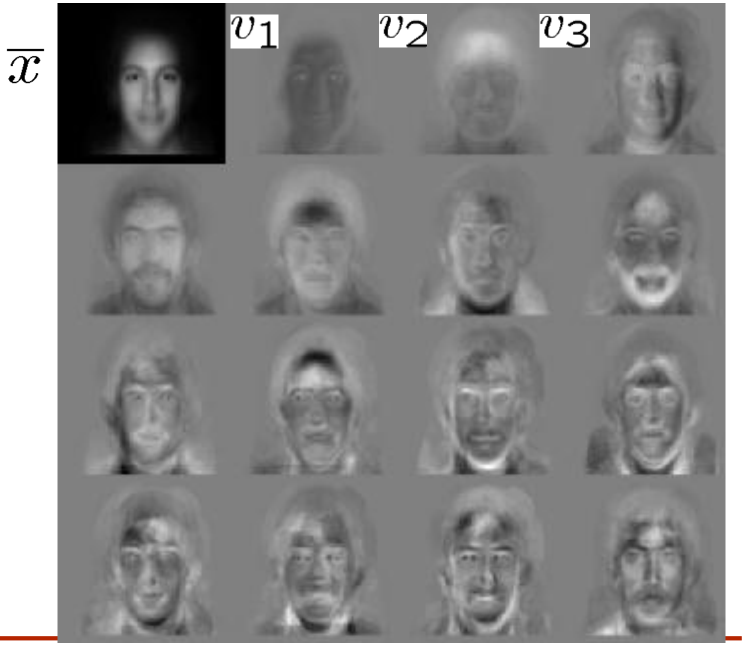
The set of faces is a "subspace" of the set of images

- We can find the best subspace using PCA
- This is like fitting a "hyper-plane" to the set of faces
  - spanned by vectors $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_K}$
  - any face $\mathbf{x} \approx \overline{\mathbf{x}} + a_1\mathbf{v_1} + a_2\mathbf{v_2} + \ldots + a_k\mathbf{v_k}$

Eigenfaces

PCA extracts the eigenvectors of **A**

- Gives a set of vectors $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$, ...
- Each vector is a direction in face space
  - what do these look like?



Projecting onto the eigenfaces

The eigenfaces $\mathbf{v}_1, \ldots, \mathbf{v}_K$ span the space of faces

A face is converted to eigenface coordinates by

$$\mathbf{x} \rightarrow (\underbrace{(\mathbf{x} - \overline{\mathbf{x}}) \cdot \mathbf{v}_1}_{a_1}, \underbrace{(\mathbf{x} - \overline{\mathbf{x}}) \cdot \mathbf{v}_2}_{a_2}, \ldots, \underbrace{(\mathbf{x} - \overline{\mathbf{x}}) \cdot \mathbf{v}_K}_{a_K})$$

$$\mathbf{x} \approx \overline{\mathbf{x}} + a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \ldots + a_K\mathbf{v}_K$$

$\mathbf{x}$

$a_1\mathbf{v}_1$ $a_2\mathbf{v}_2$ $a_3\mathbf{v}_3$ $a_4\mathbf{v}_4$ $a_5\mathbf{v}_5$ $a_6\mathbf{v}_6$ $a_7\mathbf{v}_7$ $a_8\mathbf{v}_8$

Recognition with eigenfaces

Algorithm

1. Process the image database (set of images with labels)
   - Run PCA—compute eigenfaces
   - Calculate the K coefficients for each image
2. Given a new image (to be recognized) **x**, calculate K coefficients

$$\mathbf{x} \rightarrow (a_1, a_2, \ldots, a_K)$$

3. Detect if x is a face

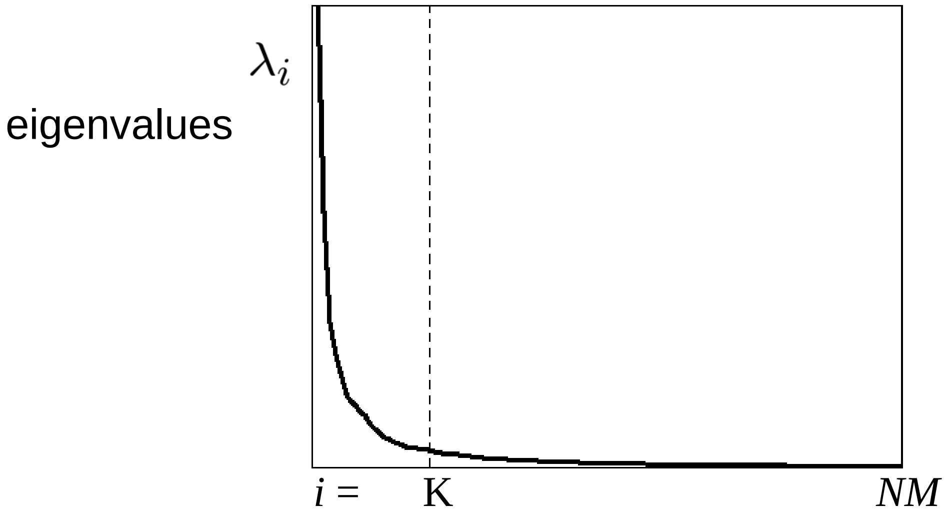
$$\|\mathbf{x} - (\overline{\mathbf{x}} + a_1\mathbf{v_1} + a_2\mathbf{v_2} + \ldots + a_K\mathbf{v_K})\| < \text{threshold}$$

4. If it is a face, who is it?

   Find closest labeled face in database

   nearest-neighbor in K-dimensional space

Choosing the dimension K



How many eigenfaces to use?

Look at the decay of the eigenvalues

- the eigenvalue tells you the amount of variance "in the direction" of that eigenface
- ignore eigenfaces with low variance

Eigen objects (continued)

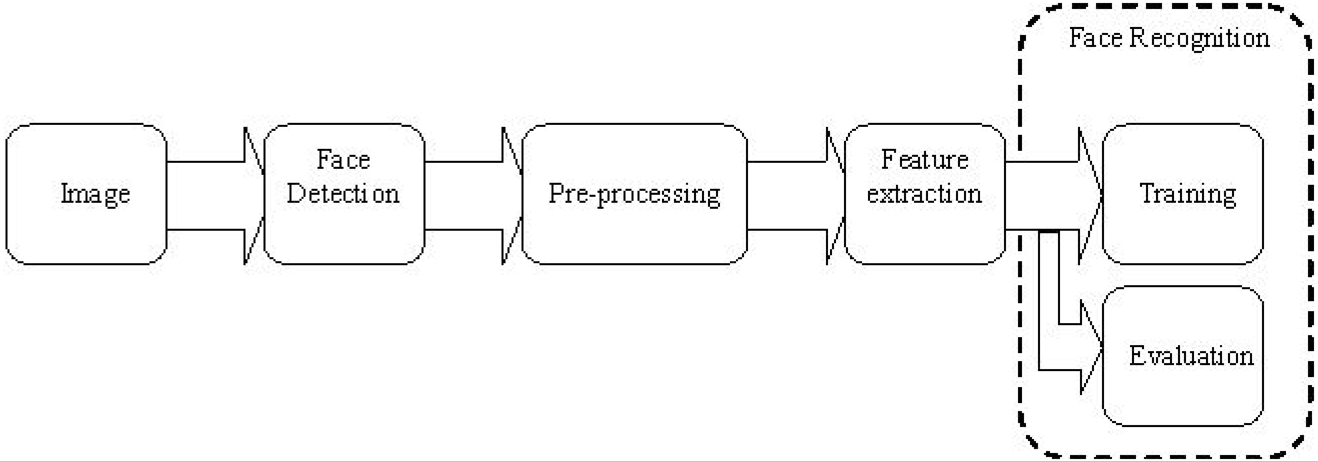
Hidden Markov Model

Hidden Markov Models (HMMs) are a class of statistical models used to characterize the observable properties of a signal. HMMs consist of two interrelated processes:

- an underlying, unobservable Markov chain with a finite number of states governed by a state transition probability matrix and an initial state probability distribution, and
- a set of observations, defined by the observation density functions associated with each state.

