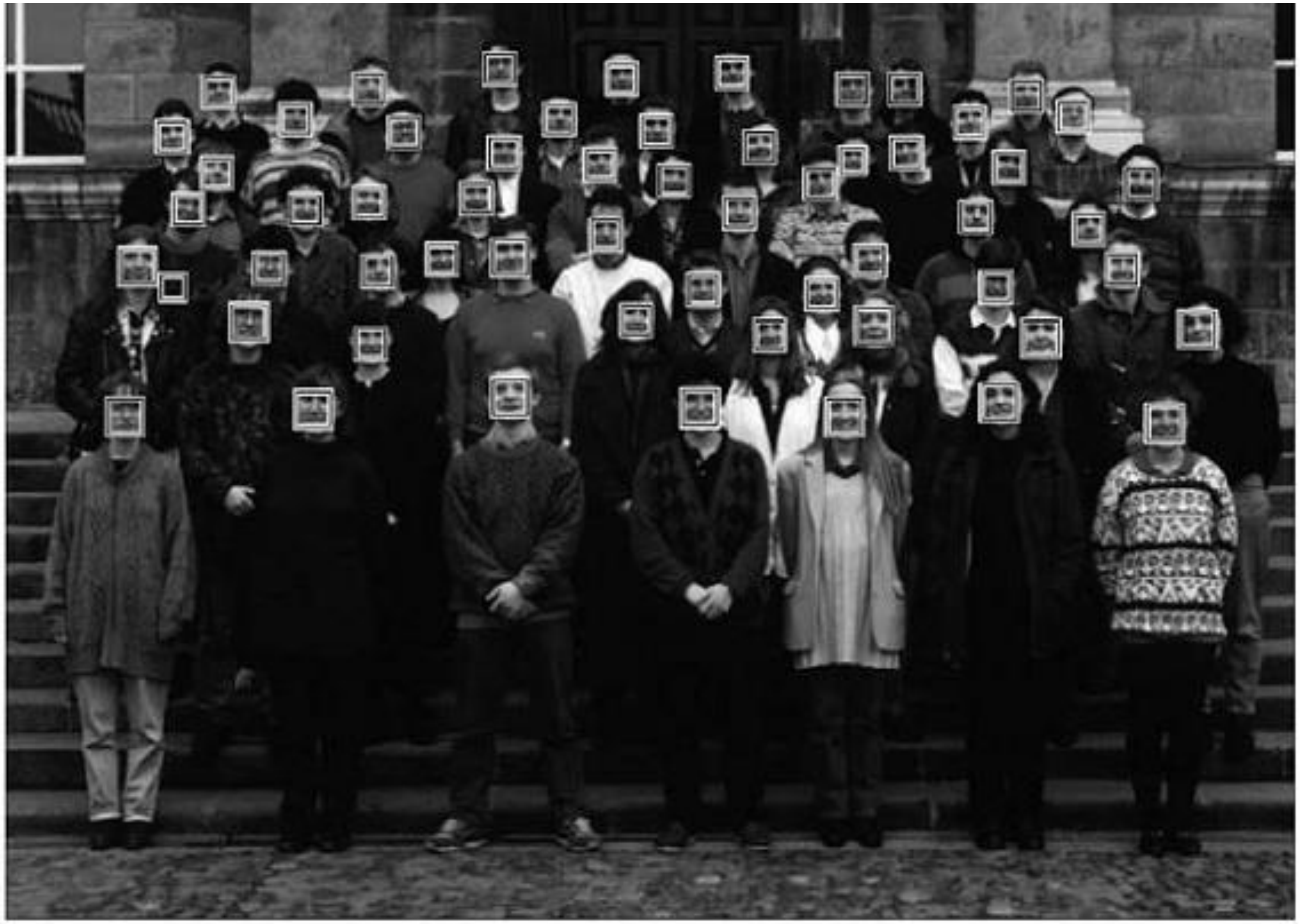
PHÁT GIÁC VÀ NHẬN DẠNG ẢNH

DETECT AND RECOGNITION OBJECT

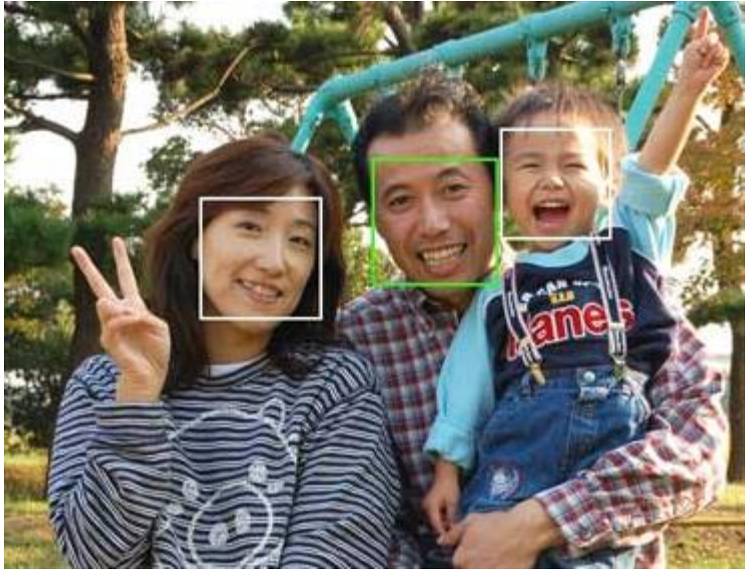
KHÁI NiỆM

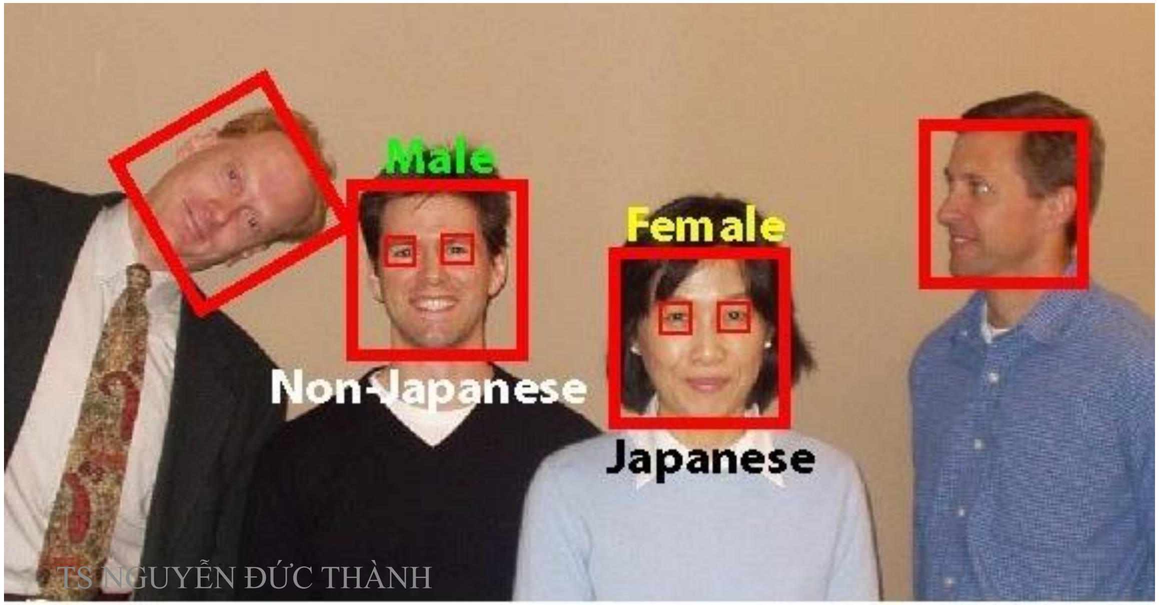
- Nhận dạng ảnh là tìm một hay nhiều vật trong ảnh dựa theo mô hình đã biết trước (pattern)
- Có thể nhận dạng ảnh theo hai lớp hoặc nhiều lớp
- Có rất nhiều ứng dụng cần nhận dạng ảnh
  - Face detection, face recognition, facial expression detection, hand gesture recognition, eye gaze tracking, motion tracking, smile recognition….
  - Fingerprint recognition, retina recognition, optical character recognition OCR, licence plate recognition…
- Có rất nhiều phương pháp nhận dạng

FACE DETECTION

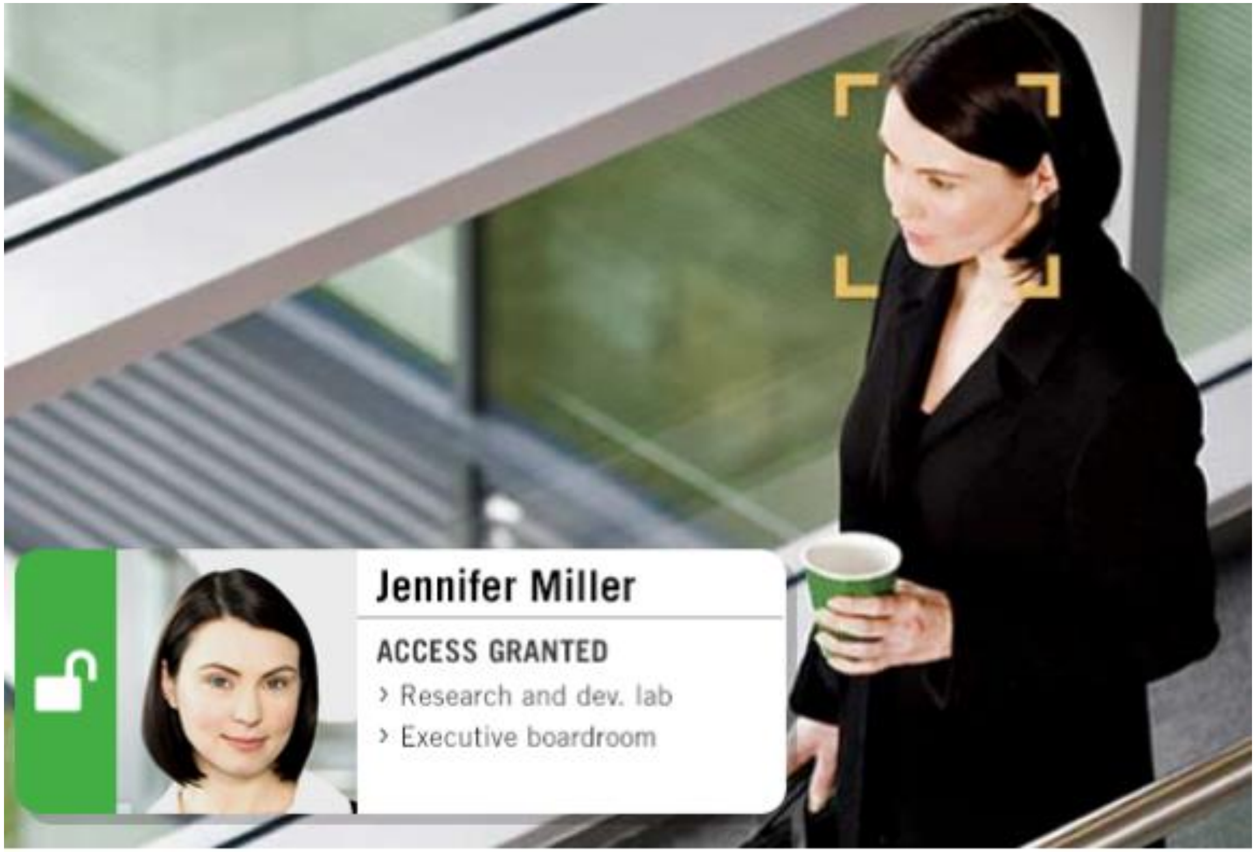


Face detection Máy ảnh số



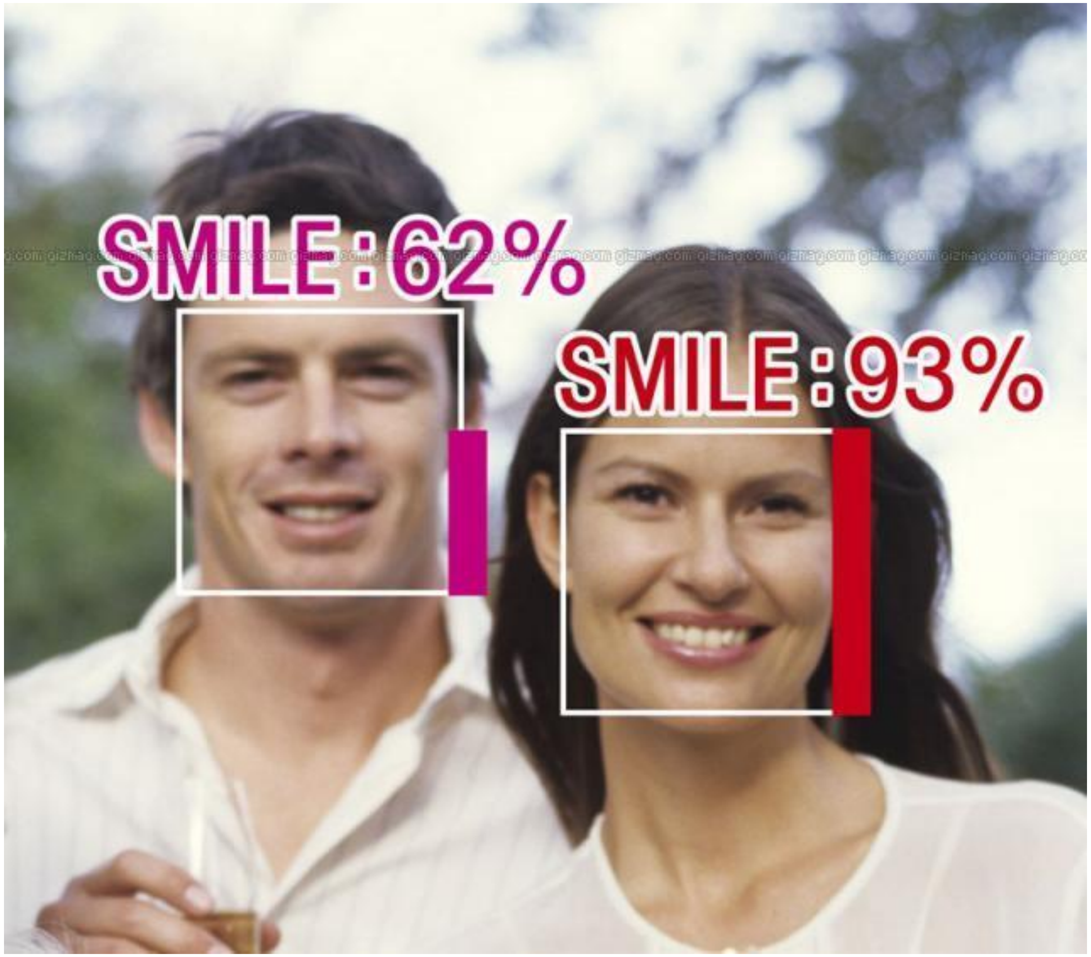


Face Recognition

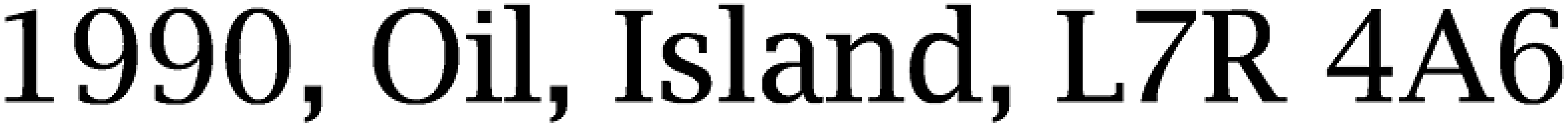


Face recognition can confirm identity. It is therefore used to control access to sensitive areas.

Nhận dạng nụ cười Máy ảnh số



NHẬN DẠNG CHỮ
OCR OPTICAL CHARACTER RECOGNITION

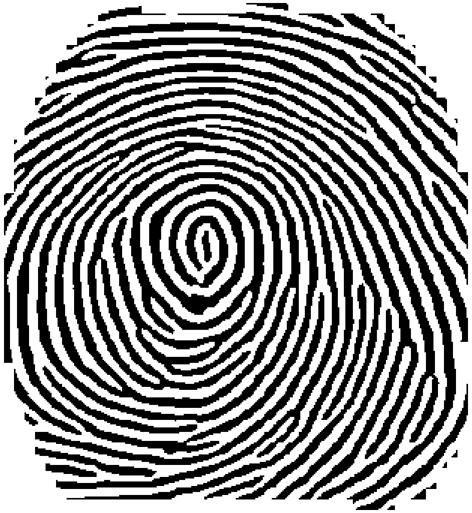


1990, Oil, Island, L7R 4A6

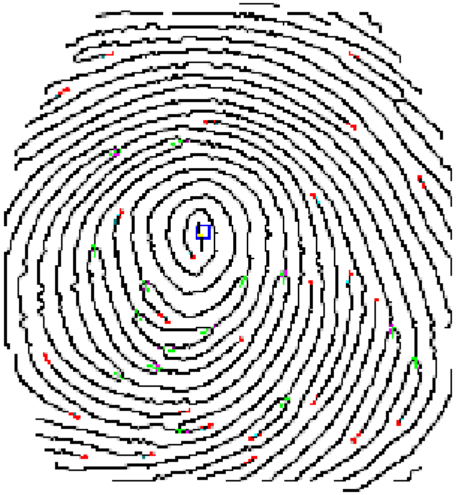
Phân biệt số 1 chữ I và l; số 0 và chữ O

SINH TRẮC BIOMETRIC FINGERPRINT RECOGNITION

Original image

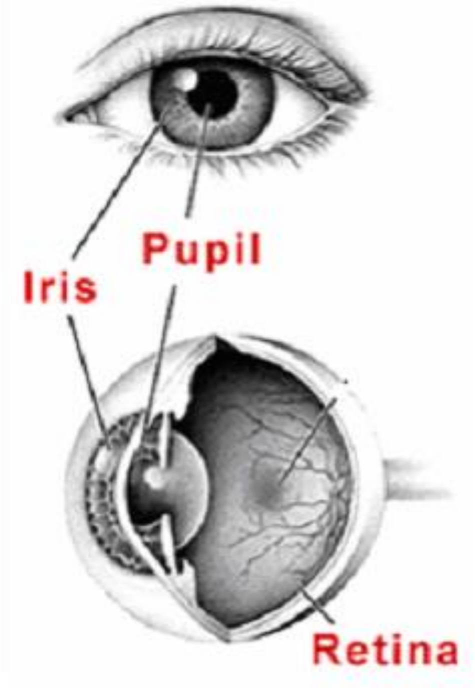


Binary image



Skeleton and extracted minutiae

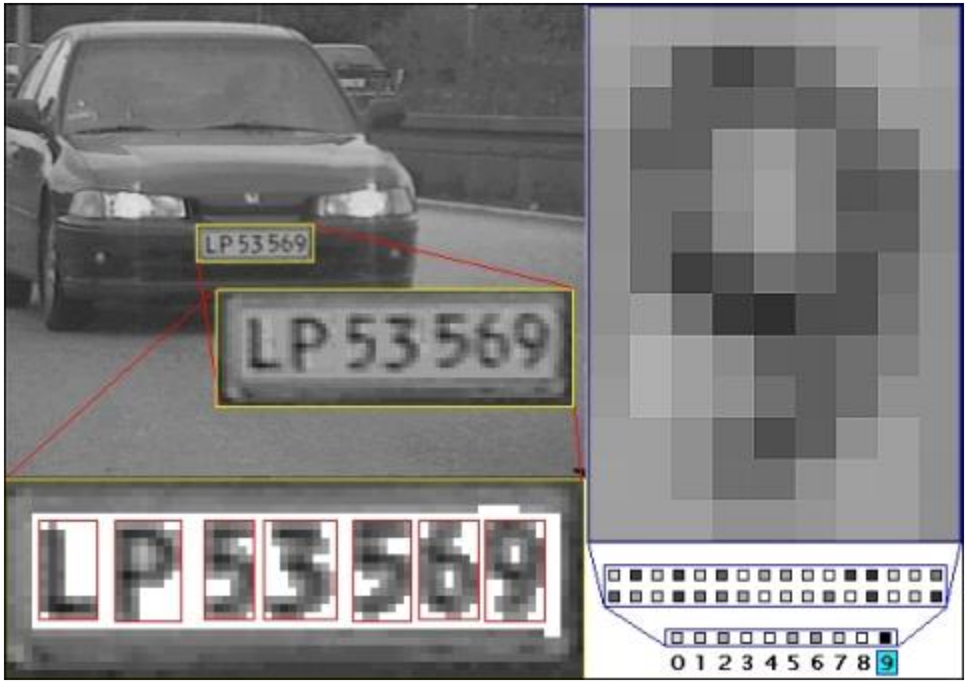
Nhận dạng mống mắt Iris (tròng đen)