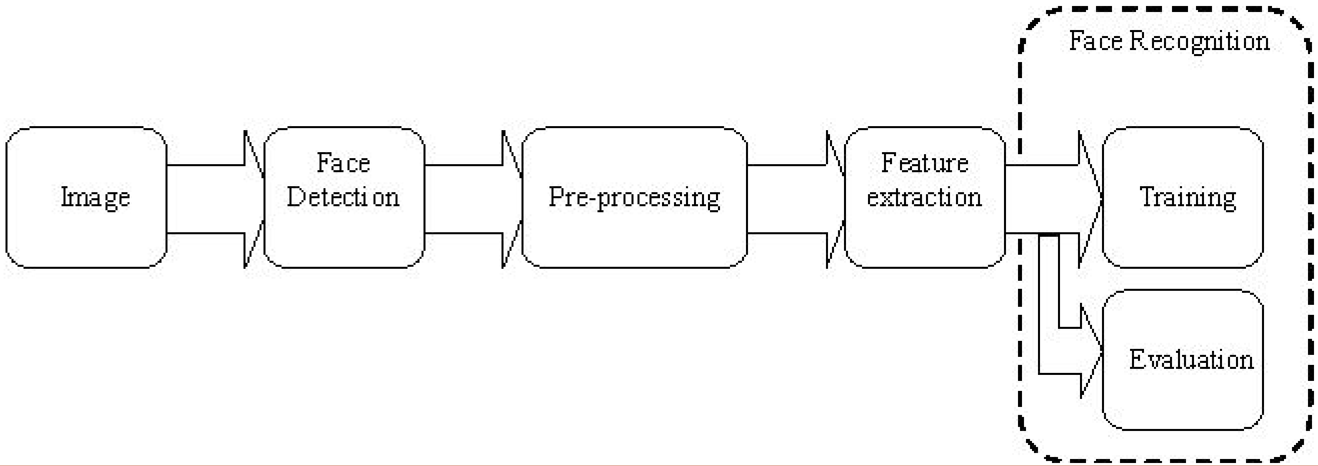
Hidden Markov Model

Face detection and cropping block: this is the first stage of any face recognition system and the key difference between a semi-automatic and a fully automatic face recognizer. In order to make the recognition system fully automatic, the detection and extraction of faces from an image should also be automatic. Face detection also represents a very important step before face recognition, because the accuracy of the recognition process is a direct function of the accuracy of the detection process



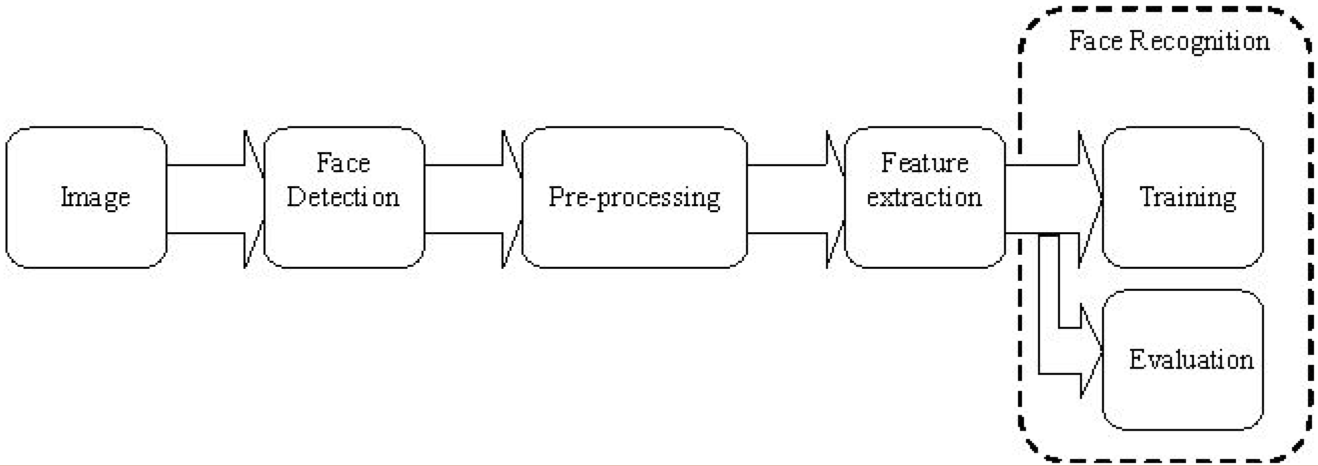
Hidden Markov Model

Pre-processing block: the face image can be treated with a series of pre-processing techniques to minimize the effect of factors that can adversely influence the face recognition algorithm. The most critical of these are *facial pose* and *illumination*



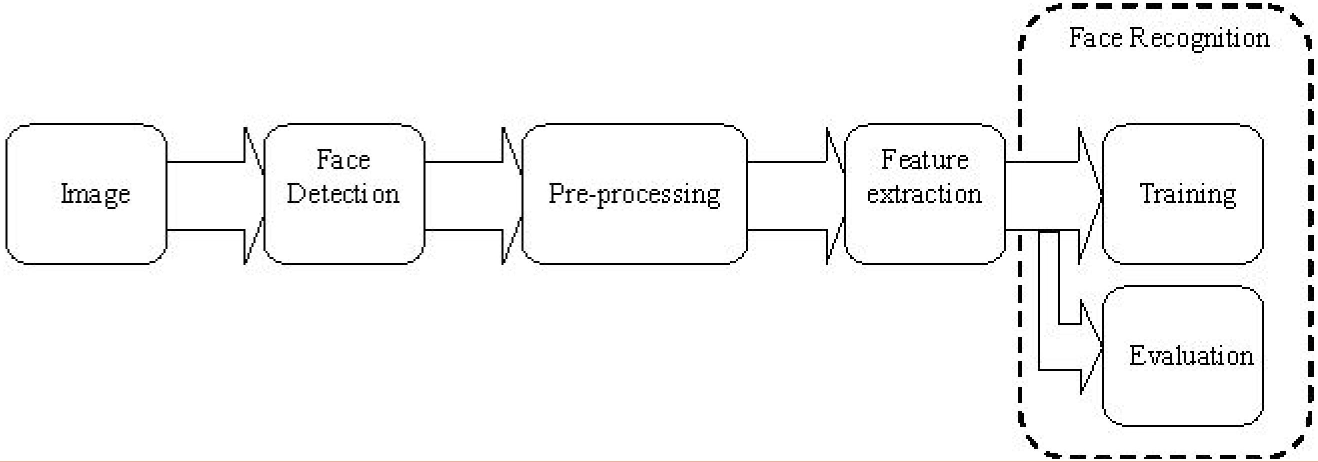
Hidden Markov Model

Feature extraction block: in this step the features used in the recognition phase are computed. These features vary depending on the automatic face recognition system used. For example, the first and most simplistic features used in face recognition were the geometrical relations and distances between important points in a face, and the recognition 'algorithm' matched these distances



Hidden Markov Model

Face recognition block: this consists of 2 separate stages: a *training process*, where the algorithm is fed samples of the subjects to be learned and a distinct model for each subject is determined; and an *evaluation process* where a model of a newly acquired test subject is compared against all existing models in the database and the most closely corresponding model is determined. If these are sufficiently close a recognition event is triggered.