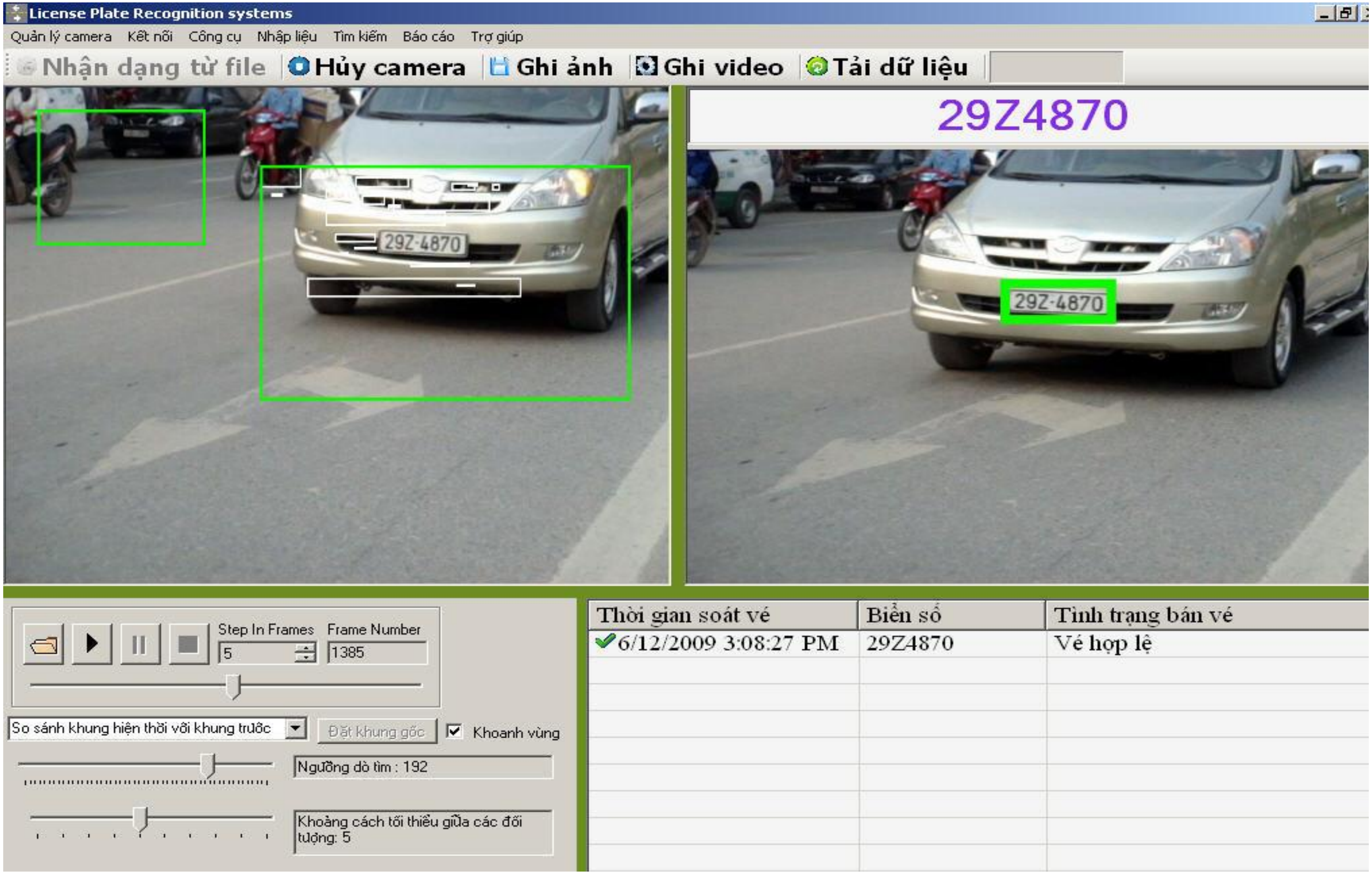


Iris Diagram

Iris Structure

BÃI XE TRẠM THU PHÍ, TRẠM CÂN
License Plate Recognition





29Z4870

Step In Frames: 5

Frame Number: 1385

So sánh khung hiện thời với khung trước

☑ Khoanh vùng

Ngưỡng dò tìm : 192

Khoảng cách tối thiểu giữa các đối tượng: 5

Thời gian soát vé	Biển số	Tình trạng bán vé
✓6/12/2009 3:08:27 PM	29Z4870	Vé hợp lệ

NHẬN DẠNG CỬ CHỈ BÀN TAY

TV CAMERA STC1100

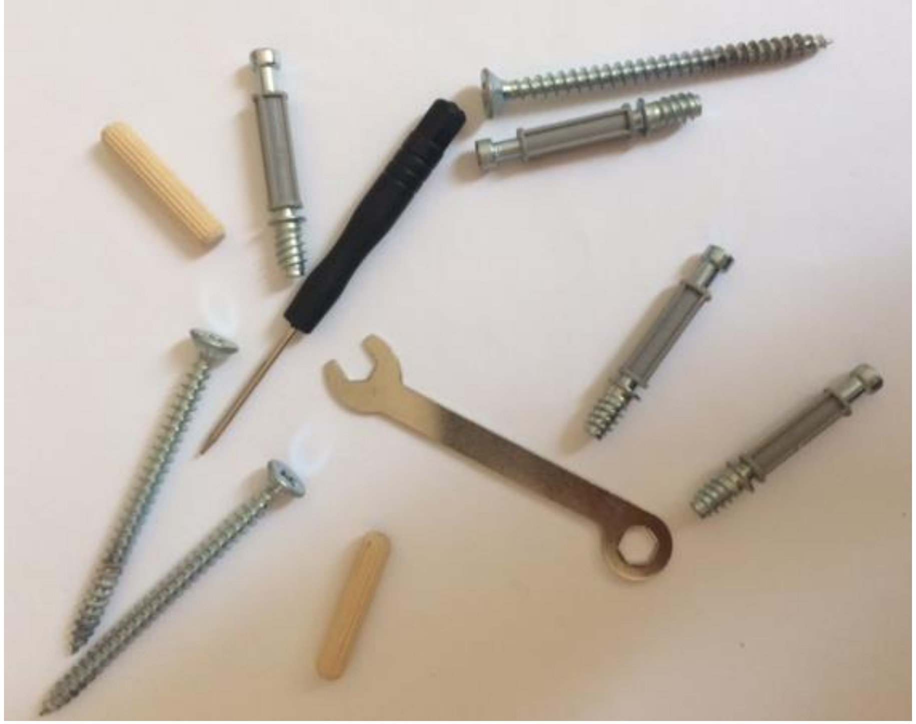
KINECT

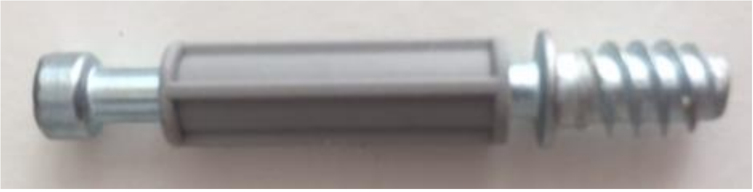
CÁC PHƯƠNG PHÁP PHÁT HIỆN ĐỐI TƯỢNG

- Đối sánh mẫu Template matching
- Dùng contour và Hu Moment
- Đối sánh điểm đặc trưng Point Feature Matching
- Tách tiền cảnh và phân tích đốm Foreground detector and blob analysis Gaussian Mixture Models
- Phân lớp theo tầng Cascade Classifier
- Histogram of Oriented Gradients HOG
- SVM Support Vector Machine
- Dùng mạng nơ rôn Artificial Neural Network
- Dùng deep learning
- …

TEMPLATE MATCHING

- Ta có ảnh của vật mẫu và cần tìm vật đó trong ảnh bằng cách so sánh mẫu





cv::matchTemplate

- Trượt ảnh mẫu T trên ảnh cần tìm mẫu T, tính giá trị tương quan R, nếu R tốt nhất kết luận đó là đối tượng cần tìm và đóng khung vùng đó
- C++: void matchTemplate(InputArray image, InputArray templ, OutputArray result, int method)
- Python: cv2.matchTemplate(image, templ, method[, result]) → result
- C: void cvMatchTemplate(const CvArr* image, const CvArr* templ, CvArr* result, int method)
- Python: cv.MatchTemplate(image, templ, result, method) → None

cv::matchTemplate

- image – Image where the search is running. It must be 8-bit or 32-bit floating-point.
- templ – Searched template. It must be not greater than the source image and have the same data type.
- result – Map of comparison results. It must be single-channel 32-bit floating-point. If image is W \times H and templ is w *h , then result is (W-w+1*(H-h+1) .
- method – Parameter specifying the comparison method: square difference, correlation coefficient

cv::matchTemplate

CV_TM_SQDIFF

$$R(x,y) = \sum_{x',y'} (T(x',y') - I(x+x',y+y'))^2$$

CV_TM_SQDIFF_NORMED

$$R(x,y) = \frac{\sum_{x',y'} (T(x',y') - I(x+x',y+y'))^2}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

CV_TM_CCORR

$$R(x,y) = \sum_{x',y'} (T(x',y') \cdot I(x+x',y+y'))$$

CV_TM_CCORR_NORMED

$$R(x,y) = \frac{\sum_{x',y'}(T(x',y') \cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

CV_TM_CCOEFF

$$R(x,y) = \sum_{x',y'} (T'(x',y') \cdot I(x+x',y+y'))$$

$$T'(x',y') = T(x',y') - 1/(w \cdot h) \cdot \textstyle\sum_{x'',y''} T(x'',y'')$$
$$I'(x+x',y+y') = I(x+x',y+y') - 1/(w \cdot h) \cdot \textstyle\sum_{x'',y''} I(x+x'',y+y'')$$

CV_TM_CCOEFF_NORMED

$$R(x,y) = \frac{\sum_{x',y'}(T'(x',y') \cdot I'(x+x',y+y'))}{\sqrt{\sum_{x',y'} T'(x',y')^2 \cdot \sum_{x',y'} I'(x+x',y+y')^2}}$$

minMaxLoc

Finds the global minimum and maximum in an array.

C++: void minMaxLoc(InputArray src, double* minVal, double* maxVal=0, Point* minLoc=0, Point* maxLoc=0, InputArray mask=noArray())

C++: void minMaxLoc(const SparseMat& a, double* minVal, double* maxVal, int* minIdx=0, int* maxIdx=0)

Python: cv2.minMaxLoc(src[, mask]) → minVal, maxVal, minLoc, maxLoc

C: void cvMinMaxLoc(const CvArr* arr, double* min_val, double* max_val, CvPoint* min_loc=NULL, CvPoint* max_loc=NULL, const CvArr* mask=NULL)

Python: cv.MinMaxLoc(arr, mask=None)-> (minVal, maxVal, minLoc, maxLoc)

cv::matchTemplate

- After the function finishes the comparison, the best matches can be found as global minimums (when CV_TM_SQDIFF was used) or maximums (when CV_TM_CCORR or CV_TM_CCOEFF was used) using the minMaxLoc() function. In case of a color image, template summation in the numerator and each sum in the denominator is done over all of the channels and separate mean values are used for each channel. The result will still be a single-channel image, which is easier to analyze.
- Loads an input image and a image patch (*template*)
- Perform a template matching procedure by using the OpenCV function matchTemplate with any of the 6 matching methods described before. The user can choose the method by entering its selection in the Trackbar.

TEMPLATE MATCHING C++

- Normalize the output of the matching procedure
- Localize the location with higher matching probability
- Draw a rectangle around the area corresponding to the highest match

```
/// Global Variables
Mat img; Mat templ; Mat result;
char* image_window = "Source Image";
char* result_window = "Result window";
int match_method;
int max_Trackbar = 5;
/// Function Headers
void MatchingMethod( int, void* );
int main( int argc, char** argv )
{
```

TEMPLATE MATCHING C++

```
char* trackbar_label = "Method: \n 0: SQDIFF \n 1: SQDIFF
NORMED \n 2: TM CCORR \n 3: TM CCORR NORMED \n 4:
TM COEFF \n 5: TM COEFF NORMED";
 createTrackbar( trackbar_label, image_window, &match_method,
max_Trackbar, MatchingMethod );
  MatchingMethod( 0, 0 );

   waitKey(0);
  return 0; }
void MatchingMethod( int, void* )
{  /// Source image to display
  Mat img_display;
  img.copyTo( img_display );
```

TEMPLATE MATCHING C++

```
int result_cols =  img.cols - templ.cols + 1;
  int result_rows = img.rows - templ.rows + 1;
  result.create( result_rows, result_cols, CV_32FC1 );
  /// Do the Matching and Normalize
  matchTemplate( img, templ, result, match_method );
  normalize( result, result, 0, 1, NORM_MINMAX, -1, Mat() );
  /// Localizing the best match with minMaxLoc
  double minVal; double maxVal; Point minLoc; Point maxLoc;
  Point matchLoc;
  minMaxLoc( result, &minVal, &maxVal, &minLoc, &maxLoc, Mat() );
```

TEMPLATE MATCHING C++

```
if( match_method  == CV_TM_SQDIFF || match_method ==
CV_TM_SQDIFF_NORMED )
    { matchLoc = minLoc; }
  else
    { matchLoc = maxLoc; }
/// Show me what you got
  rectangle( img_display, matchLoc, Point( matchLoc.x + templ.cols ,
matchLoc.y + templ.rows ), Scalar::all(0), 2, 8, 0 );
  rectangle( result, matchLoc, Point( matchLoc.x + templ.cols ,
matchLoc.y + templ.rows ), Scalar::all(0), 2, 8, 0 );
  imshow( image_window, img_display );
  imshow( result_window, result );  return;}
```

TEMPLATE MATCHING Python

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('messi5.jpg',0)
img2 = img.copy()
template = cv2.imread('template.jpg',0)
w, h = template.shape[::-1]
# All the 6 methods for comparison in a list
methods = ['cv2.TM_CCOEFF', 'cv2.TM_CCOEFF_NORMED', 'cv2.TM_CCORR',
            'cv2.TM_CCORR_NORMED', 'cv2.TM_SQDIFF', 'cv2.TM_SQDIFF_NORMED']
```

TEMPLATE MATCHING Python

```
for meth in methods:
    img = img2.copy()
    method = eval(meth)
    # Apply template Matching
    res = cv2.matchTemplate(img,template,method)
    min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)
    # If the method is TM_SQDIFF or TM_SQDIFF_NORMED, take minimum
    if method in [cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]:
        top_left = min_loc
```

TEMPLATE MATCHING Python

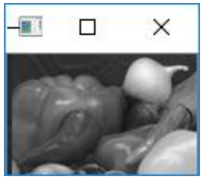
```
else:
        top_left = max_loc
    bottom_right = (top_left[0] + w, top_left[1] + h)
    cv2.rectangle(img,top_left, bottom_right, 255, 2)
    plt.subplot(121),plt.imshow(res,cmap = 'gray')
    plt.title('Matching Result'), plt.xticks([]), plt.yticks([])
    plt.subplot(122),plt.imshow(img,cmap = 'gray')
    plt.title('Detected Point'), plt.xticks([]), plt.yticks([])
    plt.suptitle(meth)
    plt.show()
```

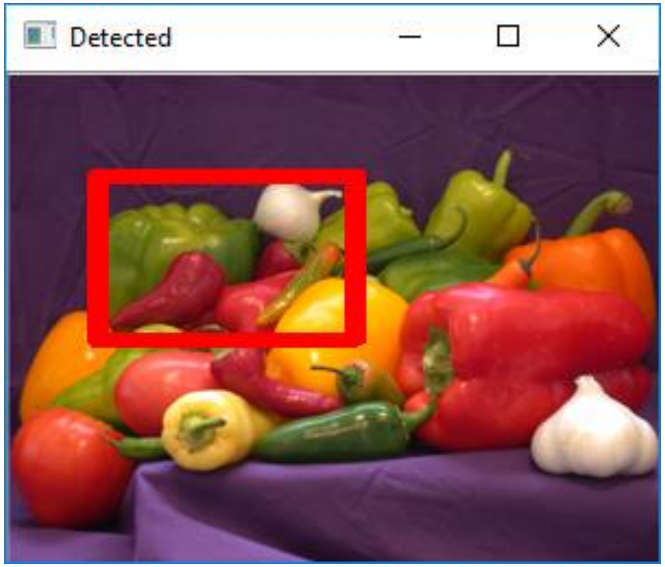
TEMPLATE MATCHING PYTHON

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img_rgb = cv2.imread('d:/Mainimage.png')
cv2.imshow('parts',img_rgb)
img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_BGR2GRAY)
template = cv2.imread('d:/template.png',0)
cv2.imshow('Partsample',template)
w, h = template.shape[::-1]
res = cv2.matchTemplate(img_gray,template,cv2.TM_CCOEFF_NORMED)
```

TEMPLATE MATCHING PYTHON

```
threshold = 0.8
loc = np.where( res >= threshold)
for pt in zip(*loc[::-1]):
    cv2.rectangle(img_rgb, pt, (pt[0] + w, pt[1] + h), (0,0,255), 1)
cv2.imshow('Detected',img_rgb)
```





vision.TemplateMatcher MATLAB

- tMatcher = vision.TemplateMatcher returns a template matcher System object, tMatcher. This object performs template matching by shifting a template in single-pixel increments throughout the interior of an image.
- tMatcher = vision.TemplateMatcher(Name,Value) sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, tMatcher = vision.TemplateMatcher('Metric','Sum of absolute differences')
- Metric — Metric used for template matching source
- 'Sum of absolute differences' (default) | 'Sum of squared differences' | 'Maximum absolute difference‘
- OutputValue — Type of output

'Best match location' (default) | 'Metric matrix‘

vision.TemplateMatcher MATLAB

- SearchMethod — Specify search criteria to find minimum difference between two inputs 'Exhaustive' (default) | 'Three-step'
- BestMatchNeighborhoodOutputPort — Enable metric values output false (default) | true
- NeighborhoodSize — Size of the metric values

3 (default) | odd number

- ROIInputPort — Enable ROI specification through input

false (default) | true

- ROIValidityOutputPort — Enable output of a flag indicating if any part of ROI is outside input image false (default) | true

vision.TemplateMatcher MATLAB

- location = tMatcher(I,T) computes the [x y] location coordinates, location, of the best template match between the image matrix, I, and the template matrix, T. The output coordinates are relative to the top left corner of the image. The object computes the location by shifting the template in single-pixel increments throughout the interior of the image.
- location = tMatcher(I,T) returns the location of the best template match LOC, the metric values around the best match NVALS, and a logical flag NVALID. A false value for NVALID indicates that the neighborhood around the best match extended outside the borders of the metric value matrix NVALS. This applies when you set the OutputValue property to Best match location and the BestMatchNeighborhoodOutputPort property to true.

vision.TemplateMatcher MATLAB

- [location,Nvals,Nvalid] = tMatcher(I,T,ROI)returns the location of the best template match location, the metric values around the best match Nvals, and a logical flag Nvalid. This applies when you set the OutputValue property to 'Best match location' and the BestMatchNeighborhoodOutputPort property to true.
- [location,Nvals,Nvalid,ROIvalid] = tMatcher(I,T,ROI) also returns a logical flag, ROIvalid to indicate whether the ROI is outside the bounds of the input image I. This applies when you set the OutputValue property to 'Best match location', and the BestMatchNeighborhoodOutputPort, ROIInputPort, and ROIValidityOutputPort properties to true.

vision.TemplateMatcher MATLAB

- [location,ROIvalid] = tMatcher(I,T,ROI)also returns a logical flag ROIvalid indicating if the specified ROI is outside the bounds of the input image I. This applies when you set the OutputValue property to 'Best match location', and both the ROIInputPort and ROIValidityOutputPort properties to true.
- I — Input image, truecolor | M-by-N 2-D grayscale image
- T — Template, binary image | truecolor | M-by-N 2-D grayscale image
- ROI — Input ROI, four-element vector