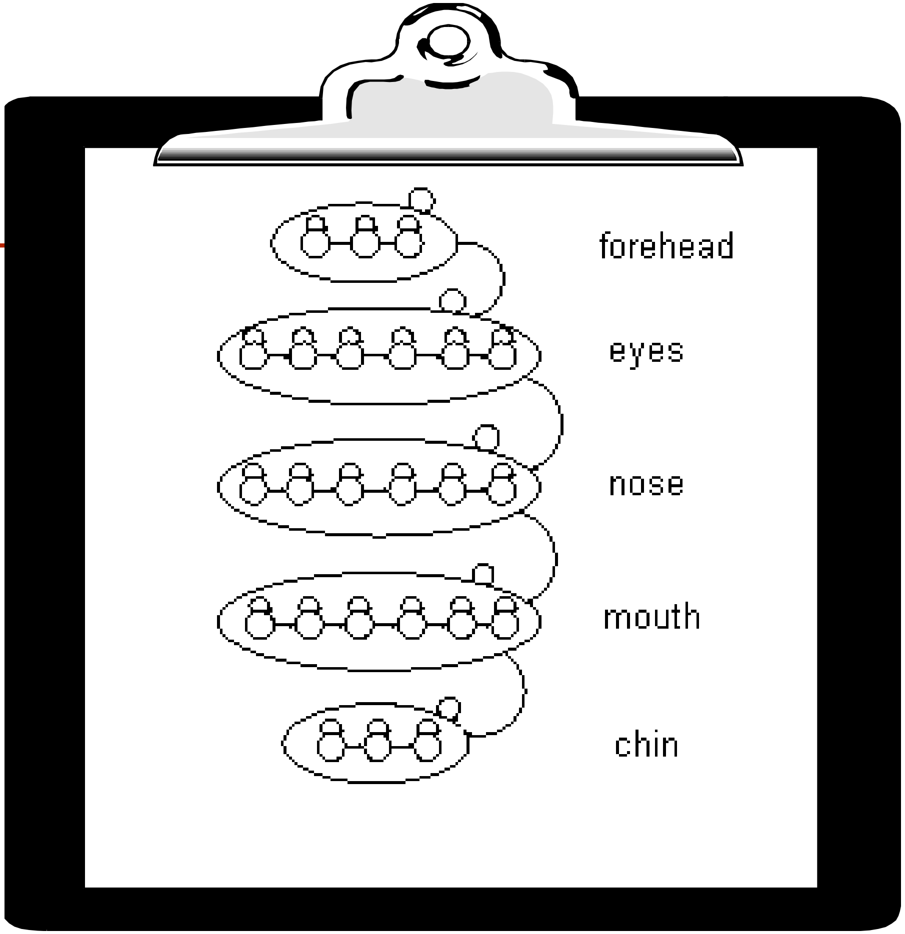


Hidden Markov Model

Based on the extracted features of a face (eyes, nose, mouth, …), the HMM can then be trained to recognize specific faces. For this, an enhanced version of the so-called Viterbi algorithm known as *double embedded Viterbi* was developed. It involves applying the Viterbi algorithm to both the embedded HMMs and to the global, or top-level HMM, hence the name.

Embedded HMM for Face Recognition

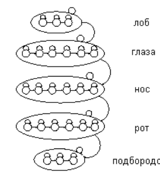
Model-

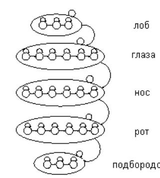


- Face ROI partition

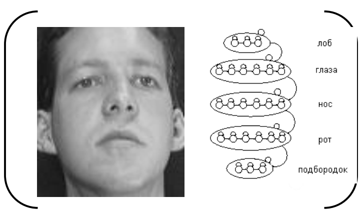
Face recognition
using Hidden Markov Models

- One person – one HMM
- Stage 1 – Train every HMM





- Stage 2 – Recognition



P_i - probability

Choose **max(P_i)**

OpenCV Functionality

- ✓ Basic structures and operations
- ✓ Image Analysis
- ✓ Structural Analysis
- ✓ Object Recognition
- Motion Analysis and Object Tracking

3D Reconstruction

Motion Analysis and Object Tracking

Motion templates

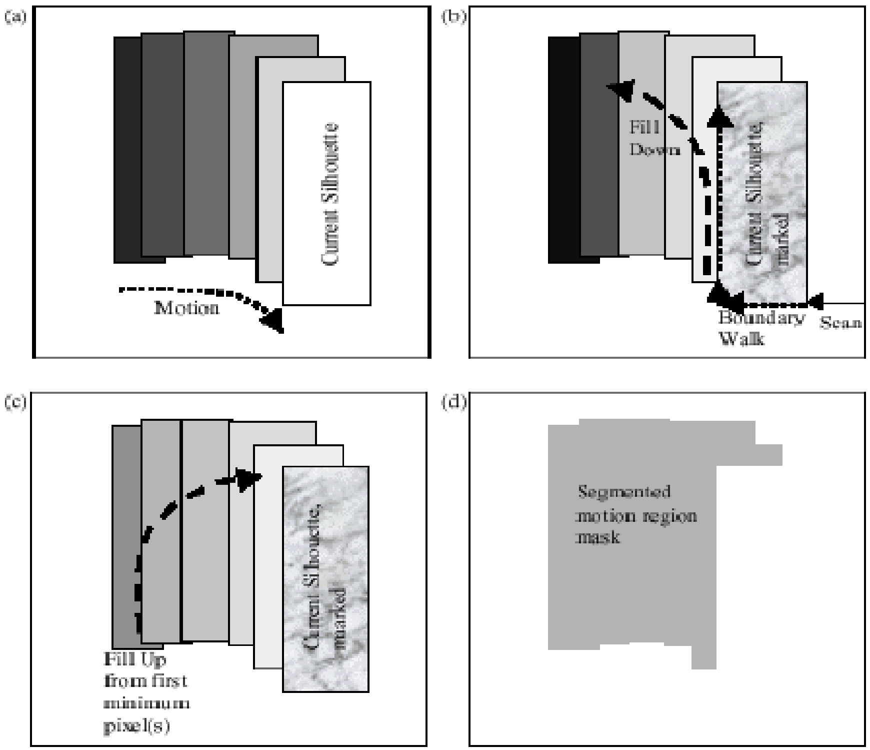
Optical flow

Active contours

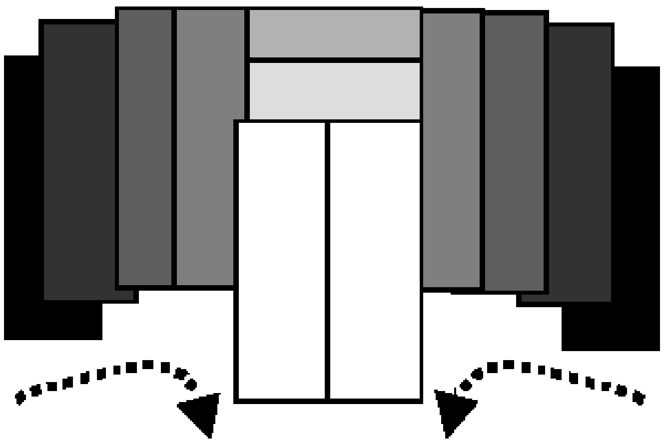
Estimators

Motion Segmentation Algorithm

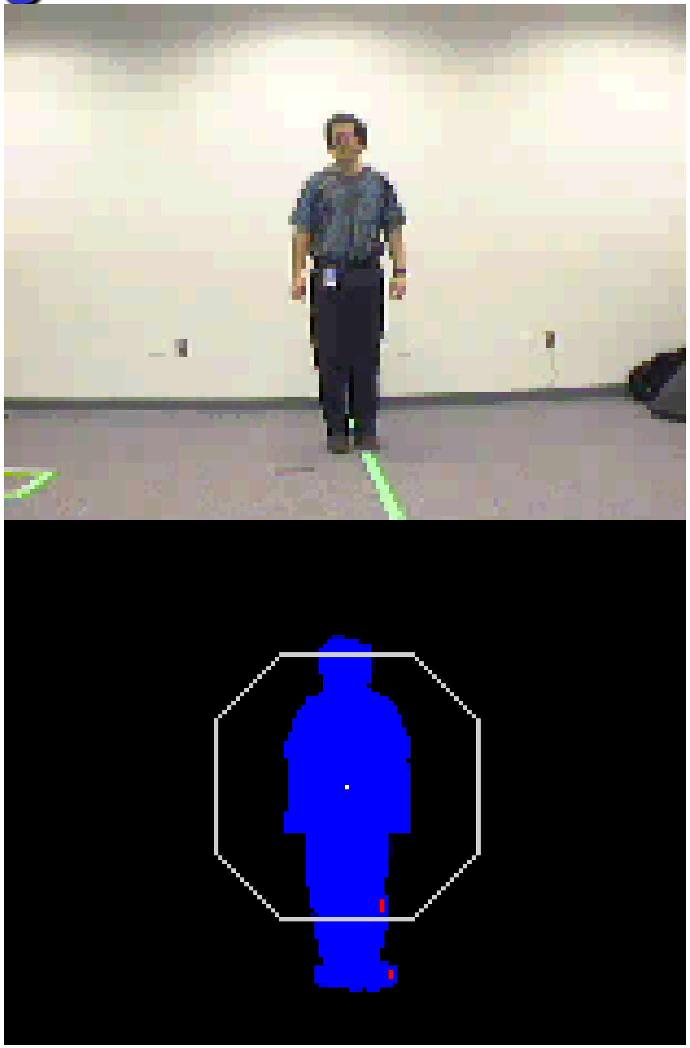
Two-pass algorithm labeling all motion segments