Video Stabilization

% We first define the target to track. In this case, it is the back of a car and the license plate. We also establish a dynamic search region, whose position is determined by the last known target location. We then search for the target only within this search region, which reduces the number of computations required to find the target. In each subsequent video frame, we determine how much the target has moved relative to the previous frame. We use this information to remove unwanted translational camera motions and generate a stabilized video.

```
% Input video file which needs to be stabilized.
filename = 'shaky_car.avi';
hVideoSource = vision.VideoFileReader(filename, ...
                    'ImageColorSpace', 'Intensity',...
                    'VideoOutputDataType', 'double');
```

Video Stabilization

```
hTM = vision.TemplateMatcher('ROIInputPort', true,
'BestMatchNeighborhoodOutputPort', true);
%Create a System object to display the original and the stabilized video.
hVideoOut = vision.VideoPlayer('Name', 'Video Stabilization');
hVideoOut.Position(1) = round(0.4*hVideoOut.Position(1));
hVideoOut.Position(2) = round(1.5*(hVideoOut.Position(2)));
hVideoOut.Position(3:4) = [650 350];
%initialize some variables used in the processing loop.
pos.template_orig = [109 100]; % [x y] upper left corner
pos.template_size = [22 18];   % [width height]
pos.search_border = [15 10];   % max horizontal and vertical disp
pos.template_center = floor((pos.template_size-1)/2);
```

Video Stabilization

```
pos.template_center_pos = (pos.template_orig + os.template_center -
1);
fileInfo = info(hVideoSource);
W = fileInfo.VideoSize(1); % Width in pixels
H = fileInfo.VideoSize(2); % Height in pixels
BorderCols = [1:pos.search_border(1)+4 W-
pos.search_border(1)+4:W];
BorderRows = [1:pos.search_border(2)+4 H-
pos.search_border(2)+4:H];
sz = fileInfo.VideoSize;
```

Video Stabilization

```
TargetRowIndices =   pos.template_orig(2)-
1:pos.template_orig(2)+pos.template_size(2)-2;
TargetColIndices =   pos.template_orig(1)-
1:pos.template_orig(1)+pos.template_size(1)-2;
SearchRegion = pos.template_orig - pos.search_border - 1;
Offset = [0 0];
Target = zeros(18,22);
firstTime = true;
```

Video Stabilization

```
%This is the main processing loop which uses the objects we instantiated above to stabilize the input video.
while ~isDone(hVideoSource)
input = hVideoSource();
% Find location of Target in the input video frame
    if firstTime
      Idx = int32(pos.template_center_pos);
      MotionVector = [0 0];
      firstTime = false;
    else
      IdxPrev = Idx;
 ROI = [SearchRegion, pos.template_size+2*pos.search_border];
```

Video Stabilization

```
Idx = hTM(input,Target,ROI);
 MotionVector = double(Idx-IdxPrev);    end
[Offset, SearchRegion] = updatesearch(sz, MotionVector, ...
        SearchRegion, Offset, pos);
% Translate video frame to offset the camera motion
Stabilized = imtranslate(input, Offset, 'linear');
Target = Stabilized(TargetRowIndices, TargetColIndices);
 % Add black border for display
 Stabilized(:, BorderCols) = 0;
 Stabilized(BorderRows, :) = 0;
 TargetRect = [pos.template_orig-Offset, pos.template_size];
```

Video Stabilization

```
SearchRegionRect = [SearchRegion, pos.template_size +
2*pos.search_border];
% Draw rectangles on input to show target and search region
input = insertShape(input, 'Rectangle', [TargetRect;
SearchRegionRect],  'Color', 'white');
 % Display the offset (displacement) values on the input image
txt = sprintf('(%+05.1f,%+05.1f)', Offset);
input = insertText(input(:,:,1),[191 215],txt,'FontSize',16, ...
                    'TextColor', 'white', 'BoxOpacity', 0);
% Display video
hVideoOut([input(:,:,1) Stabilized]);
end
```

(+06.0,+06.0)

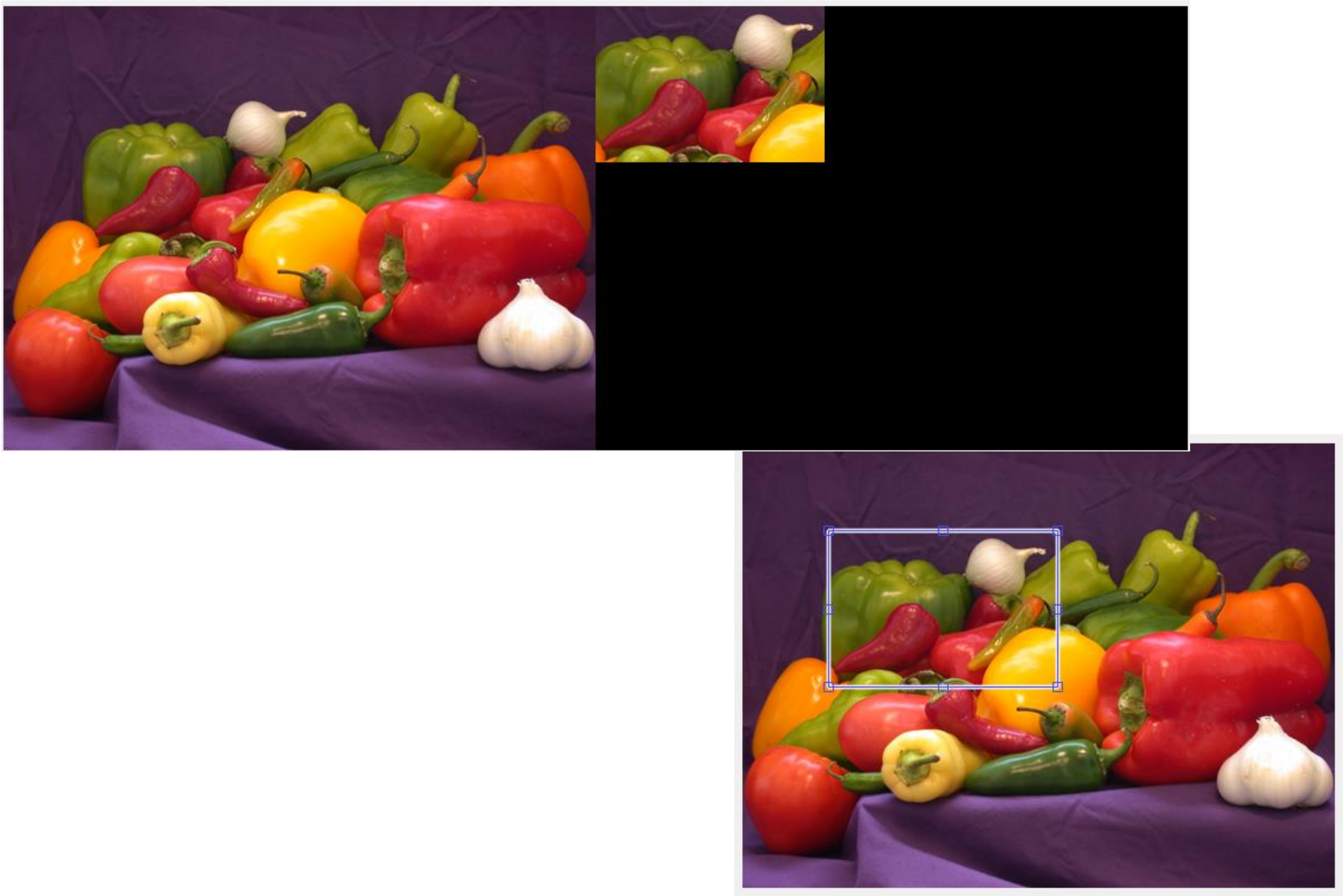
TEMPLATE MATCHING CORRELATION MATLAB

- C = normxcorr2(TEMPLATE,A) computes the normalized cross-correlation of matrices TEMPLATE and A. The matrix A must be larger than the matrix TEMPLATE for the normalization to be meaningful. The values of TEMPLATE cannot all be the same. The resulting matrix C contains correlation coefficients and its values may range from -1.0 to 1.0.

```
% Load images
close all;
% Load images
    onionrgb=imread('onion.png');
    onion   = rgb2gray(onionrgb);
    peppersrgb = imread('peppers.png');
    peppers = rgb2gray(peppersrgb);
```

TEMPLATE MATCHING CORRELATION MATLAB

```
imshowpair(peppersrgb,onionrgb,'montage')
c = normxcorr2(onion,peppers);
figure, surf(c), shading flat
[ypeak, xpeak] = find(c==max(c(:)));
% Compute translation from max location in correlation matrix
yoffSet = ypeak-size(onion,1);
 xoffSet = xpeak-size(onion,2);
 % Display matched area
 figure, hAx  = axes;
imshow(peppersrgb,'Parent', hAx);
imrect(hAx, [xoffSet+1, yoffSet+1, size(onion,2), size(onion,1)]);
```



HU MOMENT+ matchShapes

- Phương pháp template matching không hiệu quả với phép quay và tỷ lệ
- Tìm contour của object, sau đó tính moments, rồi tính Hu moment, bất biến với phép quay, tỷ lệ, tịnh tiến, phản chiếu.
- Dùng hàm matchShapes để so sánh hai đối tượng dùng Hu Moment

Moments

Calculates all of the moments up to the third order of a polygon or rasterized shape.

C++: Moments moments(InputArray array, bool binaryImage=false)

Python: cv2.moments(array[, binaryImage]) → retval

C: void cvMoments(const CvArr* arr, CvMoments* moments, int binary=0)

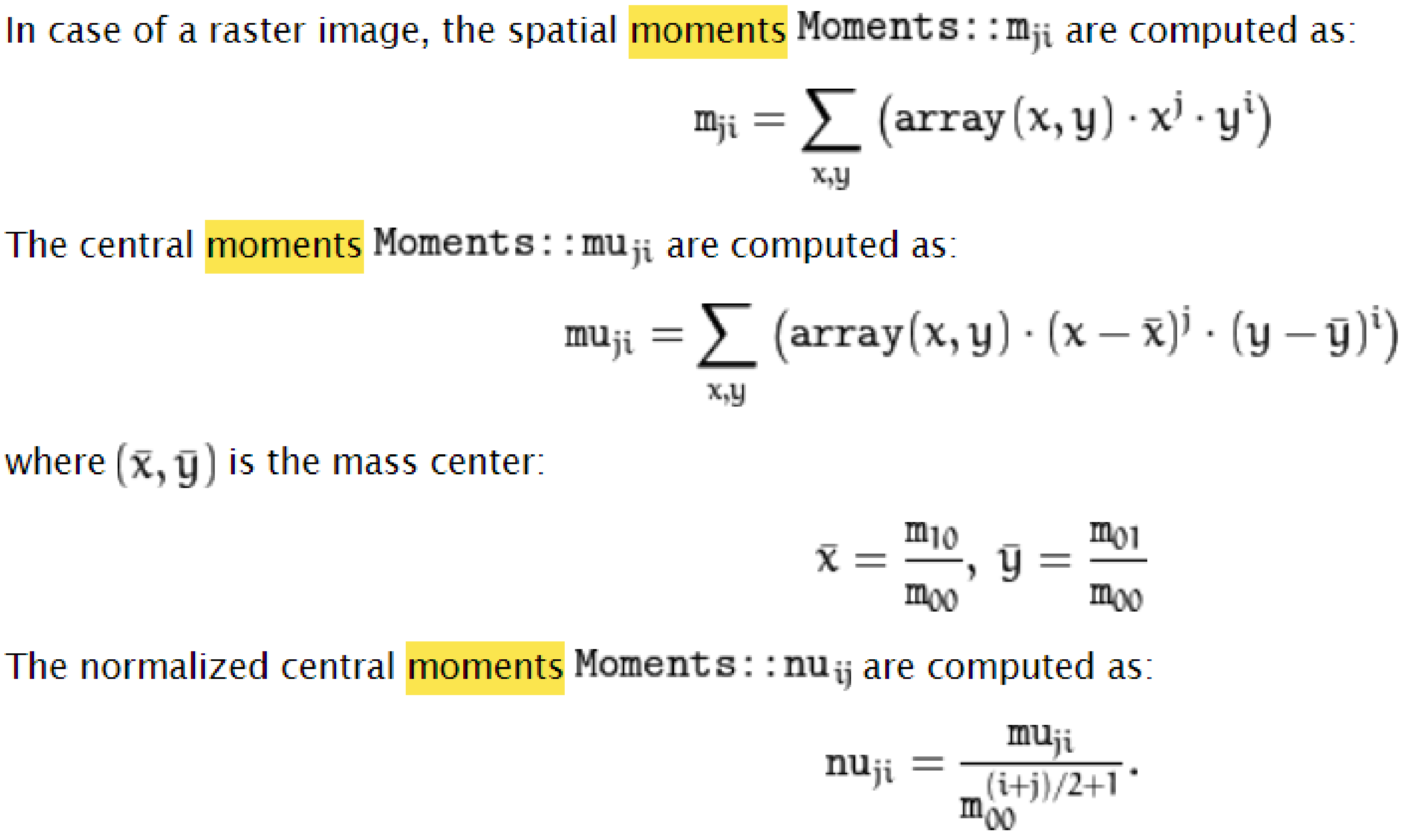
Parameters:

array – Raster image (single-channel, 8-bit or floating-point 2D array) or an array (1 \times N or N \times 1) of 2D points (Point or Point2f).

binaryImage – If it is true, all non-zero image pixels are treated as 1’s. The parameter is used for images only.

moments – Output moments.

Moments



In case of a raster image, the spatial moments $\texttt{Moments::m}_{ji}$ are computed as:

$$m_{ji} = \sum_{x,y} \left(\texttt{array}(x,y) \cdot x^j \cdot y^i\right)$$

The central moments $\texttt{Moments::mu}_{ji}$ are computed as:

$$mu_{ji} = \sum_{x,y} \left(\texttt{array}(x,y) \cdot (x - \bar{x})^j \cdot (y - \bar{y})^i\right)$$

where $(\bar{x}, \bar{y})$ is the mass center:

$$\bar{x} = \frac{m_{10}}{m_{00}}, \ \bar{y} = \frac{m_{01}}{m_{00}}$$

The normalized central moments $\texttt{Moments::nu}_{ij}$ are computed as:

$$nu_{ji} = \frac{mu_{ji}}{m_{00}^{(i+j)/2+1}}.$$

HuMoments

Calculates seven Hu invariants.

C++: void HuMoments(const Moments& m, OutputArray hu)

C++: void HuMoments(const Moments& moments, double hu[7])

Python: cv2.HuMoments(m[, hu]) → hu

C: void cvGetHuMoments(CvMoments* moments, CvHuMoments* hu_moments)

Parameters:

moments – Input moments computed with moments() .

hu – Output Hu invariants, calculated from normalized central moment

HuMoments

$$h_1 = v_{20} + v_{02}$$

$$h_2 = (v_{20} + v_{02})^2 + 4v_{11}^2$$

$$h_3 = (v_{30} - 3v_{12})^2 + (3v_{21} - v_{03})^2$$

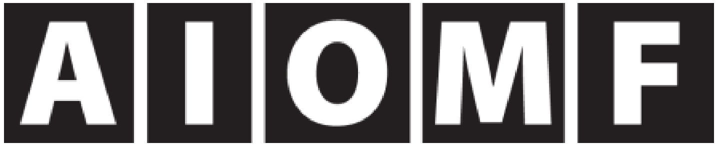
$$h_4 = (v_{30} + v_{12})^2 + (v_{21} + v_{03})^2$$

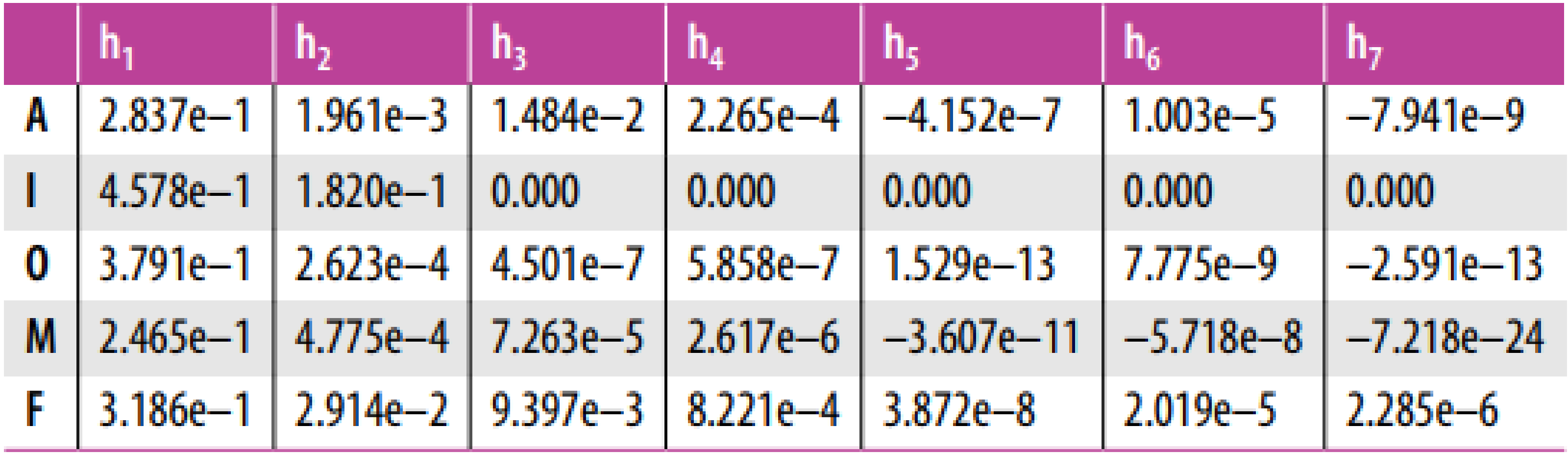
$$h_5 = (v_{30} - 3v_{12})(v_{30} + v_{12})\left[(v_{30} + v_{12})^2 - 3(v_{21} + v_{03})^2\right]$$
$$+ (3v_{21} - v_{03})(v_{21} + v_{03})[3(v_{30} + v_{12}) - (v_{21} + v_{03})]$$

$$h_6 = (v_{20} - v_{02})[(v_{30} + v_{12})^2 - (v_{21} + v_{03})^2] + 4v_{11}(v_{30} + v_{12})(v_{21} + v_{03})$$

$$h_7 = (3v_{21} - v_{03})(v_{30} + v_{12})\left[(v_{30} + v_{12})^2 - 3(v_{21} + v_{03})^2\right]$$
$$- (v_{30} - 3v_{12})(v_{21} + v_{03})\left[3(v_{30} + v_{12})^2 - (v_{21} + v_{03})^2\right]$$

HuMoments





	h_1	h_2	h_3	h_4	h_5	h_6	h_7
A	2.837e–1	1.961e–3	1.484e–2	2.265e–4	–4.152e–7	1.003e–5	–7.941e–9
I	4.578e–1	1.820e–1	0.000	0.000	0.000	0.000	0.000
O	3.791e–1	2.623e–4	4.501e–7	5.858e–7	1.529e–13	7.775e–9	–2.591e–13
M	2.465e–1	4.775e–4	7.263e–5	2.617e–6	–3.607e–11	–5.718e–8	–7.218e–24
F	3.186e–1	2.914e–2	9.397e–3	8.221e–4	3.872e–8	2.019e–5	2.285e–6

matchShapes

Compares two shapes.

C++: double matchShapes(InputArray contour1, InputArray contour2, int method, double parameter)

Python: cv2.matchShapes(contour1, contour2, method, parameter) → retval

C: double cvMatchShapes(const void* object1, const void* object2, int method, double parameter=0)

Parameters:

object1 – First contour or grayscale image.

object2 – Second contour or grayscale image.

method – Comparison method: CV_CONTOURS_MATCH_I1 , CV_CONTOURS_MATCH_I2 or CV_CONTOURS_MATCH_I3 parameter – Method-specific parameter (not supported now).

matchShapes

- method=CV_CONTOURS_MATCH_I1

$$I_1(A,B) = \sum_{i=1...7} \left| \frac{1}{m_i^A} - \frac{1}{m_i^B} \right|$$

- method=CV_CONTOURS_MATCH_I2

$$I_2(A,B) = \sum_{i=1...7} \left| m_i^A - m_i^B \right|$$

- method=CV_CONTOURS_MATCH_I3

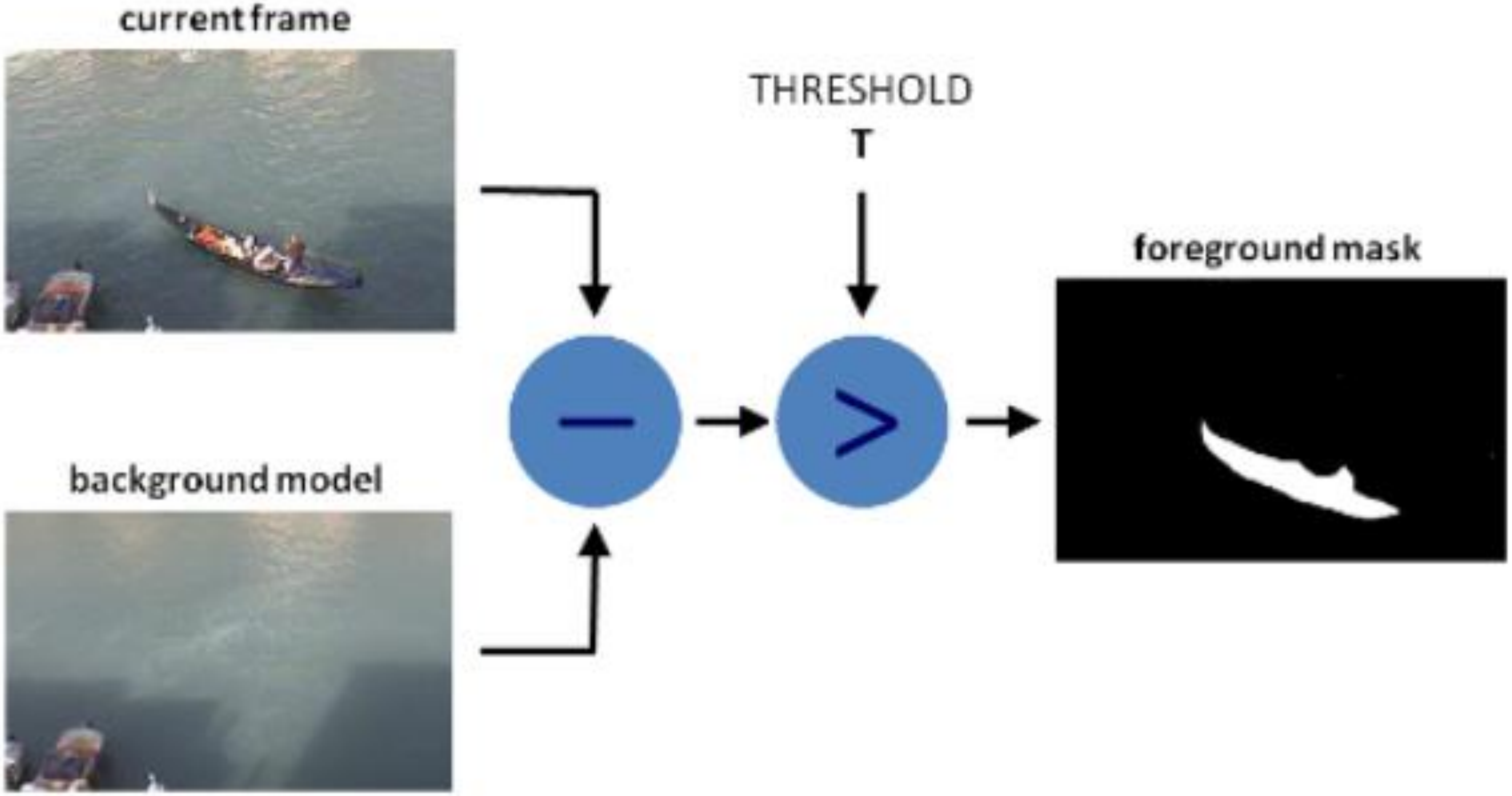
$$I_3(A,B) = \max_{i=1...7} \frac{\left| m_i^A - m_i^B \right|}{\left| m_i^A \right|}$$

where

$$\begin{aligned} m_i^A &= \mathrm{sign}(h_i^A) \cdot \log h_i^A \\ m_i^B &= \mathrm{sign}(h_i^B) \cdot \log h_i^B \end{aligned}$$

and h_i^A, h_i^B are the Hu moments of A and B, respectively.

Motion Detect
Foreground detector and blob analysis
Gaussian Mixture Models


current frame
THRESHOLD
T
foreground mask
−
>
background model

PYTHON MOTION DETECT

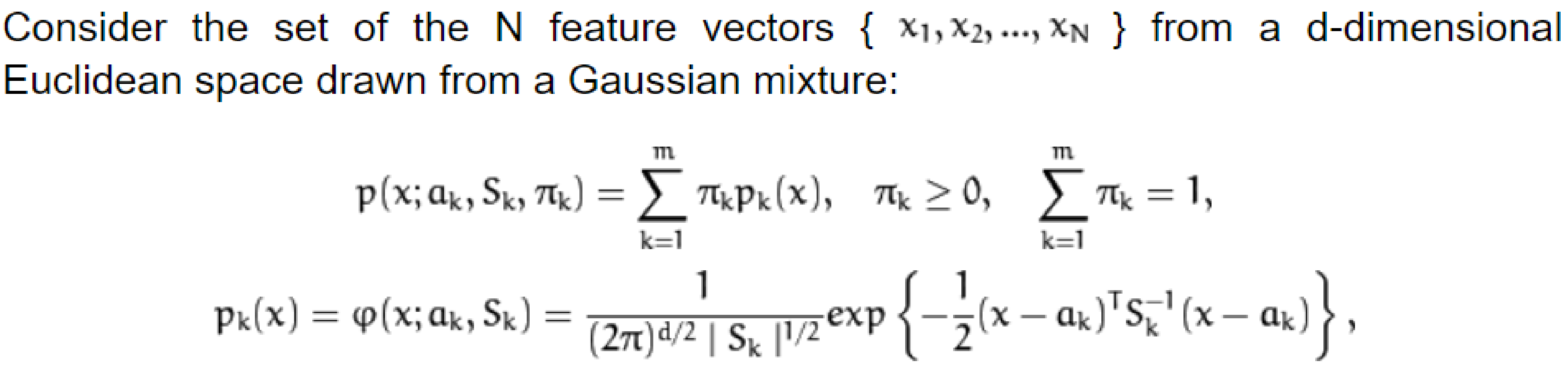
```
import cv2
import numpy as np
cap = cv2.VideoCapture(0)
ret, frame1 = cap.read()
ret, frame2 = cap.read()
while ret:
     d = cv2.absdiff(frame1, frame2)
    grey = cv2.cvtColor(d, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(grey, (5, 5), 0)
    ret, th = cv2.threshold( blur, 20, 255, cv2.THRESH_BINARY)
    dilated = cv2.dilate(th, np.ones((3, 3), np.uint8), iterations=1 )
```

PYTHON MOTION DETECT

```
 eroded = cv2.erode(dilated, np.ones((3, 3), np.uint8), iterations=1 )
    c, h = cv2.findContours(eroded, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
    cv2.drawContours(frame1, c, -1, (0, 0, 255), 2)
    cv2.imshow("Original", frame2)
    cv2.imshow("Output", frame1)
    if cv2.waitKey(1) == 27: # exit on ESC
        break
    frame1 = frame2
    ret, frame2 = cap.read()
cv2.destroyAllWindows()
cap.release()
```

Gaussian mixture models (GMM)

- Gaussian mixture models (GMM) are composed of k multivariate normal density components, where k is a positive integer. Each component has a d-dimensional mean (d is a positive integer), d-by-d covariance matrix, and a mixing proportion. Mixing proportion j determines the proportion of the population composed by component j, j = 1,...,k.



Consider the set of the N feature vectors { $x_1, x_2, \ldots, x_N$ } from a d-dimensional Euclidean space drawn from a Gaussian mixture:

$$p(x; a_k, S_k, \pi_k) = \sum_{k=1}^{m} \pi_k p_k(x), \quad \pi_k \geq 0, \quad \sum_{k=1}^{m} \pi_k = 1,$$

$$p_k(x) = \varphi(x; a_k, S_k) = \frac{1}{(2\pi)^{d/2} \mid S_k \mid^{1/2}} \exp\left\{ -\frac{1}{2}(x - a_k)^T S_k^{-1} (x - a_k) \right\},$$

where m is the number of mixtures, p_k is the normal distribution density with the mean a_k and covariance matrix S_k, \pi_k is the weight of the k-th mixture

https://docs.opencv.org/3.3.0/db/d5c/tutorial_py_bg_subtraction.html

Background subtraction is a major preprocessing steps in many vision based applications. For example, consider the cases like visitor counter where a static camera takes the number of visitors entering or leaving the room, or a traffic camera extracting information about the vehicles etc. In all these cases, first you need to extract the person or vehicles alone. Technically, you need to extract the moving foreground from static background.

If you have an image of background alone, like image of the room without visitors, image of the road without vehicles etc, it is an easy job. Just subtract the new image from the background. You get the foreground objects alone. But in most of the cases, you may not have such an image, so we need to extract the background from whatever images we have. It become more complicated when there is shadow of the vehicles. Since shadow is also moving, simple subtraction will mark that also as foreground. It complicates things.

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 colours stay in the scene. The probable background colours are the ones which stay longer and more static.

While coding, we need to create a background object using the function createBackgroundSubtractorMOG(). It has some optional parameters like length of history, number of gaussian mixtures, threshold etc. It is all set to some default values. Then inside the video loop, use backgroundsubtractor.apply() method to get the foreground mask.

PYTHON MOG

```
import numpy as np
import cv2
cap = cv2.VideoCapture('d:/vtest.avi')
fgbg = cv2.bgsegm.createBackgroundSubtractorMOG()
while(1):
    ret, frame = cap.read()
    cv2.imshow('frame1',frame)
    fgmask = fgbg.apply(frame)
    cv2.imshow('frame',fgmask)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break
cap.release()
cv2.destroyAllWindows()
```

BackgroundSubtractorMOG2

It is also a Gaussian Mixture-based Background/Foreground Segmentation Algorithm. It is based on two papers by Z.Zivkovic, "Improved adaptive Gausian 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 adaptibility to varying scenes due illumination changes etc.

As in previous case, we have to create a background subtractor object. Here, you have an option of selecting whether shadow to be detected or not. If detectShadows = True (which is so by default), it detects and marks shadows, but decreases the speed. Shadows will be marked in gray color.

PYTHON MOG2

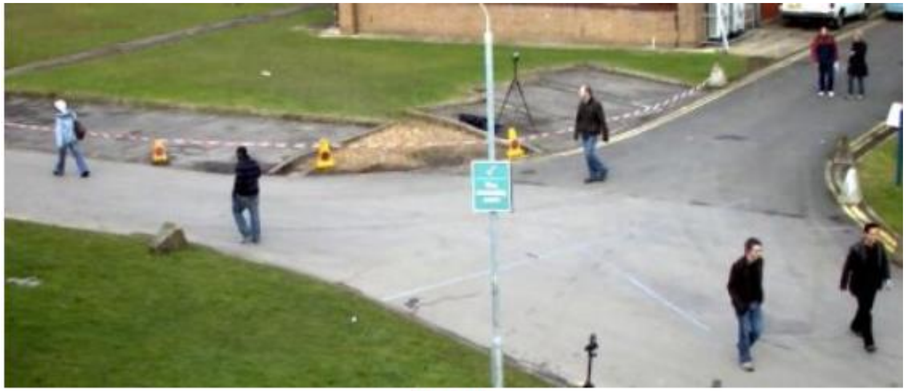
```
import numpy as np
import cv2
cap = cv2.VideoCapture('vtest.avi')
fgbg = cv2. bgsegm.createBackgroundSubtractorMOG2()
while(1):
    ret, frame = cap.read()
    fgmask = fgbg.apply(frame)
    cv2.imshow('frame',fgmask)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break
cap.release()
cv2.destroyAllWindows()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

BackgroundSubtractorGMG

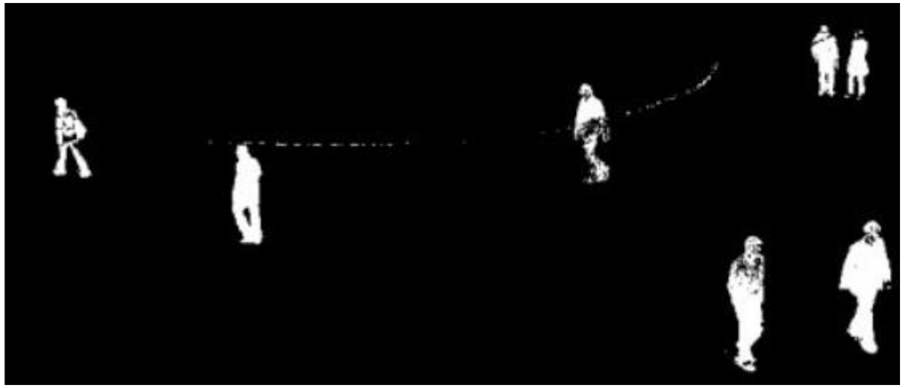
This algorithm combines statistical background image estimation and per-pixel Bayesian segmentation. It was introduced by Andrew B. Godbehere, Akihiro Matsukawa, Ken Goldberg in their paper "Visual Tracking of Human Visitors under Variable-Lighting Conditions for a Responsive Audio Art Installation" in 2012. It employs 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.

PYTHON GMG

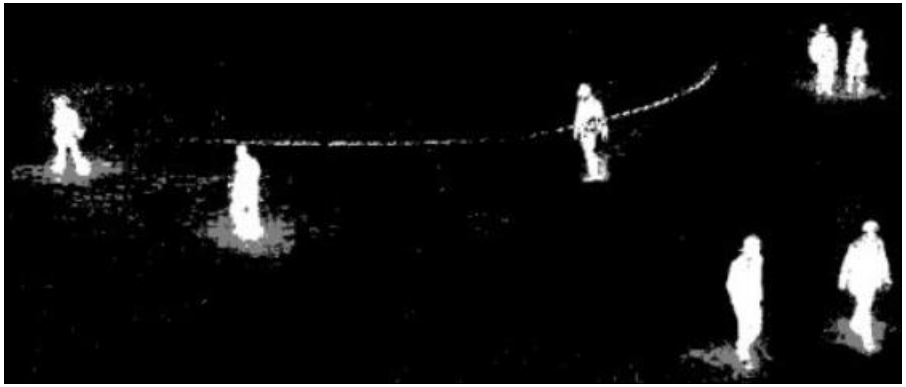
```
import numpy as np
import cv2
cap = cv2.VideoCapture('vtest.avi')
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(3,3))
fgbg = cv2. bgsegm.createBackgroundSubtractorGMG()
while(1):
    ret, frame = cap.read()
    fgmask = fgbg.apply(frame)
    fgmask = cv2.morphologyEx(fgmask, cv2.MORPH_OPEN,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



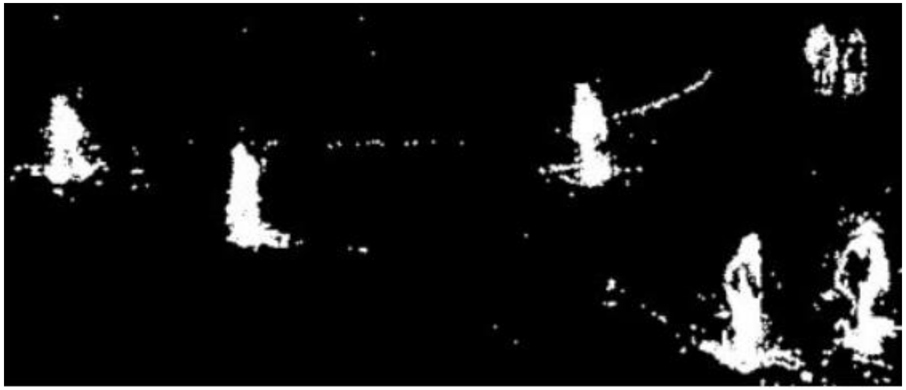
image



Result of BackgroundSubtractorMOG



Result of BackgroundSubtractorMOG2



Result of BackgroundSubtractorGMG

vision.ForegroundDetector System object

- The ForegroundDetector compares a color or grayscale video frame to a background model to determine whether individual pixels are part of the background or the foreground. It then computes a foreground mask. By using background subtraction, you can detect foreground objects in an image taken from a stationary camera.
- To detect foreground in an image :
  - Create the vision.ForegroundDetector object and set its properties.
  - Call the object with arguments, as if it were a function.
- detector = vision.ForegroundDetector computes and returns a foreground mask using the Gaussian mixture model (GMM).
- detector = vision.ForegroundDetector(Name,Value) sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, detector = vision.ForegroundDetector('LearningRate',0.005)

vision.ForegroundDetector System object

AdaptLearningRate — Adapt learning rate

'true' (default) | 'false'

NumTrainingFrames — Number of initial video frames for training background model

150 (default) | integer

LearningRate — Learning rate for parameter updates 0.005 (default) | numeric scalar

MinimumBackgroundRatio — Threshold to determine background model 0.7 (default) | numeric scalar

NumGaussians — Number of Gaussian modes in the mixture model 5 (default) | positive integer

InitialVariance — Initial mixture model variance

'Auto' (default) | numeric scalar

vision.ForegroundDetector System object

foregroundMask = detector(I) computes the foreground mask for input image I, and returns a logical mask. Values of 1 in the mask correspond to foreground pixels.

foregroundMask = detector(I,learningRate) computes the foreground mask using the LearningRate.

I — Input image, grayscale | truecolor (RGB)

learningRate — Learning rate for parameter updates

0.005 (default) | numeric scalar

foregroundMask — Foreground mask , binary mask

vision.BlobAnalysis System object

To compute statistics for connected regions in a binary image. To track a set of points:

Hblob = vision.BlobAnalysis returns a blob analysis object, H, used to compute statistics for connected regions in a binary image.

Hblob = vision.BlobAnalysis(Name,Value) sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, Hblob = vision.BlobAnalysis('AreaOutputPort',true)

AreaOutputPort — Return blob area true (default) | false

CentroidOutputPort — Return coordinates of blob centroids

true (default) | false

BoundingBoxOutputPort — Return coordinates of bounding boxes true (default) | false

MajorAxisLengthOutputPort — Return vector whose values represent lengths of ellipses' major axes false (default) | true

vision.BlobAnalysis System object

MinorAxisLengthOutputPort — Return vector whose values represent lengths of ellipses' minor axes false (default) | true

OrientationOutputPort — Return vector whose values represent angles between ellipses' major axes and x-axis false (default) | true

EccentricityOutputPort — Return vector whose values represent ellipses' eccentricities false (default) | true

EquivalentDiameterSquaredOutputPort — Return vector whose values represent equivalent diameters squared false (default) | true

ExtentOutputPort — Return vector whose values represent results of dividing blob areas by bounding box areas false (default) | true

PerimeterOutputPort — Return vector whose values represent estimates of blob perimeter lengths false (default) | true

OutputDataType — Output data type of statistics

double (default) | single | Fixed point

vision.BlobAnalysis System object

Connectivity — Connected pixels 8 (default) | 4

LabelMatrixOutputPort — Maximum number of labeled regions in each input image 50 (default) | positive scalar integer.

MinimumBlobArea — Minimum blob area in pixels

0 (default) | positive scalar integer.

MaximumBlobArea — Maximum blob area in pixels

intmax('uint32') (default) | integer

ExcludeBorderBlobs — Exclude blobs that contain at least one image border pixel false (default) | true

MaximumCount — Maximum number of labeled regions in each input image 50 (default) | positive scalar integer

vision.BlobAnalysis System object

[area,centrioid,bbox] = Hblob(bw)returns the area, centroid, and the bounding box of the blobs when the AreaOutputPort, CentroidOutputPort and BoundingBoxOutputPort properties are set to true. These are the only properties that are set to true by default. If you set any additional properties to true, the corresponding outputs follow the area,centrioid, and bbox outputs.

[___,majoraxis] = Hblob(bw) computes the major axis length majoraxis of the blobs found in input binary image bw when you set the MajorAxisLengthOutputPort property to true.

vision.BlobAnalysis System object

[___,minoraxis] = Hblob(bw) computes the minor axis length minoraxis of the blobs found in input binary image BW when you set the MinorAxisLengthOutputPort property to true.