Clapping boxes together and down

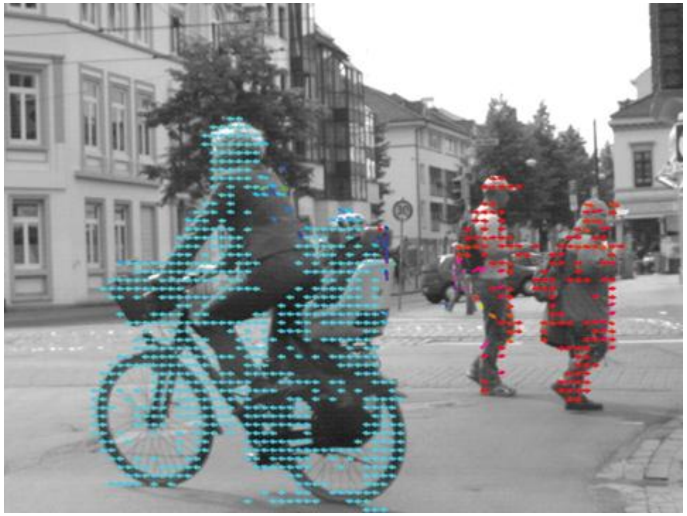


Motion Templates Example



- Motion templates allow to retrieve the dynamic characteristics of the moving object

Optical Flow



Block matching technique

Horn & Schunck technique

Lucas & Kanade technique

Pyramidal LK algorithm

6DOF (6 degree of freedom) algorithm

Optical flow equations:

$$I(x+dx, y+dy, t+dt) = I(x, y, t);$$

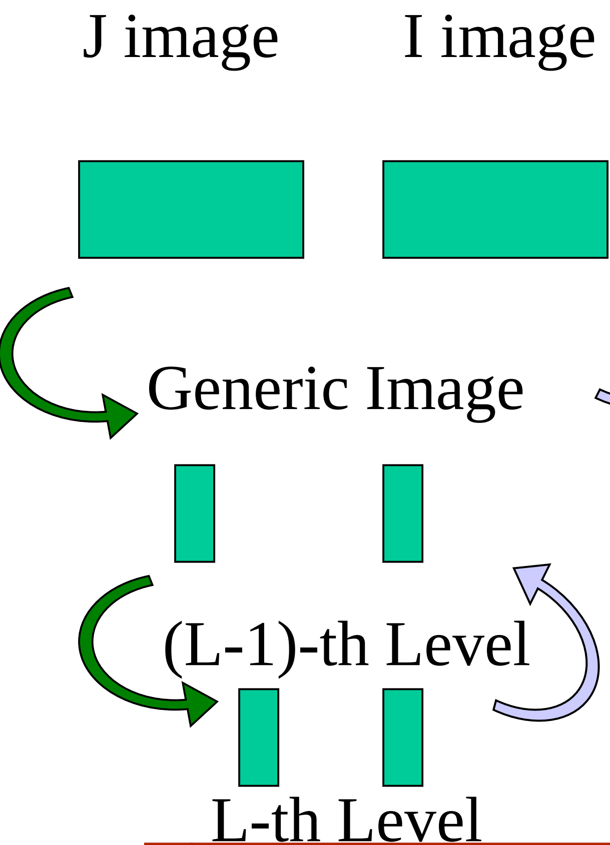
$$-\partial I / \partial t = \partial I / \partial x \cdot (dx / dt) + \partial I / \partial y \cdot (dy / dt);$$

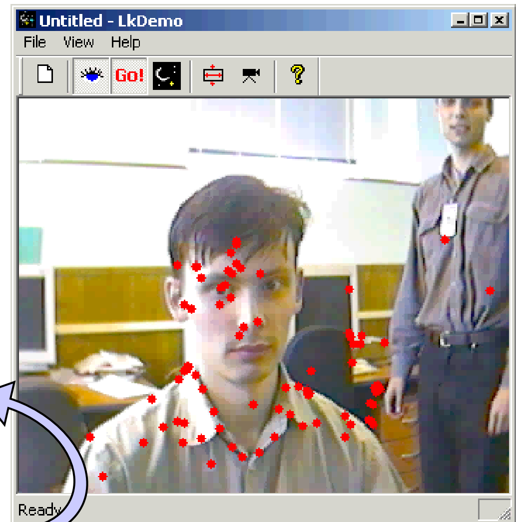
$$G \cdot \partial X = b,$$

$$\partial X = (\partial x, \partial y), G = \sum \begin{bmatrix} I_x^2, & I_x I_y \\ I_x I_y, & I_y^2 \end{bmatrix}, b = \sum I_t \begin{bmatrix} I_x \\ I_y \end{bmatrix}$$

Pyramidal Implementation of the optical flow algorithm

Image Pyramid Representation





Iterative Lucas – Kanade Scheme

Location of point u on image $u^L = u/2^L$

Spatial gradient matrix $G = \sum \begin{bmatrix} I_x^2, I_x I_y \\ I_x I_y, I_y^2 \end{bmatrix}$

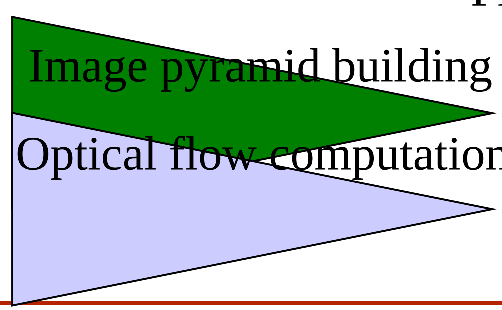
Standard Lucas – Kanade scheme for optical flow computation at level L d^L

Guess for next pyramid level L – 1

$$g^{L+1} = 2(g^L + d^L)$$

Finally, $d = d^0 + g^0$

$$V = U + d$$



6DOF Algorithm

Parametrical optical flow equations:

$$X = \pi(s).$$

$$\nabla I = \partial I / \partial s = \partial I / \partial X \cdot \partial \pi / \partial s$$

$$\sum_{i=1}^{N} \sum_{ROI} \nabla I_i^T \cdot \nabla I_i \cdot ds = \sum_{i=1}^{N} \sum_{ROI} I_{t_i} \cdot \nabla I_i^T$$

Active Contours

Snake energy: $$E = E_{\text{int}} + E_{ext}$$

Internal energy: $$E_{\text{int}} = E_{cont} + E_{curv}$$

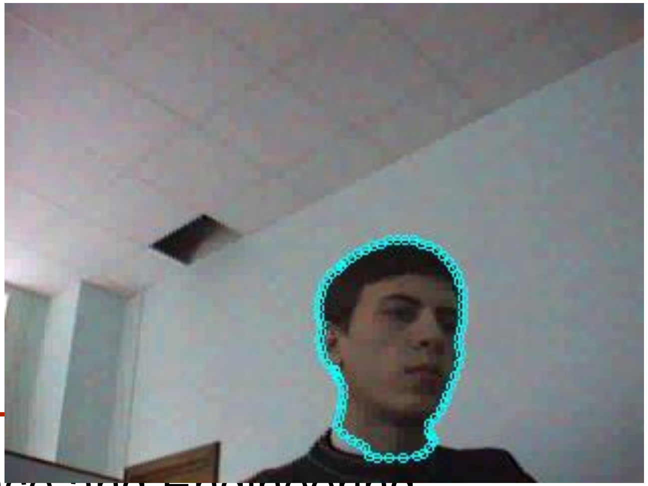
External energy:

Two external energy types: $$E_{ext} = E_{img} + E_{con}$$

$$E_{img} = -I,$$

$$E_{img} = -\left\| grad(I) \right\|,$$

$$E = \alpha \cdot E_{cont} + \beta \cdot E_{curv} + \gamma \cdot E_{img} \Rightarrow \min$$