[bw___,orientation] = Hblob(bw) computes the orientation of the blobs found in input binary image bw when you set the OrientationOutputPort property to true.

[___,eccentricity] = Hblob(bw) computes the eccentricity of the blobs found in input binary image bw when you set the EccentricityOutputPort property to true.

[___,EQDIASQ] = Hblob(bw) computes the equivalent diameter squared EQDIASQ of the blobs found in input binary image bw when you set the EquivalentDiameterSquaredOutputPort property to true.

vision.BlobAnalysis System object

[___,EXTENT] = Hblob(bw) computes the EXTENT of the blobs found in input binary image bw when the ExtentOutputPort property is set to true.

[___,perimeter] = Hblob(bw) computes the perimeter of the blobs found in input binary image bw when you set the PerimeterOutputPort property to true.

[___,label] = Hblob(bw) returns a label matrix label of the blobs found in input binary image bw when you set the LabelMatrixOutputPort property to true.

CAR COUNTING

- Detecting and counting cars can be used to analyze traffic patterns. Detection is also a first step prior to performing more sophisticated tasks such as tracking or categorization of vehicles by their type.
- This example shows how to use the foreground detector and blob analysis to detect and count cars in a video sequence. It assumes that the camera is stationary. The example focuses on detecting objects.
- This example shows how to detect and count cars in a video sequence using foreground detector based on Gaussian mixture models (GMMs).
-

CAR COUNTING

%Rather than immediately processing the entire video, the example starts by obtaining an initial video frame in which the moving objects are segmented from the background. This helps to gradually introduce the steps used to process the video.

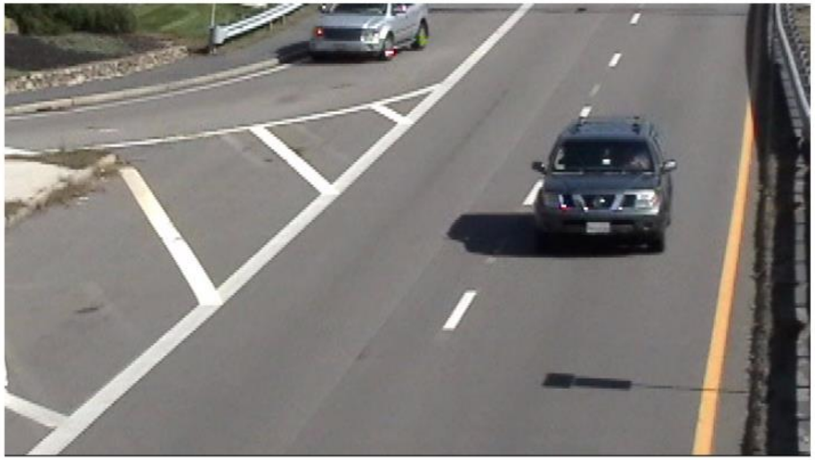
%The foreground detector requires a certain number of video frames in order to initialize the Gaussian mixture model. This example uses the first 50 frames to initialize three Gaussian modes in the mixture model.

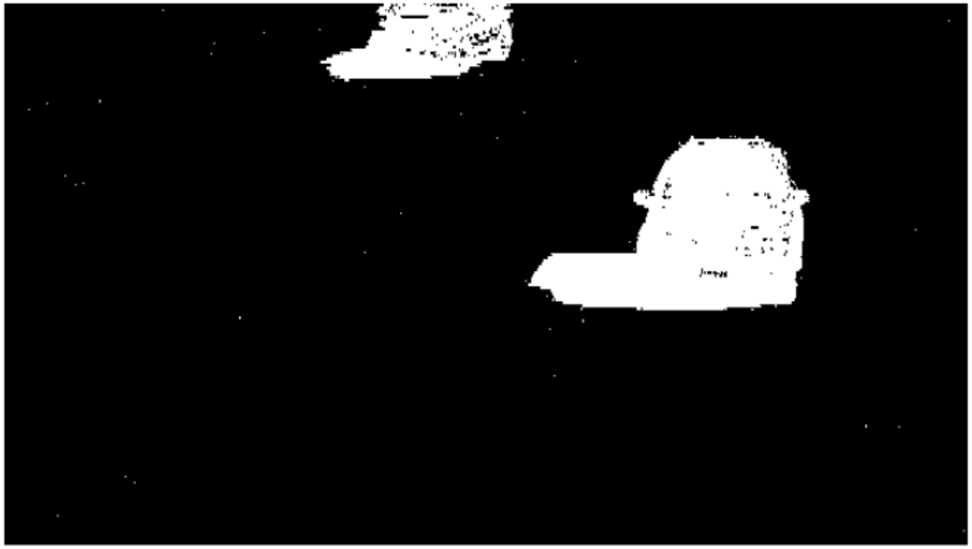
```
foregroundDetector =
vision.ForegroundDetector('NumGaussians', 3, ...
    'NumTrainingFrames', 50);
```

CAR COUNTING

```
videoReader = vision.VideoFileReader('visiontraffic.avi');
for i = 1:150
    frame = step(videoReader); % read the next video frame
    foreground = step(foregroundDetector, frame);
End
%After the training, the detector begins to output more reliable segmentation results. The two figures below show one of the video frames and the foreground mask computed by the detector.
figure; imshow(frame); title('Video Frame');
figure; imshow(foreground); title('Foreground');
```

CAR COUNTING


Video Frame


Foreground

CAR COUNTING

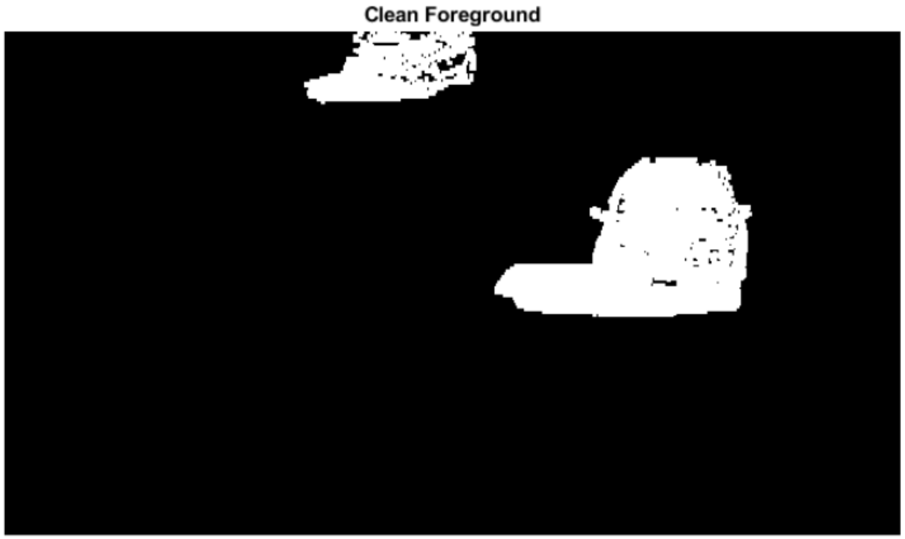
%Step 2 - Detect Cars in an Initial Video Frame

%The foreground segmentation process is not perfect and often includes undesirable noise. The example uses morphological opening to remove the noise and to fill gaps in the detected objects.

```
se = strel('square', 3);
filteredForeground = imopen(foreground, se);
figure; imshow(filteredForeground); title('Clean Foreground');
```

%Next, we find bounding boxes of each connected component corresponding to a moving car by using vision.BlobAnalysis object. The object further filters the detected foreground by rejecting blobs which contain fewer than 150 pixels.

CAR COUNTING

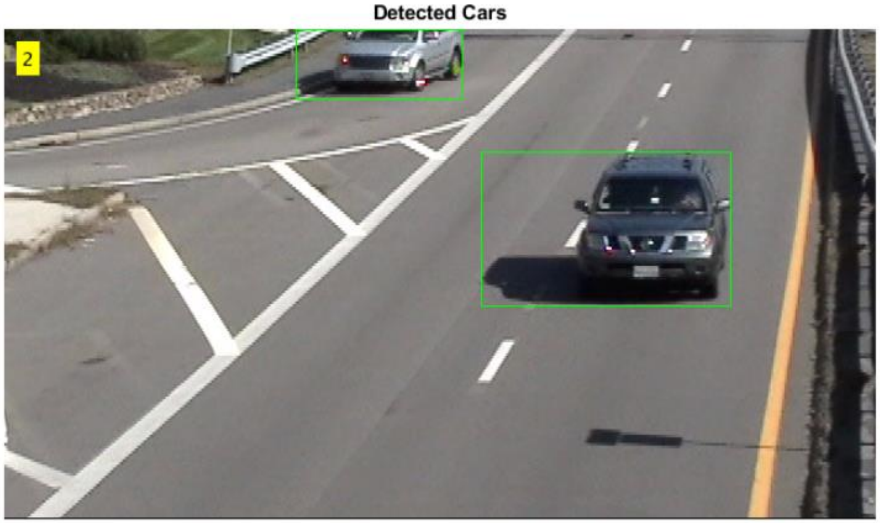


```
blobAnalysis = vision.BlobAnalysis('BoundingBoxOutputPort',
true,  'AreaOutputPort', false, 'CentroidOutputPort', false, ...
    'MinimumBlobArea', 150);
bbox = step(blobAnalysis, filteredForeground);
result = insertShape(frame, 'Rectangle', bbox, 'Color', 'green');
```

CAR COUNTING

%The number of bounding boxes corresponds to the number of cars found in the video frame. We display the number of found cars in the upper left corner of the processed video frame.

```
numCars = size(bbox, 1);
result = insertText(result, [10 10], numCars, 'BoxOpacity', 1, ...
    'FontSize', 14);
figure; imshow(result); title('Detected Cars');
```



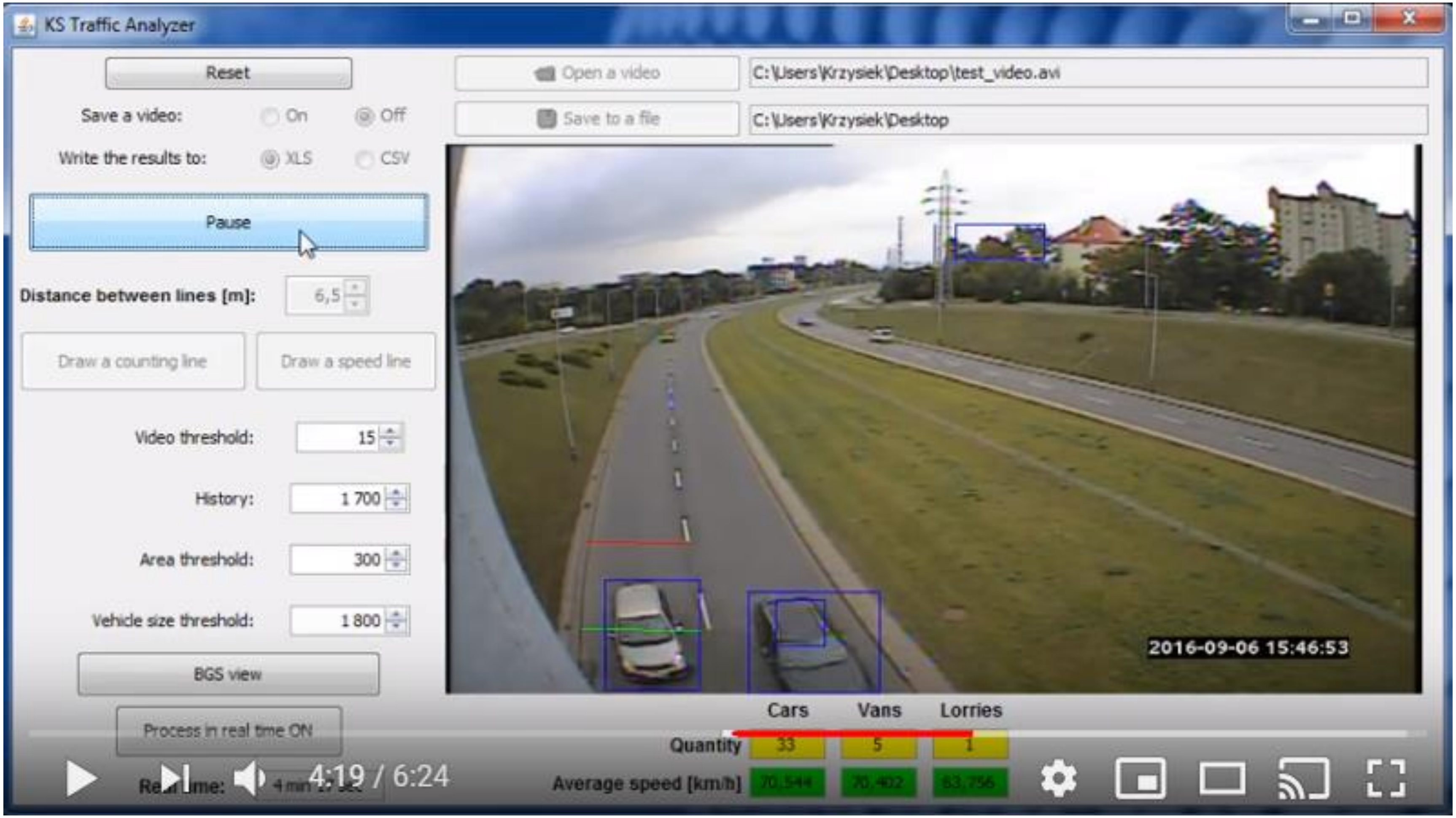
CAR COUNTING

```
%Step 3 - Process the Rest of Video Frames
%In the final step, we process the remaining video frames.
videoPlayer = vision.VideoPlayer('Name', 'Detected Cars');
videoPlayer.Position(3:4) = [650,400];  % window size: [width, height]
se = strel('square', 3); % morphological filter for noise removal
while ~isDone(videoReader)
    frame = step(videoReader); % read the next video frame
    % Detect the foreground in the current video frame
    foreground = step(foregroundDetector, frame);
    % Use morphological opening to remove noise in the foreground
```

CAR COUNTING

```
 filteredForeground = imopen(foreground, se);
% Detect the connected components with the specified minimum area, and compute their bounding boxes
bbox = step(blobAnalysis, filteredForeground);
% Draw bounding boxes around the detected cars
result = insertShape(frame, 'Rectangle', bbox, 'Color', 'green');
% Display the number of cars found in the video frame
numCars = size(bbox, 1);
result = insertText(result, [10 10], numCars, 'BoxOpacity', 1, ...
        'FontSize', 14);
step(videoPlayer, result);  % display the results
end
release(videoReader); % close the video file
```

CAR COUNTING



C:\Users\Krzysiek\Desktop\test_video.avi

C:\Users\Krzysiek\Desktop

Save a video: ○ On ● Off

Write the results to: ● XLS ○ CSV

Distance between lines [m]: 6,5

Video threshold: 15

History: 1 700

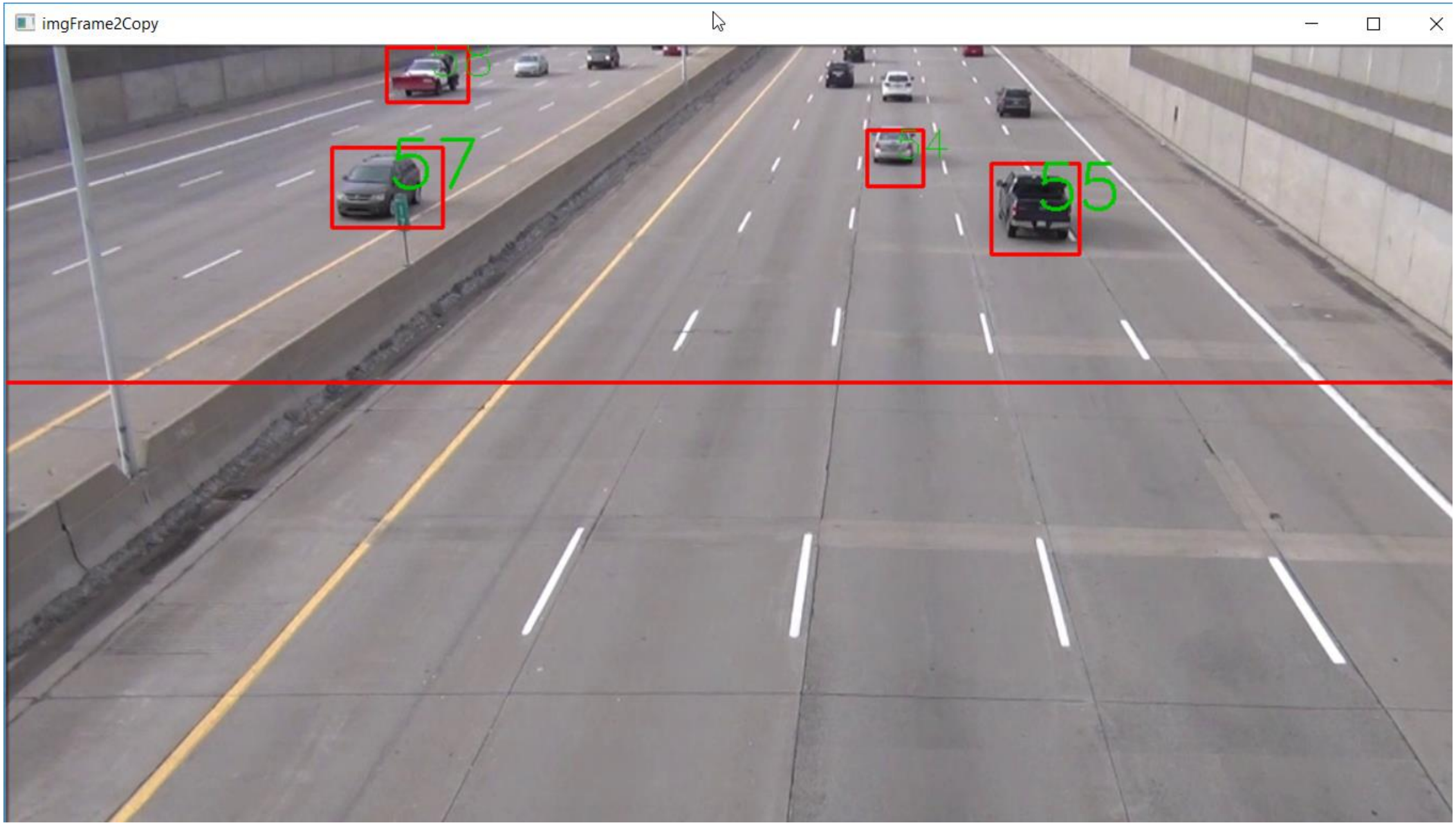
Area threshold: 300

Vehicle size threshold: 1 800

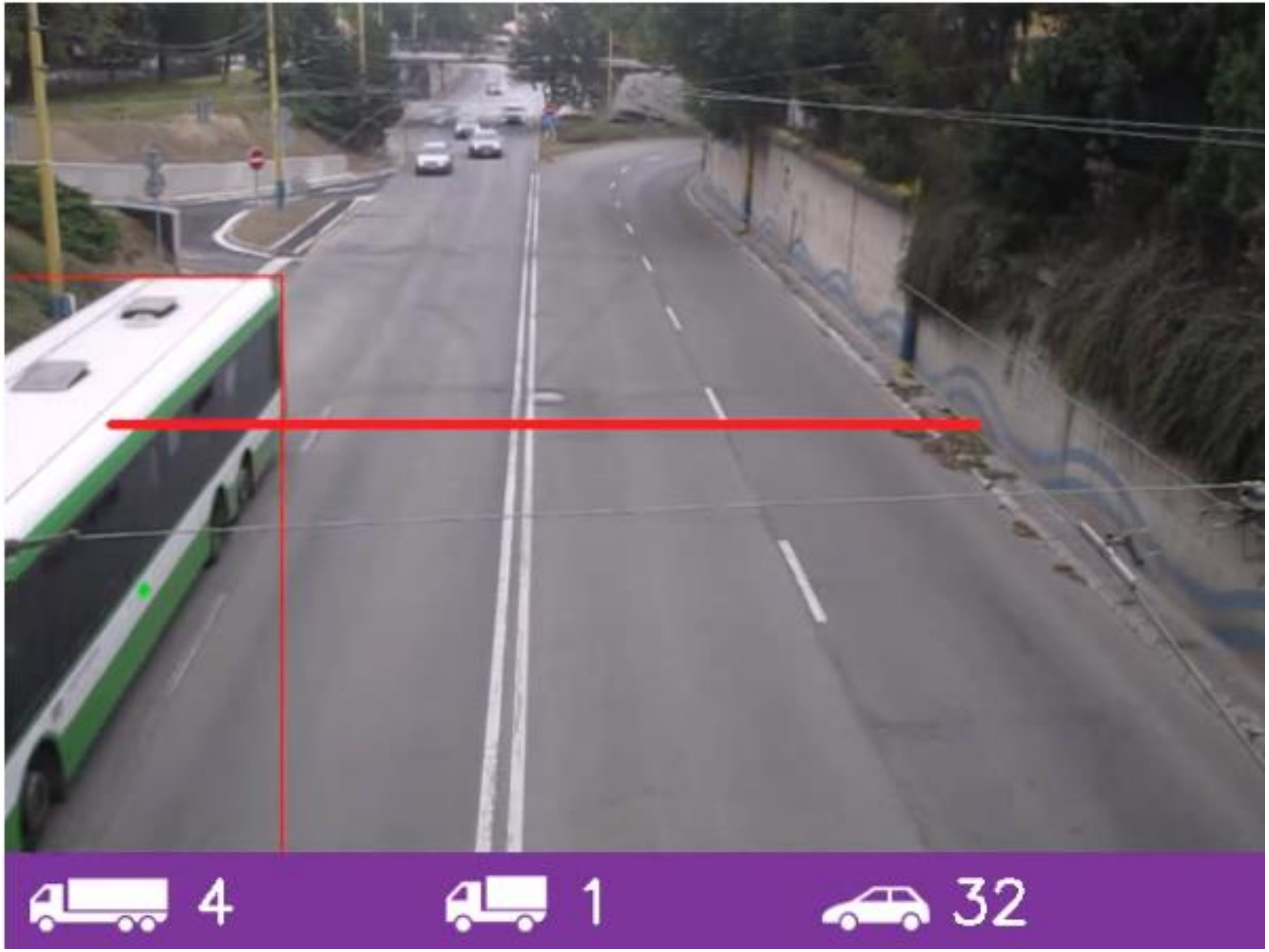
Real time: 4 min 17 sec

	Cars	Vans	Lorries
Quantity	33	5	1
Average speed [km/h]	70,544	70,402	83,756

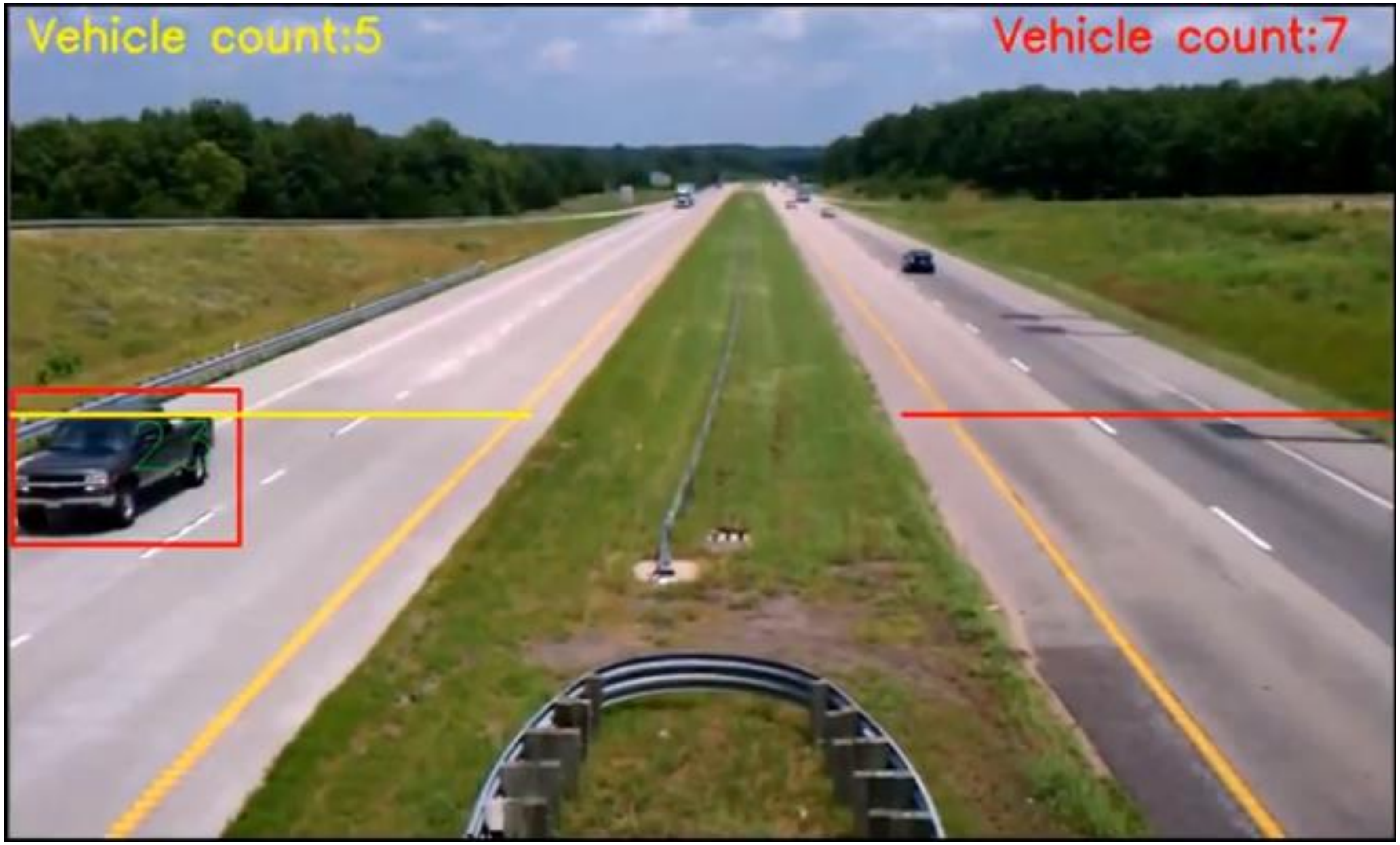
CAR COUNTING https://github.com/MicrocontrollersAndMore/OpenCV_3_Car_Counting_Cpp



CAR COUNTING https://www.youtube.com/watch?v=z1Cvn3_4yGo

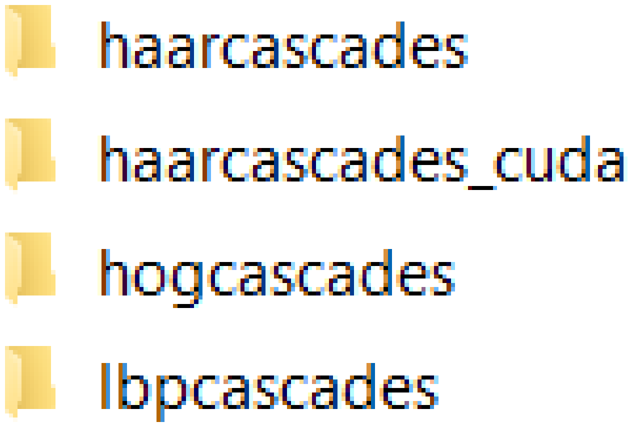


CAR COUNTING https://github.com/ahmetozlu/vehicle_counting



FACE DETECT
CASCADE CLASSIFIER

- OpenCV chứa các bộ phân lớp đã huấn luyện trong sources/data



haarcascades

haarcascades_cuda

hogcascades

lbpcascades

- Trong thư mục haarcascades cóc các bộ phân lớp mặt nhìn thẳng nhìn nghiêng, mắt, toàn thân người, nụ cười, biển số xe, các file này có đuôi xml (*eXtensible Markup Language)*
- Bộ phân lớp tốt nhất cho mặt nhìn thẳng là *haarcascade_frontalface_alt2.xml*.
  Thư mục lbpcascades *local binary patterns* chứa bộ phân lớp cải tiến tốt hơn haarcascade

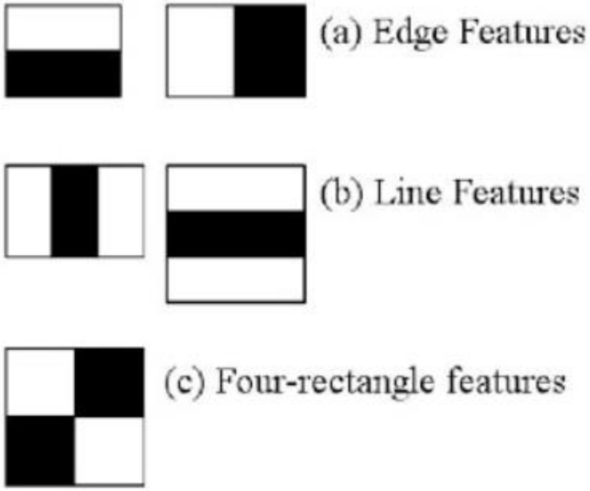
HAAR CASCADE

Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

Here we will work with face detection. Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then we need to extract features from it. For this, haar features shown in below image are used. They are just like our convolutional kernel. Each feature is a single value obtained by subtracting sum of pixels under white rectangle from sum of pixels under black rectangle.

HAAR CASCADE

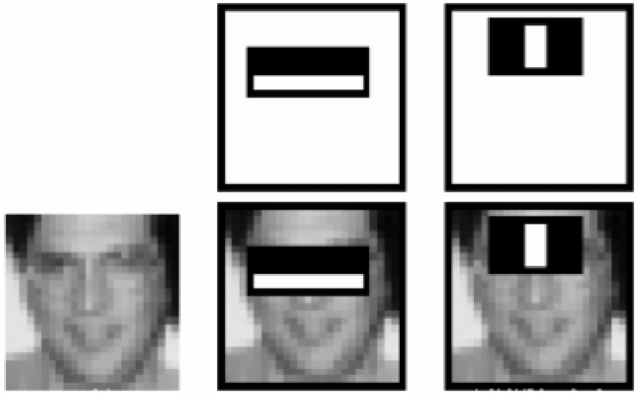
Now all possible sizes and locations of each kernel is used to calculate plenty of features. (Just imagine how much computation it needs? Even a 24x24 window results over 160.000 features). For each feature calculation, we need to find sum of pixels under white and black rectangles. To solve this, they introduced the integral images.



It simplifies calculation of sum of pixels, how large may be the number of pixels, to an operation involving just four pixels. But among all these features we calculated, most of them are irrelevant. For example, consider the image below. Top row shows two good features.

HAAR CASCADE

The first feature selected seems to focus on the property that the region of the eyes is often darker than the region of the nose and cheeks. The second feature selected relies on the property that the eyes are darker than the bridge of the nose. But the same windows applying on cheeks or any other place is irrelevant. So how do we select the best features out of 160000+ features? It is achieved by Adaboost.



HAAR CASCADE

For this, we apply each and every feature on all the training images. For each feature, it finds the best threshold which will classify the faces to positive and negative. But obviously, there will be errors or misclassifications. We select the features with minimum error rate, which means they are the features that best classifies the face and non-face images. (The process is not as simple as this. Each image is given an equal weight in the beginning. After each classification, weights of misclassified images are increased. Then again same process is done. New error rates are calculated. Also new weights. The process is continued until required accuracy or error rate is achieved or required number of features are found).

Final classifier is a weighted sum of these weak classifiers. It is called weak because it alone can't classify the image, but together

HAAR CASCADE

with others forms a strong classifier. The paper says even 200 features provide detection with 95% accuracy. Their final setup had around 6000 features. (Imagine a reduction from 160000+ features to 6000 features. That is a big gain).

So now you take an image. Take each 24x24 window. Apply 6000 features to it. Check if it is face or not. Wow.. Wow.. Isn't it a little inefficient and time consuming? Yes, it is. Authors have a good solution for that.

In an image, most of the image region is non-face region. So it is a better idea to have a simple method to check if a window is not a face region. If it is not, discard it in a single shot. Don't process it again. Instead focus on region where there can be a face. This way, we can find more time to check a possible face region.

HAAR CASCADE

For this they introduced the concept of Cascade of Classifiers. Instead of applying all the 6000 features on a window, group the features into different stages of classifiers and apply one-by-one. (Normally first few stages will contain very less number of features). If a window fails the first stage, discard it. We don't consider remaining features on it. If it passes, apply the second stage of features and continue the process. The window which passes all stages is a face region. How is the plan !!!

Authors' detector had 6000+ features with 38 stages with 1, 10, 25, 25 and 50 features in first five stages. (Two features in the above image is actually obtained as the best two features from Adaboost). According to authors, on an average, 10 features out of 6000+ are evaluated per sub-window.

FACE DETECT

- Các bộ phân lớp đọc vào chương trình qua hai lệnh
  - CascadeClassifier tên_ biến;
  - tên_ biến.load(tên file xml)
- Thực hiện phát giác đối tượng bằng lệnh

detectMultiScale(const Mat& image, vector<Rect>& objects, double scaleFactor=1.1, int minNeighbors=3, int flags=0, Size minSize=Size(), Size maxSize=Size())

Image: ảnh 8bit cần phát giác đối tượng

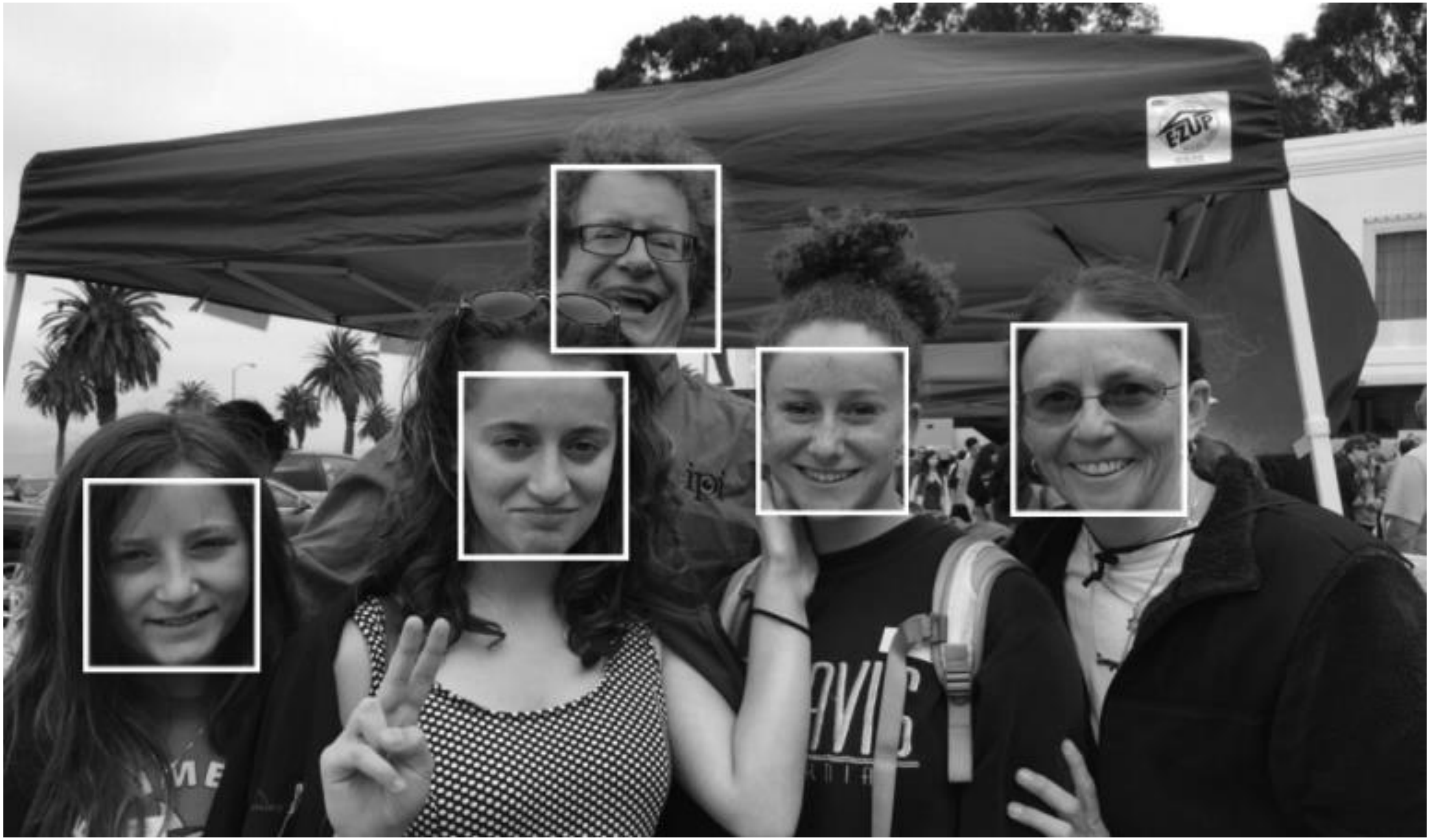
objects : hình chữ nhật trả về chứa đối tượng đã phát giác,

scaleFactor: tỷ lệ zoom ảnh=1,1

minNeighbors: đối tượng được phát giác nếu có số tối thiểu vùng cận cũng đã phát giác =3, flag=0 có thể bỏ không dùng

minSize, maxSize: kích thước vùng ảnh

FACE DETECT



Face Detect C++

```
#include "stdafx.h"
#include "opencv2/opencv.hpp"
#include <iostream>
#include <stdio.h>
using namespace std;
using namespace cv;
int main()
{Mat image;
image = imread("c:/lenna.jpg", CV_LOAD_IMAGE_COLOR);
namedWindow("window1", 0);
imshow("window1", image);
```

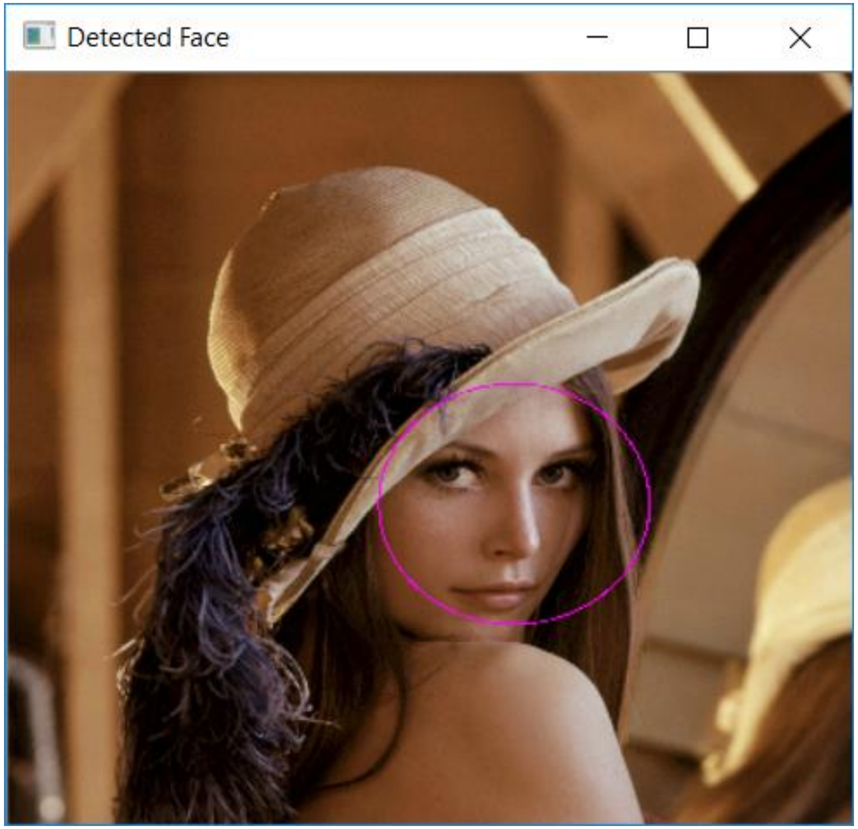
Face Detect C++

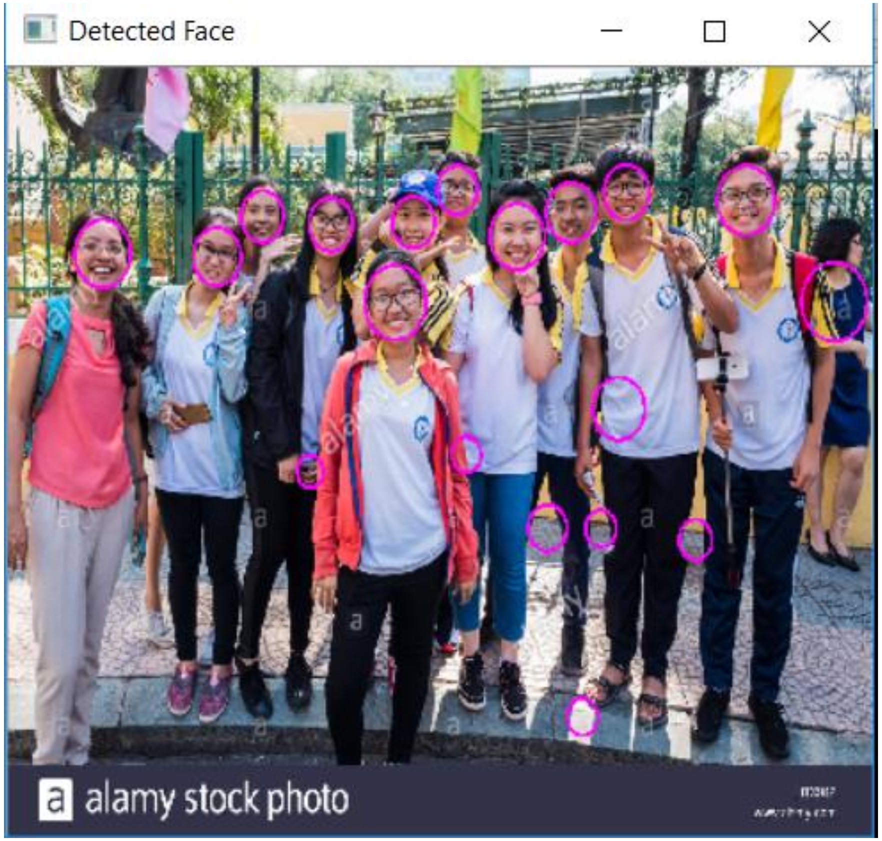
```
// Load Face cascade (.xml file)
CascadeClassifier face_cascade;
face_cascade.load("C:/opencv/sources/data/haarcascades/haarcascade_frontalface_alt2.xml");
// Detect faces
std::vector<Rect> faces;
face_cascade.detectMultiScale(image, faces, 1.1, 2, 0 | CV_HAAR_SCALE_IMAGE, Size(30, 30));
//flag 0| CV_HAAR_SCALE_IMAGE can be removed
// Draw circles on the detected faces
for (int i = 0; i < faces.size(); i++)
```