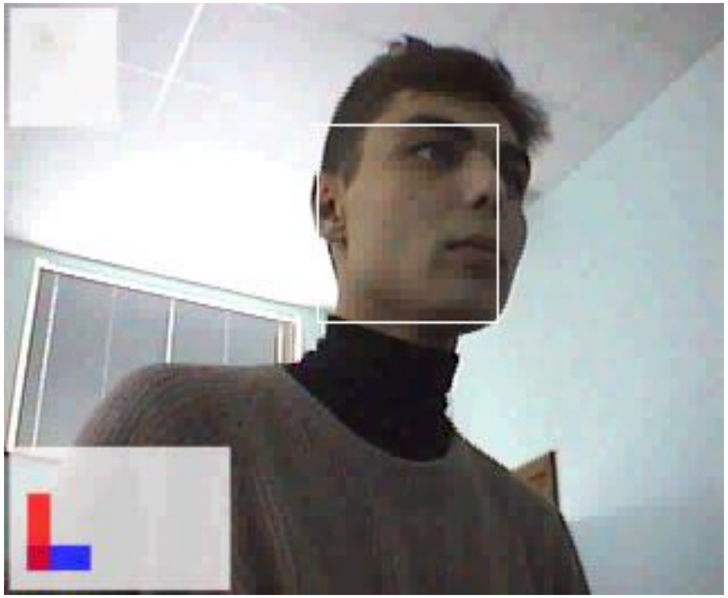
Estimators

Kalman filter

ConDensation filter

Kalman object tracker

The idea of using a Kalman filter for object tracking is to attenuate the noise associated with the position detection of the object based on estimating the system state. It can also be used to predict the position based on the state transition model when no new measurements are available



OpenCV Functionality

- ✓ Basic structures and operations
- ✓ Image Analysis
- ✓ Structural Analysis
- ✓ Object Recognition
- ✓ Motion Analysis and Object Tracking
- 3D Reconstruction

3D reconstruction

Camera Calibration

View Morphing

POSIT

Camera Calibration

Define intrinsic and extrinsic camera parameters.

Define Distortion parameters

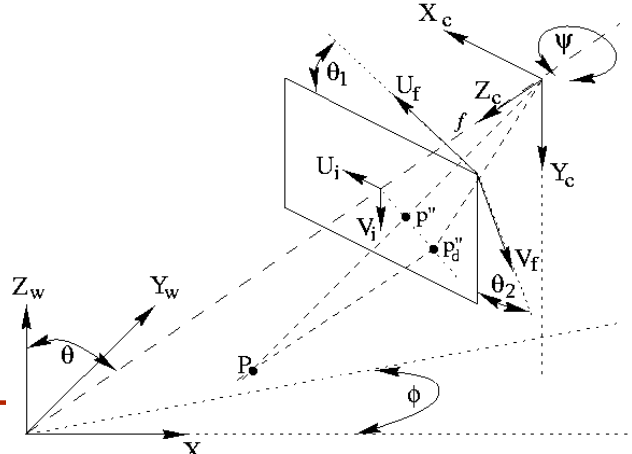
$$p = A[RT]P,$$

$$A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \quad R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}, \quad T = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}, \quad P = [X, Y, Z], \quad p = [u, v]$$

$$\tilde{u} = u + (u - c_x) \cdot [k_1 \cdot r^2 + k_2 \cdot r^4 + 2 p_1 y + p_2 (r^2 / x + 2x)],$$

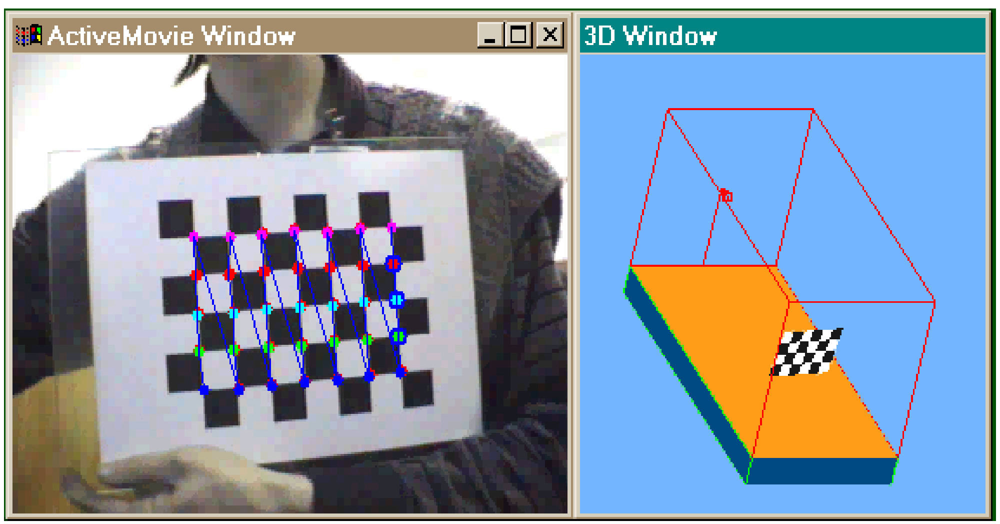
$$\tilde{v} = v + (v - c_y) \cdot [k_1 \cdot r^2 + k_2 \cdot r^4 + 2 p_2 x + p_1 (r^2 / y + 2y)],$$

$$r^2 = x^2 + y^2.$$



Camera Calibration

3D view of etalon



Un-distorted image

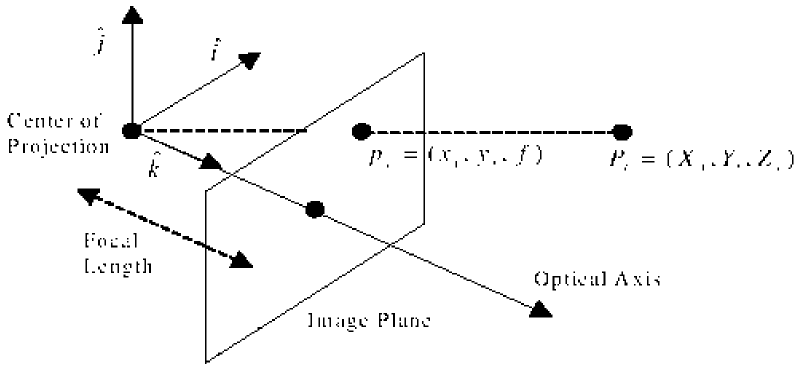
View Morphing

Original Image From Left Camera

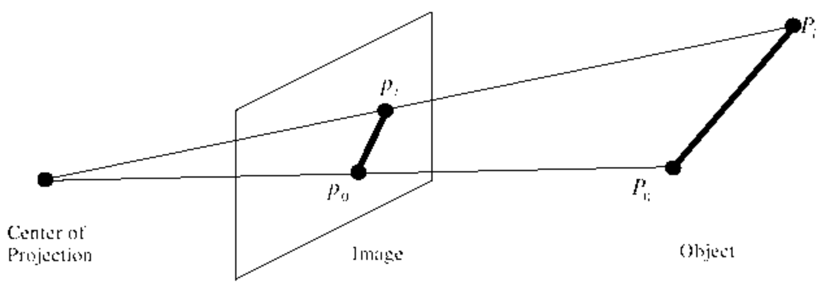
Original Image From Right Camera

POSIT Algorithm

Perspective projection: $x_i = (f / Z_i) \cdot X_i, \; y_i = (f / Z_i) \cdot Y_i$



- Weak-perspective projection: $x_i = s \cdot X_i, \; y_i = s \cdot Y_i, \; s = f / Z.$



OpenCV web sites

http://www.intel.com/research/mrl/research/opencv/

http://sourceforge.net

References

Gunilla Borgefors. *Distance Transformations in Digital Images.*Computer Vision, Graphics and Image Processing 34, 344-371,(1986).

G. Bradski and J. Davis. *Motion Segmentation and Pose Recognition with Motion History Gradients.* IEEE WACV'00, 2000.

P. J. Burt, T. H. Hong, A. Rosenfeld. *Segmentation and Estimation of Image Region Properties Through Cooperative Hierarchical Computation.* IEEE Tran. On SMC, Vol. 11, N.12, 1981, pp.802-809.

J.Canny.*A Computational Approach to Edge Detectio*n, IEEE Trans. on Pattern Analysis and Machine Intelligence, 8(6), pp.679-698 (1986).

J. Davis and Bobick. *The Representation and Recognition of Action Using Temporal Template*s. MIT Media Lab Technical Report 402,1997.

Daniel F. DeMenthon and Larry S. Davis. *Model-Based Object Pose in 25 Lines of Code.* In Proceedings of ECCV '92, pp. 335-343, 1992.

Andrew W. Fitzgibbon, R.B.Fisher. *A Buyer's Guide to Conic Fittin*g.Proc.5 th British Machine Vision Conference, Birmingham, pp. 513-522, 1995.

Berthold K.P. Horn and Brian G. Schunck. *Determining Optical Flo*w. Artificial Intelligence, 17, pp. 185-203, 1981.

References

M.Hu.*Visual Pattern Recognition by Moment Invariants*, IRE Transactions on Information Theory, 8:2, pp. 179-187, 1962.

B. Jahne. *Digital Image Processin*g. Springer, New York, 1997.

M. Kass, A. Witkin, and D. Terzopoulos. *Snakes: Active Contour Model*s, International Journal of Computer Vision, pp. 321-331, 1988.

J.Matas, C.Galambos, J.Kittler. *Progressive Probabilistic Hough Transfor*m. British Machine Vision Conference, 1998.

A. Rosenfeld and E. Johnston. *Angle Detection on Digital Curve*s. IEEE Trans. Computers, 22:875-878, 1973.

Y.Rubner.C.Tomasi,L.J.Guibas.*Metrics for Distributions with Applications to Image Databases.* Proceedings of the 1998 IEEE International Conference on Computer Vision, Bombay, India, January 1998, pp. 59-66.

Y. Rubner. C. Tomasi, L.J. Guibas. *The Earth Mover's Distance as a Metric for Image Retrieval.* Technical Report STAN-CS-TN-98-86, Department of Computer Science, Stanford University, September, 1998.

Y.Rubner.C.Tomasi.*Texture Metric*s. Proceeding of the IEEE International Conference on Systems, Man, and Cybernetics, San-Diego, CA, October 1998, pp. 4601- 4607. http://robotics.stanford.edu/~rubner/publications.html

References

J. Serra. *Image Analysis and Mathematical Morphology.* Academic Press, 1982.

Bernt Schiele and James L. Crowley. *Recognition without Correspondence Using Multidimensional Receptive Field Histograms.* In International Journal of Computer Vision 36 (1), pp. 31-50, January 2000.

S. Suzuki, K. Abe. *Topological Structural Analysis of Digital Binary Images by Border Following.* CVGIP, v.30, n.1. 1985, pp. 32-46.

C.H.Teh, R.T.Chin.*On the Detection of Dominant Points on Digital Curves.* - IEEE Tr. PAMI, 1989, v.11, No.8, p. 859-872.

Emanuele Trucco, Alessandro Verri. *Introductory Techniques for 3-D Computer Visio*n. Prentice Hall, Inc., 1998.

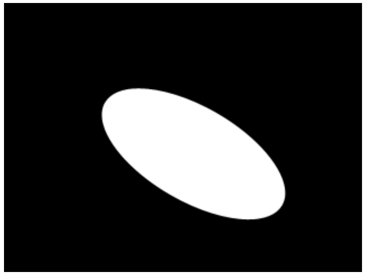
D. J. Williams and M. Shah. *A Fast Algorithm for Active Contours and Curvature Estimatio*n. CVGIP: Image Understanding, Vol. 55, No. 1, pp. 14-26, Jan., 1992. http://www.cs.ucf.edu/~vision/papers/shah/92/WIS92A.pdf.

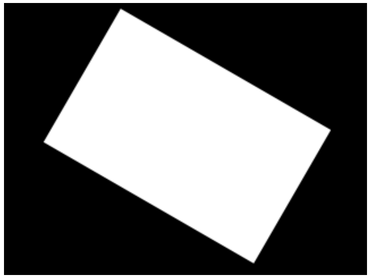
A.Y.Yuille, D.S.Cohen, and P.W.Hallinan. *Feature Extraction from Faces Using Deformable Templates in CVP*R, pp. 104-109, 1989.

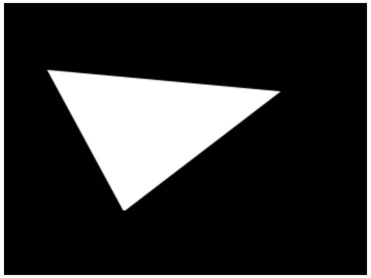
Zhengyou Zhang. *Parameter Estimation Techniques: A Tutorial with Application to Conic Fitting*, Image and Vision Computing Journal, 1996.

Using contours and geometry to classify shapes

Given the contour classify the geometrical figure shape (triangle, circle, etc)







OpenCV shape classification capabilities

Contour approximation

Moments (image&contour)

Convexity analysis

Pair-wise geometrical histogram

Fitting functions (line, ellipse)

Contour approximation

Min-epsilon approximation (Imai&Iri)

Min#-approximation (Douglas-Peucker method)

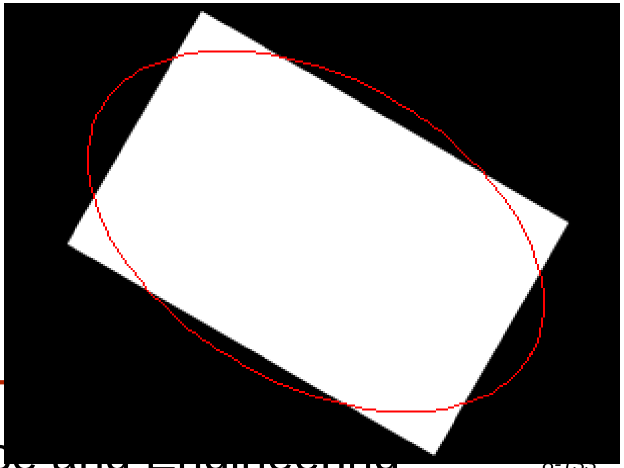
Moments

- Image moments (binary, grayscale)
- Contour moments (faster)
- Hu invariants

Line and ellipse fitting

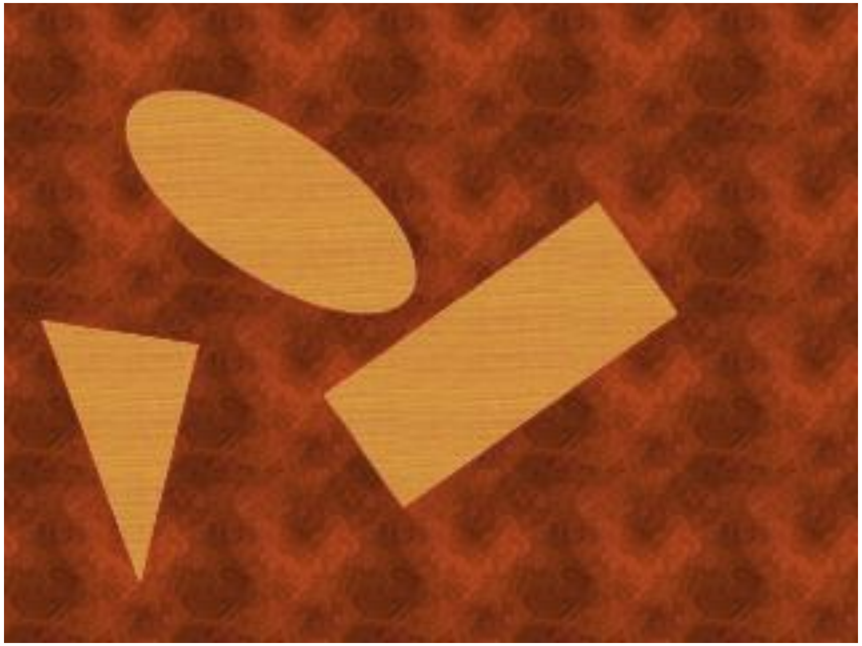
Algebraic ellipse fitting

Fitting lines by m-estimators



Using OpenCV to do color segmentation

Locate all nonoverlapping geometrical figures of the same unknown color



OpenCV segmentation capabilities

Edge-based approach

Histogram

Color segmentation

Edge-based segmentation

Smoothing functions (gaussian filter[IPL], bilateral filter)