Face Detect C++

```
{
Point center(faces[i].x + faces[i].width*0.5, faces[i].y +
faces[i].height*0.5);
ellipse(image, center, Size(faces[i].width*0.5,
faces[i].height*0.5), 0, 0, 360, Scalar(255, 0, 255), 4, 8, 0);
}
namedWindow("Detected Face", 0);
imshow("Detected Face", image);
waitKey(0);
return 0;
}
```

Face Detect C++





FACE EYE DETECT CAMERA

```
#include "stdafx.h"
#include "opencv2/opencv.hpp"
#include <iostream>
#include <stdio.h>
using namespace std;
using namespace cv;
void detectAndDisplay(Mat frame);
String face_cascade_name = "c:/opencv/sources/data/haarcascades/haarcascade_frontalface_alt.xml";
```

FACE EYE DETECT CAMERA

```
String eyes_cascade_name =
"c:/opencv/sources/data/haarcascades/haarcascade_eye_tree_eyeglasses.xml";
CascadeClassifier face_cascade;
CascadeClassifier eyes_cascade;
/** @function main */
int main(int argc, const char** argv)
{
	VideoCapture cap;
	Mat frame;
	namedWindow("Detected", 0);
	//-- 1. Load the cascades
```

FACE EYE DETECT CAMERA

```
if (!face_cascade.load(face_cascade_name)) { printf("--(!)Error
loading\n"); return -1; };
if (!eyes_cascade.load(eyes_cascade_name)) { printf("--(!)Error
loading\n"); return -1; };
//-- 2. Read the video stream
cap.open(0);
if (cap.isOpened())
{
        while (true)
        {cap >> frame;
//-- 3. Apply the classifier to the frame
        if (!frame.empty())
```

FACE EYE DETECT CAMERA

```
{
        detectAndDisplay(frame);    }
        else
        {printf(" --(!) No captured frame -- Break!"); break;
                        }
int c = waitKey(10);
if ((char)c == 'c') { break; }
                }
        }
        return 0;
}
```

FACE EYE DETECT CAMERA

```
/*function detectAndDisplay */
void detectAndDisplay(Mat frame)
{
        std::vector<Rect> faces;
        Mat frame_gray;
        cvtColor(frame, frame_gray, CV_BGR2GRAY);
        equalizeHist(frame_gray, frame_gray);
        //-- Detect faces
        face_cascade.detectMultiScale(frame_gray, faces, 1.1, 2,
0 | CV_HAAR_SCALE_IMAGE, Size(30, 30));
        for (size_t i = 0; i < faces.size(); i++)
        {
```

FACE EYE DETECT CAMERA

```
Point center(faces[i].x + faces[i].width*0.5, faces[i].y +
faces[i].height*0.5);
                ellipse(frame, center, Size(faces[i].width*0.5,
faces[i].height*0.5), 0, 0, 360, Scalar(255, 0, 255), 4, 8, 0);
                Mat faceROI = frame_gray(faces[i]);
                std::vector<Rect> eyes;
                //-- In each face, detect eyes
        eyes_cascade.detectMultiScale(faceROI, eyes, 1.1, 2, 0 |
CV_HAAR_SCALE_IMAGE, Size(30, 30));
                for (size_t j = 0; j < eyes.size(); j++)
                {
```

FACE EYE DETECT CAMERA

```
Point center(faces[i].x + eyes[j].x + eyes[j].width*0.5, faces[i].y
+ eyes[j].y + eyes[j].height*0.5);
int radius = cvRound((eyes[j].width + eyes[j].height)*0.25);
circle(frame, center, radius, Scalar(255, 0, 0), 4, 8, 0);
}
}
//-- Show what you got
imshow("Detected", frame);
}
```

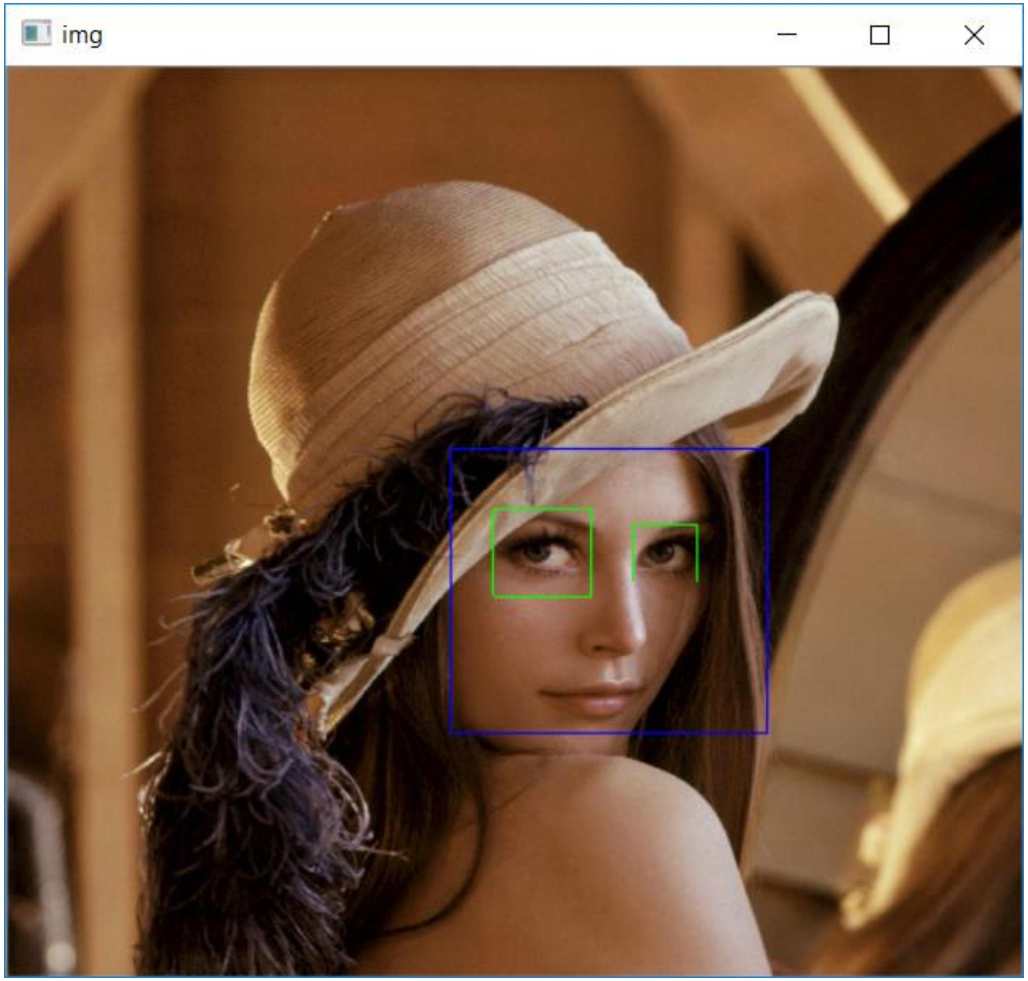
FACE DETECT PYTHON

```
import numpy as np
import cv2
face_cascade = cv2.CascadeClassifier('c:/opencv/sources/data/haarcascades/haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('c:/opencv/sources/data/haarcascades/haarcascade_eye_tree_eyeglasses.xml')
img = cv2.imread('c:/lenna.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
for (x,y,w,h) in faces:
    img = cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
```

FACE DETECT PYTHON

```
roi_gray = gray[y:y+h, x:x+w]
    roi_color = img[y:y+h, x:x+w]
    eyes = eye_cascade.detectMultiScale(roi_gray)
    for (ex,ey,ew,eh) in eyes:
        cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)
cv2.namedWindow('img',0)
cv2.imshow('img',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

FACE DETECT PYTHON



TRAIN CASCADE

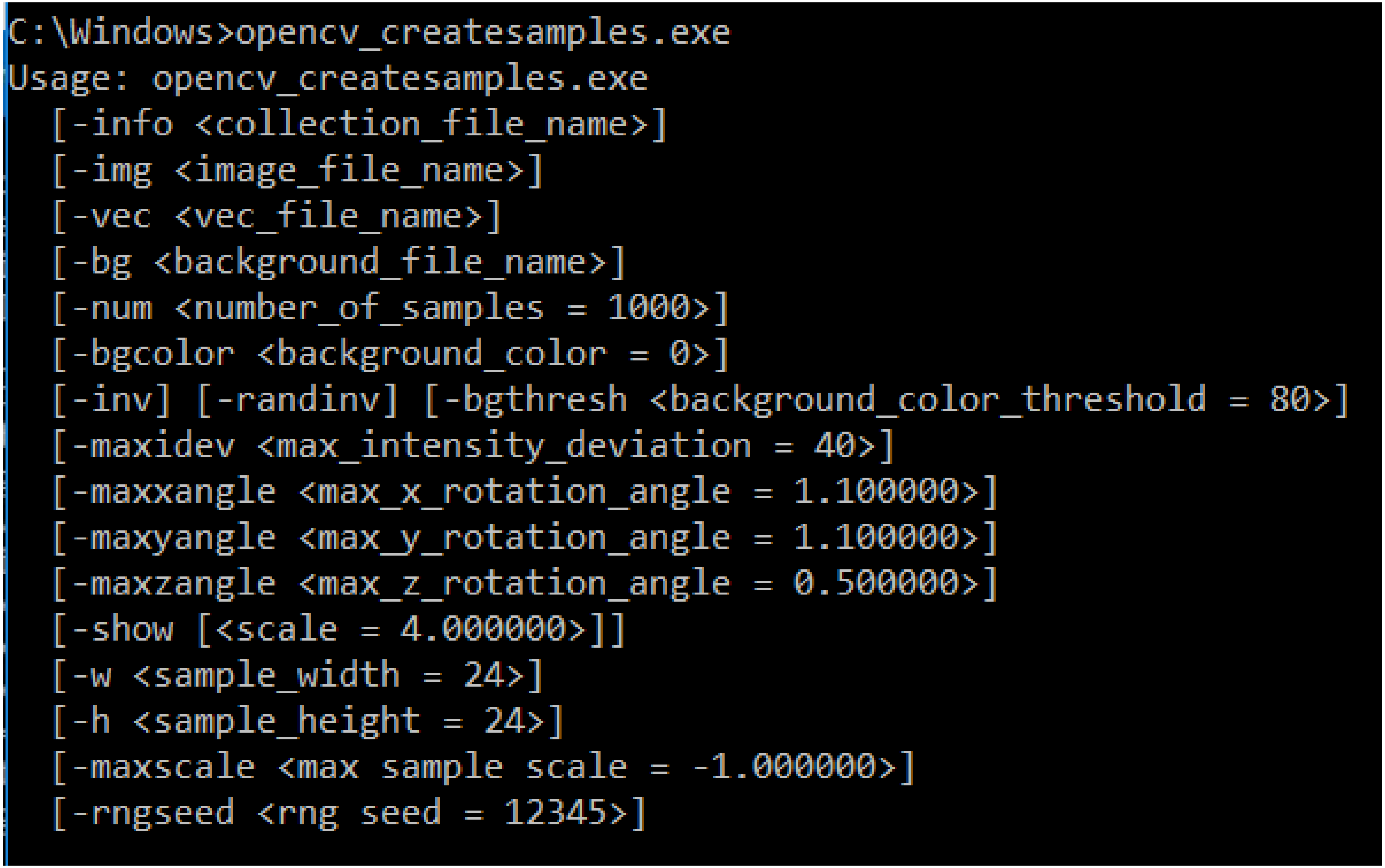
- Step1:
  Create a Folder ImageSample in drive for example C, in it Create two folders: Positive and Negative to contain positive images and negative images
- **Set of Positive Images:** Images which need to be detected or in other words the actual objects. E.g. Face for Face detection, eyes for Eye detection, Pen for Pen Detection etc.
  For Training Classifier for face detection we need to collect a vast database of faces, where face belong to almost each age group, males and females, with and without mustaches and beards, with varied skin colour etc.
- When we need to train classifier for one unique object, only one image can be enough. For e.g Image of a Company Logo, a Particular sign board etc.

- **Set of Negative Images:** Images other than the positive images or in other words the one without object in it. It should include all the backgrounds where you want your object to get detected.
- Open Command windows, move to the folder positive and type dir /b> positive.txt or dir /b> negative.txt These files contain names of image file.
- Open that positive.txt / negative.txt file. You will also find the name of the file positive.txt/ negative.txt in it. Delete it.
  So, after deletion, the positive.txt/ negative.txt file would only contain the names of the image.
- A large dataset of positive images is created by applying perspective transform (rotating the images at various angle and changing the intensity of light). The amount of randmoness can

be controlled by varing the command line arguments of opencv_createsamples.exe in folder opencv/build/x64/vc14/bin/

- **Command line arguments:**
- -info <collection_file_name> : Description file of positive images positive.txt. (Multiple Positive Image)
- -img <image_file_name> : One Positive Image (e.g., a company logo).
- -bg : Background description file; contains a list of images which are used as a background for randomly distorted versions of the object, negative.txt, use with option -img
- -vec : Name of the output file *.vec containing the positive samples for training.

TRAIN CASCADE



```
C:\Windows>opencv_createsamples.exe
Usage: opencv_createsamples.exe
  [-info <collection_file_name>]
  [-img <image_file_name>]
  [-vec <vec_file_name>]
  [-bg <background_file_name>]
  [-num <number_of_samples = 1000>]
  [-bgcolor <background_color = 0>]
  [-inv] [-randinv] [-bgthresh <background_color_threshold = 80>]
  [-maxidev <max_intensity_deviation = 40>]
  [-maxxangle <max_x_rotation_angle = 1.100000>]
  [-maxyangle <max_y_rotation_angle = 1.100000>]
  [-maxzangle <max_z_rotation_angle = 0.500000>]
  [-show [<scale = 4.000000>]]
  [-w <sample_width = 24>]
  [-h <sample_height = 24>]
  [-maxscale <max sample scale = -1.000000>]
  [-rngseed <rng seed = 12345>]
```

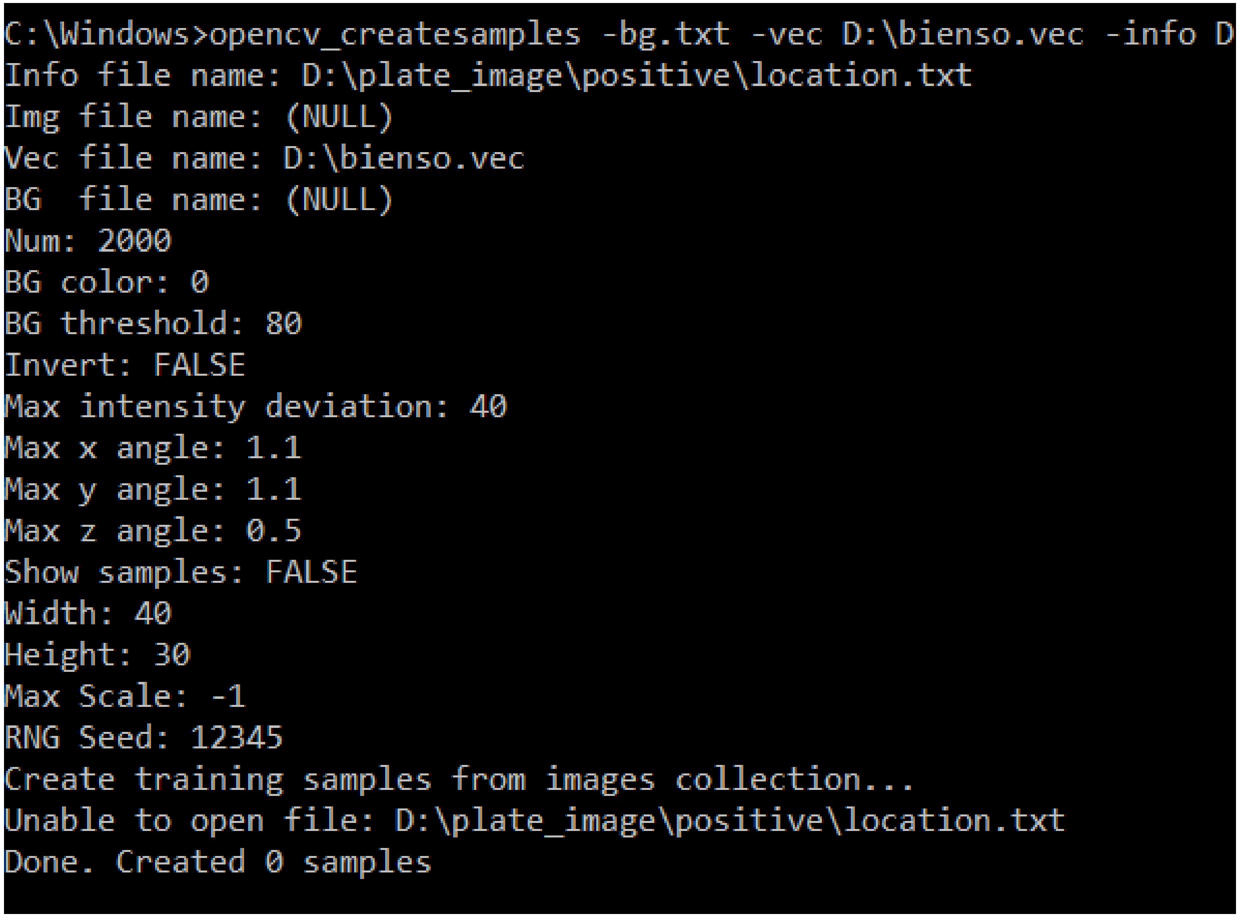
- -num :Number of positive samples to generate.
- -bgcolor –bgthresh:Background color (currently grayscale images are assumed); the background color denotes the transparent color. Since there might be compression artifacts, the amount of color tolerance can be specified by -bgthresh. All pixels within bgcolor-bgthresh and bgcolor+bgthresh range are interpreted as transparent.
- -inv: If specified, colors will be inverted.
- -randinv: If specified, colors will be inverted randomly.
- -maxidev : Maximal intensity deviation of pixels in foreground samples.

- -maxxangle , -maxyangle , -maxzangle: Maximum rotation angles must be given in radians.
- -show: Useful debugging option. If specified, each sample will be shown. Pressing Esc will continue the samples creation process without.
- -w: Width (in pixels) of the output samples.
- -h :Height (in pixels) of the output samples.
- -pngoutput: With this option switched on opencv_createsamples tool generates a collection of PNG samples and a number of associated annotation files, instead of a single vec file.

- Copy all these .exe (opencv_createsamples.exe , opencv_traincascade.exe) in ImageSample folder.
- **Step 2**:The next step is to create a positive .vec file
- Open Notepad
- Type the following command for a single image called my_image.jpg:
- C:\ImageSample\opencv_createsamples.exe -img \positive\ my_image.jpg -bg -vec samples.vec -num 1000 -w 30 -h 30 PAUSE
- Note:Though we have taken one positive image we are specifying -num 250.Because it will do perspective transformation and generate 250 positive images.
- And save it with .bat extension.
- Now double click the .bat file created.

TRAIN CASCADE

opencv_createsamples.exe -vec D:\bienso.vec -info D:\plate_image\positive\location.txt -num 2000 -w 40 -h 30