Apply edge detector (sobel, laplace, canny, gradient strokes)

Find connected components in an inverted image

Pyramid segmentation

Water down the color space in order to join up the neighbor image pixels that are close to each other in XY and color spaces

Histogram

Calculate the histogram

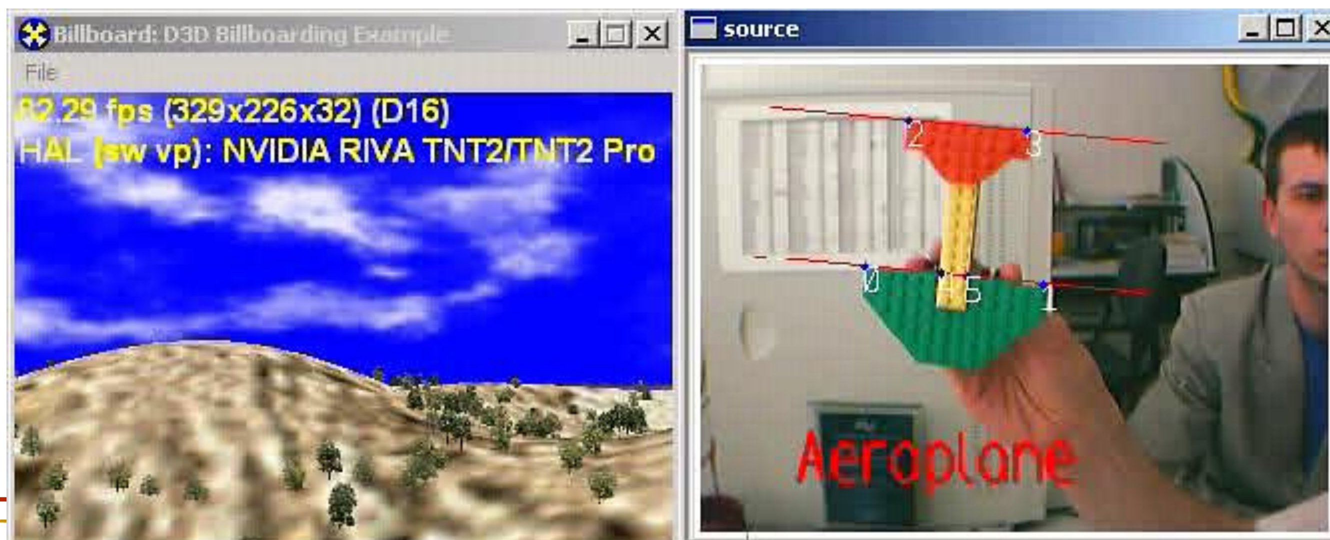
Separate the object and background histograms

Find the objects of the selected histogram in the image

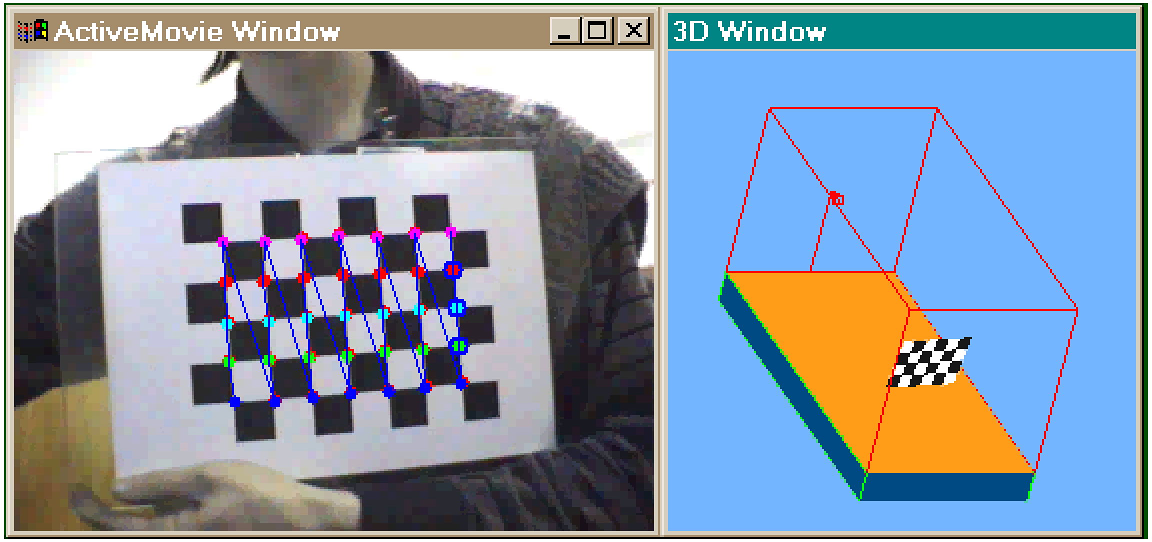
Using OpenCV to detect the 3D object's position

Calibrate the camera

Reconstruct the position and orientation of the rigid 3D body given it's geometry



Camera calibration routines, ActiveX



Reconstruction task

Given

camera model

3D coordinates of the feature points

and 2D coordinates corresponding projections on the image

Reconstruct the 3D position and orientation

Reconstruction task (continued)

POSIT algorithm for 3D objects

FindExtrinsicCameraParams for arbitrary objects

Technical content

- Software requirements
- OpenCV structure
- Data types
- Error Handling
- I/O libraries (HighGUI, CvCAM)
- Scripting
  - Hawk
  - Using OpenCV in MATLAB
- OpenCV lab (code samples)

Software Requirements

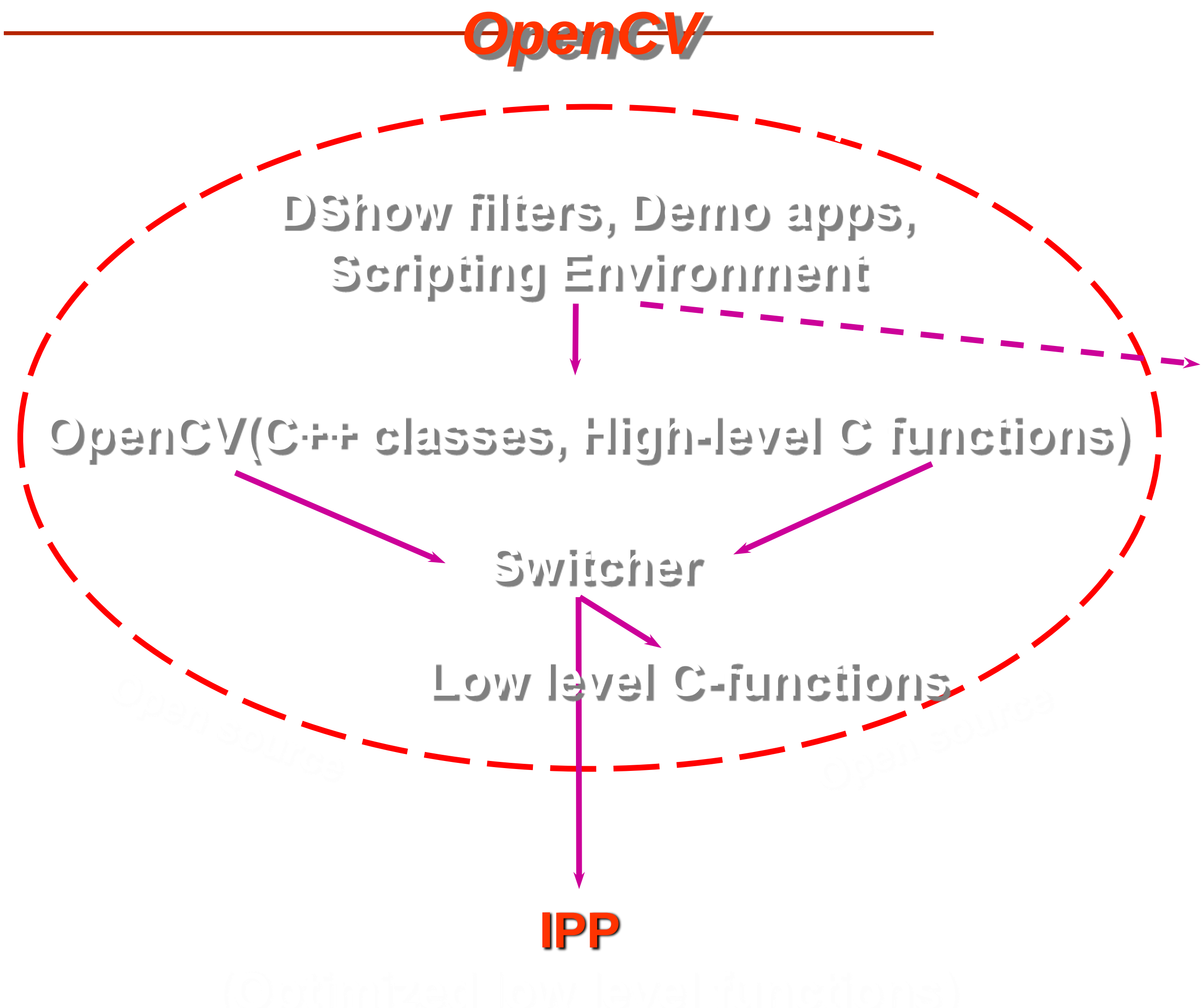
Win32 platforms:

- **Win9x/WinNT/Win2000**
- **C++ Compiler** (makefiles for Visual C++ 6.0,Intel C++ Compiler 5.x,Borland C++ 5.5, Mingw GNU C/C++ 2.95.3 are included) for core libraries
- Visual C++ to build the most of demos
- DirectX 8.x SDK for directshow filters
- ActiveTCL 8.3.3 for TCL demos
- IPL 2.2+ for the core library tests

Linux/*NIX:

- C++ Compiler (tested with GNU C/C++ 2.95.x, 2.96, 3.0.x)
- TCL 8.3.3 + BWidgets for TCL demos
- Video4Linux + Camera drivers for most of demos
- IPL 2.2+ for the core library tests

OpenCV structure



Data Types

Image (IplImage);

Matrix (CvMat);

Histogram (CvHistogram);

Dynamic structures (CvSeq, CvSet, CvGraph);

Spatial moments (CvMoments);

Helper data types (CvPoint, CvSize, CvTermCriteria, IplConvKernel and others).

Error Handling

There are no return error codes

There is a global error status that can be set or checked via special functions

By default a message box appears if error happens

Portable GUI library (HighGUI)

Reading/Writing images in several formats (BMP,JPEG,TIFF,PxM,Sun Raster)

Creating windows and displaying images in it. HighGUI windows remember their content (no need to implement repainting callbacks)

Simple interaction facilities: trackbars, getting input from keyboard

and mouse (new in Win32 version).

Portable Video Capture Library (CvCAM)

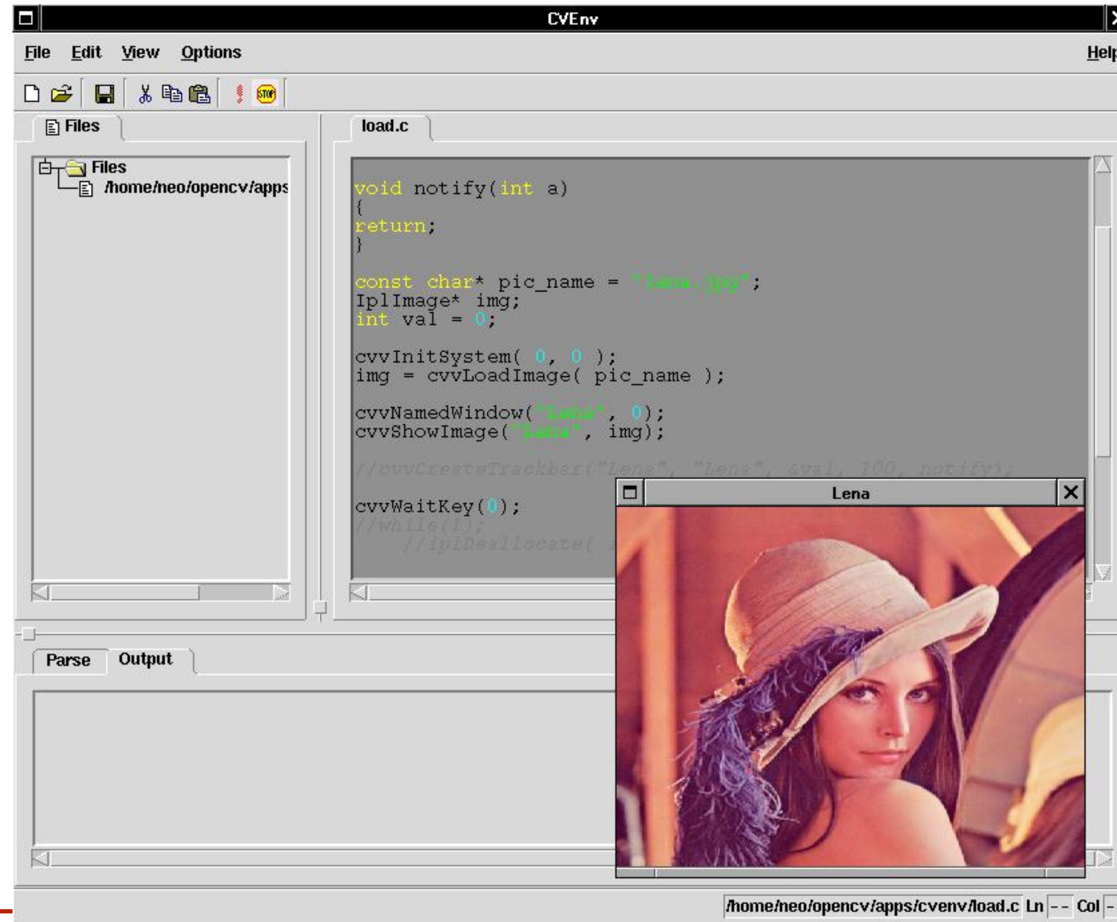
Single interface for video capture and playback under Linux and Win32

Provides callback for subsequent processing of frames from camera or AVI-file

Easy stereo from 2 USB cameras or stereo-camera

Scripting I: Hawk

- Visual Environment
- ANSI C interpreter (EiC) as a core
- Plugin support
- Interface to OpenCV,IPL and HighGUI via plugins
- Video support



Scripting II: OpenCV + MATLAB

Design principles and data types organization

Working with images

Working with dynamic structures

Example

Design Principles and Data Types Organization

Simplicity: Use of native MATLAB types (matrices, structures), rather than introducing classes

Compatibility: … with Image Processing Toolbox

Irredundancy: matrix and basic image processing operations are not wrapped

[dst …] = cv*<func>(src …)*

↕

// data type conv., error handling

void mexFunction (…) { … }

↕

cvFunc*(src …, dst …) {…}*

Working with Images

% erosion with 3x3 rectangular element

% strong corners detection (quality level = 0.1, min distance = 10)

% Optical Flow on pyramids: window 10*2+1x10*2+1, 4 scales

% Color object tracking, default termination criteria (epsilon = 1):

Working with Dynamic Structures

% get all the connected components of binary image,
% don't approximate them

% get bounding box of the first contour
% get the first child of the second contour
% get Nx2 array of vertices of the child
% draw the child contour
% on the image with green
% approximate all contours using Douglas-Peucker method with accuracy = 2.

% compare contours via pair-wise histogram comparison

Example:

```
% Camshift tracker, enhanced with noise filter

function new_window = track_obj( img, obj_hist, window, thresh )
    probimg = cvcalcbackproject( img, obj_hist );
    probimg = cvclose( probimg, 3, 2 ); % remove small holes via morphological
     'close' operation
    probimg = cvthresh( probimg, thresh );
     contours = cvfindcontours( probimg, 'external' );
     mask_img = zeros(size(contours));
     for i = 1:length(contours)
          if  contous(i).rect(3)*contous(i).rect(4) < 30
             contours(i).pt = []; % remove small contours;
          end
     end
     mask_img = cvfillcontours( mask_img, contours, 'w' );
     new_window = cvcamshift( mask_img, window );
```

Victor Eruhimov:

Questions?

Trainings

Go to lab…