```
C:\Windows>opencv_createsamples -bg.txt -vec D:\bienso.vec -info D
Info file name: D:\plate_image\positive\location.txt
Img file name: (NULL)
Vec file name: D:\bienso.vec
BG  file name: (NULL)
Num: 2000
BG color: 0
BG threshold: 80
Invert: FALSE
Max intensity deviation: 40
Max x angle: 1.1
Max y angle: 1.1
Max z angle: 0.5
Show samples: FALSE
Width: 40
Height: 30
Max Scale: -1
RNG Seed: 12345
Create training samples from images collection...
Unable to open file: D:\plate_image\positive\location.txt
Done. Created 0 samples
```

TRAIN CASCADE One Positive Image

- **Use option img**

When running opencv_createsamples in this way, the following procedure is used to create a sample object instance: The given source image is rotated randomly around all three axes. The chosen angle is limited by -maxxangle, -maxyangle and -maxzangle. Then pixels having the intensity from the [bg_color-bg_color_threshold; bg_color+bg_color_threshold] range are interpreted as transparent. White noise is added to the intensities of the foreground. If the -inv key is specified then foreground pixel intensities are inverted. If -randinv key is specified then algorithm randomly selects whether inversion should be applied to this sample. Finally, the obtained image is placed onto an arbitrary background from the background description file, resized to the desired size specified by -w and -h and stored to the vec-file, specified by the -vec command line option.

TRAIN CASCADE Multiple Positive Image

- **Use option info**

Positive samples also may be obtained from a collection of previously marked up images, which is the desired way when building robust object models. This collection is described by a text file similar to the background description file. Each line of this file corresponds to an image. The first element of the line is the filename, followed by the number of object annotations, followed by numbers describing the coordinates of the objects bounding rectangles (x, y, width, height).

In order to create positive samples from such collection, -info argument should be specified instead of -img:

-info info.dat : Description file of marked up images collection.

TRAIN CASCADE Multiple Positive Image

Directory structure:

/img

img1.jpg

img2.jpg

info.dat

File info.dat:

img/img1.jpg 1 140 100 45 45

img/img2.jpg 2 100 200 50 50 50 30 25 25

Image img1.jpg contains single object instance with the following coordinates of bounding rectangle: (140, 100, 45, 45). Image img2.jpg contains two object instances.

TRAIN CASCADE Multiple Positive Image

Note that in this case, parameters like -bg, -bgcolor, -bgthreshold, -inv, -randinv, -maxxangle, -maxyangle, -maxzangle are simply ignored and not used anymore. The scheme of samples creation in this case is as follows. The object instances are taken from the given images, by cutting out the supplied bounding boxes from the original images. Then they are resized to target samples size (defined by -w and -h) and stored in output vec-file, defined by the -vec parameter. No distortion is applied, so the only affecting arguments are -w, -h, -show and -num.

The manual process of creating the -info file can also been done by using the opencv_annotation tool. This is an open source tool for visually selecting the regions of interest of your object instances in any given images. The following subsection will discuss in more detail on how to use this application.

opencv_annotation tool

- Using the tool is quite straightforward. The tool accepts several required and some optional parameters:
- --annotations (required) : path to annotations txt file, where you want to store your annotations, which is then passed to the -info parameter [example - /data/annotations.txt]
- --images (required) : path to folder containing the images with your objects [example - /data/testimages/]
- --maxWindowHeight (optional) : if the input image is larger in height than the given resolution here, resize the image for easier annotation, using --resizeFactor.
- --resizeFactor (optional) : factor used to resize the input image when using the --maxWindowHeight parameter.
- opencv_annotation --annotations=/path/to/annotations/file.txt --images=/path/to/image/folder/

opencv_annotation tool

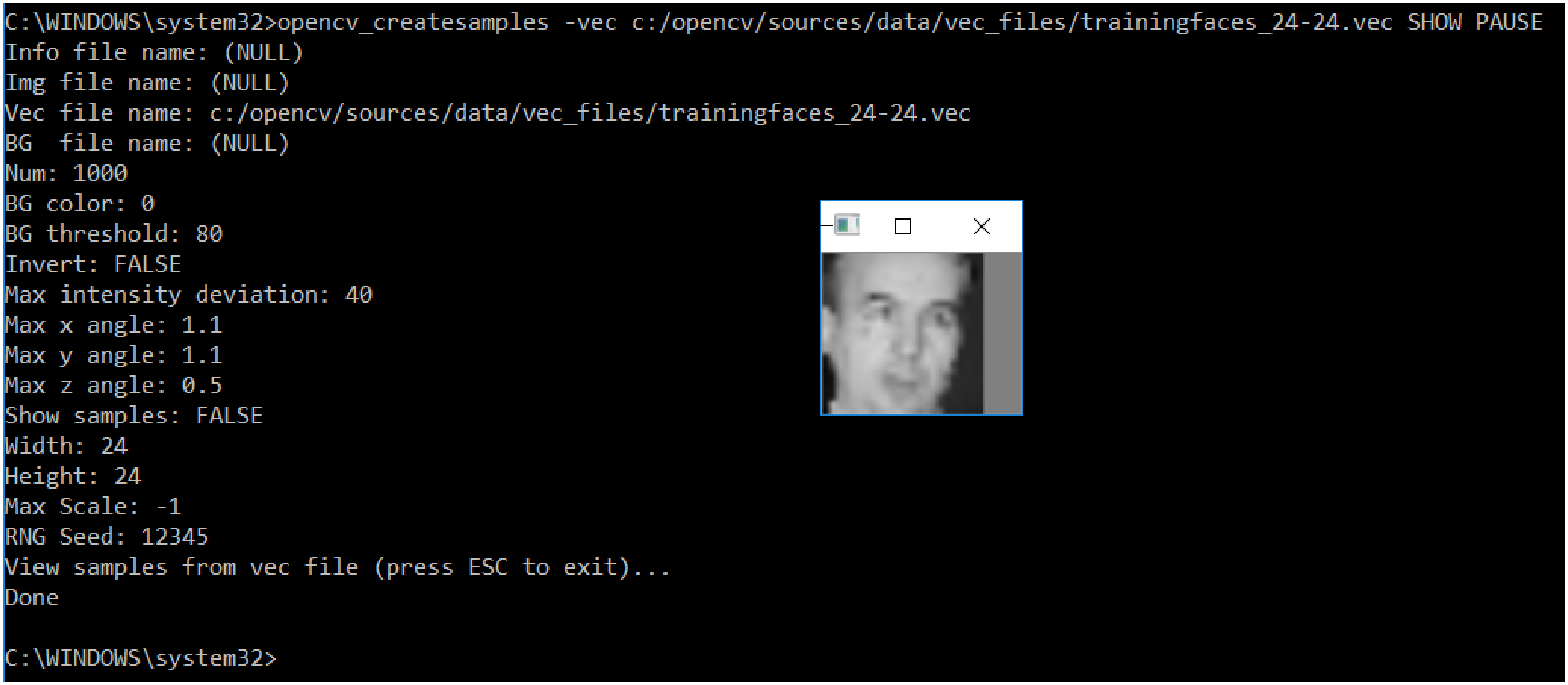
- This command will fire up a window containing the first image and your mouse cursor which will be used for annotation. The left mouse button is used to select the first corner of your object, then keeps drawing until you are fine, and stops when a second left mouse button click is registered. After each selection you have the following choices:
  - Pressing c : confirm the annotation, turning the annotation green and confirming it is stored
  - Pressing d : delete the last annotation from the list of annotations (easy for removing wrong annotations)
  - Pressing n : continue to the next image
  - Pressing ESC : this will exit the annotation software.
- Finally you will end up with a usable annotation file that can be passed to the -info argument of opencv_createsamples.

Showing the content of the vec file

- opencv_createsamples utility may be used for examining samples stored in positive samples file. In order to do this only -vec, -w and -h parameters should be specified.
- Note that for training, it does not matter how vec-files with positive samples are generated. But opencv_createsamples utility is the only one way to collect/create a vector file of positive samples, provided by OpenCV.
- Example of vec-file is available here opencv/sources/data/vec_files/trainingfaces_24-24.vec. It can be used to train a face detector with the following window size: -w 24 -h 24.
- Open vec file

```
opencv_createsamples -vec c:/opencv/sources/data/vec_files/trainingfaces_24-24.vec SHOW
PAUSE
```

Showing the content of the vec file



```
C:\WINDOWS\system32>opencv_createsamples -vec c:/opencv/sources/data/vec_files/trainingfaces_24-24.vec SHOW PAUSE
Info file name: (NULL)
Img file name: (NULL)
Vec file name: c:/opencv/sources/data/vec_files/trainingfaces_24-24.vec
BG  file name: (NULL)
Num: 1000
BG color: 0
BG threshold: 80
Invert: FALSE
Max intensity deviation: 40
Max x angle: 1.1
Max y angle: 1.1
Max z angle: 0.5
Show samples: FALSE
Width: 24
Height: 24
Max Scale: -1
RNG Seed: 12345
View samples from vec file (press ESC to exit)...
Done

C:\WINDOWS\system32>
```

opencv_traincascade

- The next step is the actual training of the boosted cascade of weak classifiers, based on the positive and negative dataset that was prepared beforehand.
- Command line arguments of opencv_traincascade application grouped by purposes:

○ -data <cascade_dir_name> : Where the trained classifier *.xml should be stored. This folder should be created manually beforehand.

○ -vec <vec_file_name> : vec-file with positive samples (created by opencv_createsamples utility).

○ -bg <background_file_name> : Background description file. This is the file containing the negative sample images.

○ -numPos <number_of_positive_samples> : Number of positive samples used in training for every classifier stage.

opencv_traincascade

- -numNeg <number_of_negative_samples> : Number of negative samples used in training for every classifier stage.
- -numStages <number_of_stages> : Number of cascade stages to be trained.
- -precalcValBufSize <precalculated_vals_buffer_size_in_Mb> : Size of buffer for precalculated feature values (in Mb). The more memory you assign the faster the training process, however keep in mind that -precalcValBufSize and -precalcIdxBufSize combined should not exceed you available system memory.
- -precalcIdxBufSize <precalculated_idxs_buffer_size_in_Mb> : Size of buffer for precalculated feature indices (in Mb). The more memory you assign the faster the training process, however keep in mind that -precalcValBufSize and -precalcIdxBufSize combined should not exceed you available system memory.

opencv_traincascade

- -baseFormatSave : This argument is actual in case of Haar-like features. If it is specified, the cascade will be saved in the old format. This is only available for backwards compatibility reasons and to allow users stuck to the old deprecated interface, to at least train models using the newer interface.
- -numThreads <max_number_of_threads> : Maximum number of threads to use during training. Notice that the actual number of used threads may be lower, depending on your machine and compilation options.
- -acceptanceRatioBreakValue <break_value> : This argument is used to determine how precise your model should keep learning and when to stop. A good guideline is to train not further than 10e-5, to ensure the model does not overtrain on your training data. By default this value is set to -1 to disable this feature.

opencv_traincascade

- Cascade parameters:
- ○ -stageType <BOOST(default)> : Type of stages. Only boosted classifiers are supported as a stage type at the moment.
- ○ -featureType<{HAAR(default), LBP}> : Type of features: HAAR - Haar-like features, LBP - local binary patterns.
- ○ -w <sampleWidth> : Width of training samples (in pixels). Must have exactly the same value as used during training samples creation (opencv_createsamples utility).
- ○ -h <sampleHeight> : Height of training samples (in pixels). Must have exactly the same value as used during training samples creation (opencv_createsamples utility).

opencv_traincascade

- Boosted classifer parameters:
- ○ -bt <{DAB, RAB, LB, GAB(default)}> : Type of boosted classifiers: DAB - Discrete AdaBoost, RAB - Real AdaBoost, LB - LogitBoost, GAB - Gentle AdaBoost.
- ○ -minHitRate <min_hit_rate> : Minimal desired hit rate for each stage of the classifier. Overall hit rate may be estimated as (min_hit_rate ^ number_of_stages), [180] §4.1.
- ○ -maxFalseAlarmRate <max_false_alarm_rate> : Maximal desired false alarm rate for each stage of the classifier. Overall false alarm rate may be estimated as (max_false_alarm_rate ^ number_of_stages), [180] §4.1.
- ○ -weightTrimRate <weight_trim_rate> : Specifies whether trimming should be used and its weight. A decent choice is 0.95.

opencv_traincascade

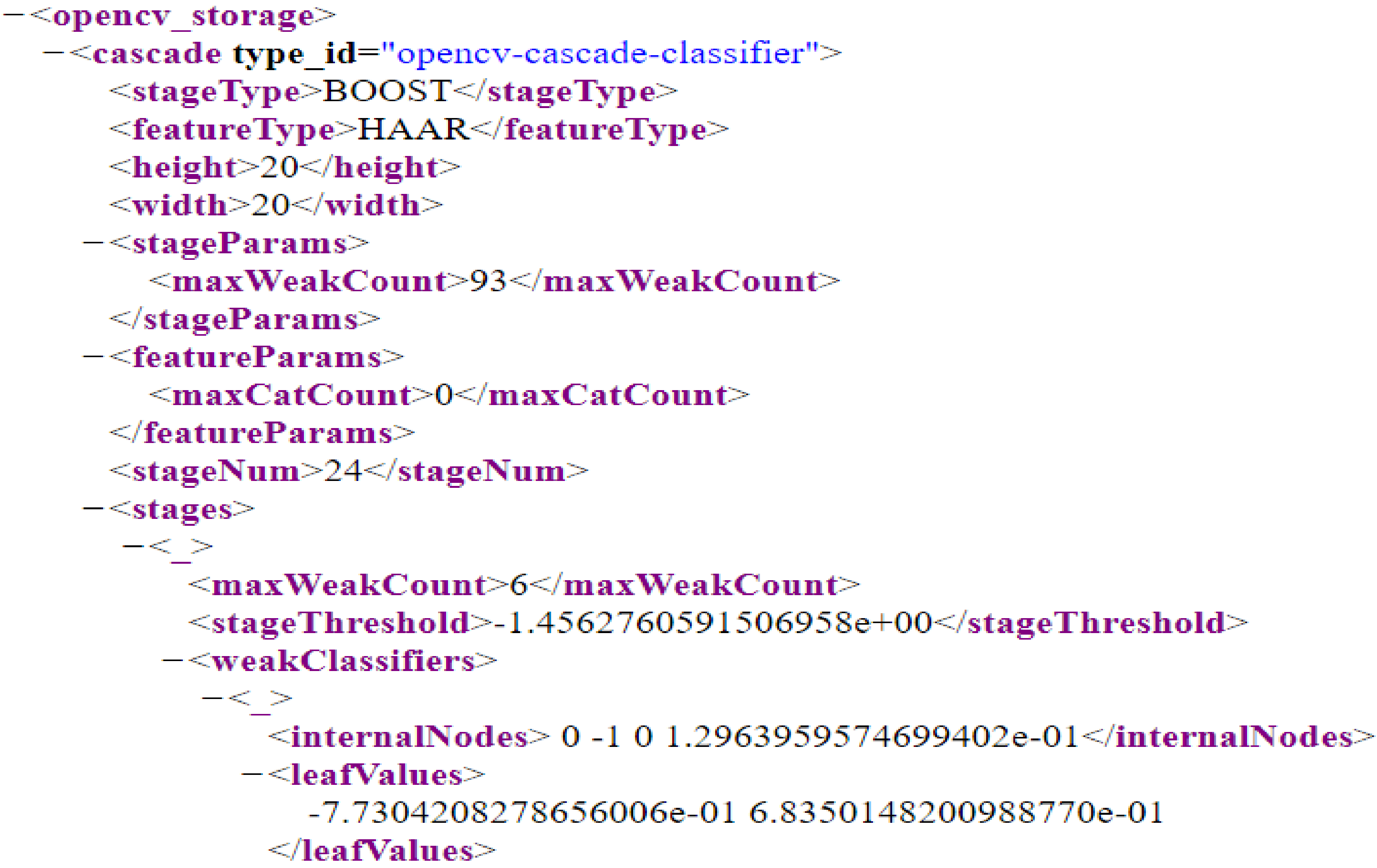
- o -maxDepth <max_depth_of_weak_tree> : Maximal depth of a weak tree. A decent choice is 1, that is case of stumps.
- o -maxWeakCount <max_weak_tree_count> : Maximal count of weak trees for every cascade stage. The boosted classifier (stage) will have so many weak trees (<=maxWeakCount), as needed to achieve the given -maxFalseAlarmRate.
- Haar-like feature parameters:
- o -mode <BASIC (default) | CORE | ALL> : Selects the type of Haar features set used in training. BASIC use only upright features, while ALL uses the full set of upright and 45 degree rotated feature set. See [103] for more details.
- Local Binary Patterns parameters: Local Binary Patterns don't have parameters.

opencv_traincascade

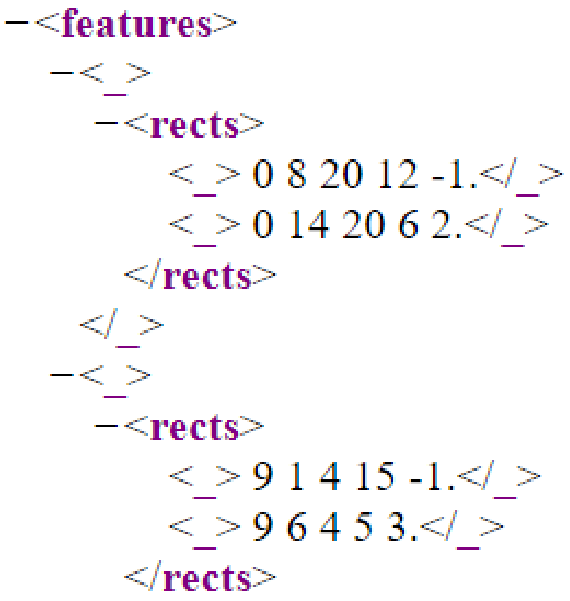
- After the opencv_traincascade application has finished its work, the trained cascade will be saved in *.xml file in the -data folder. Other files in this folder are created for the case of interrupted training, so you may delete them after completion of training.
- Training is finished and you can test your cascade classifier!
- Available training data are in folder c:/opencv/sources/data
- Xml file can be read by browser

C:/opencv/sources/data/haarcascades/haarcascade_eye.xml

opencv_traincascade



```
– <opencv_storage>
  – <cascade type_id="opencv-cascade-classifier">
      <stageType>BOOST</stageType>
      <featureType>HAAR</featureType>
      <height>20</height>
      <width>20</width>
    – <stageParams>
        <maxWeakCount>93</maxWeakCount>
      </stageParams>
    – <featureParams>
        <maxCatCount>0</maxCatCount>
      </featureParams>
      <stageNum>24</stageNum>
    – <stages>
      – <_>
          <maxWeakCount>6</maxWeakCount>
          <stageThreshold>-1.4562760591506958e+00</stageThreshold>
        – <weakClassifiers>
          – <_>
              <internalNodes> 0 -1 0 1.2963959574699402e-01</internalNodes>
            – <leafValues>
                -7.7304208278656006e-01 6.8350148200988770e-01
              </leafValues>
```



```
−<features>
   −<_>
      −<rects>
           <_> 0 8 20 12 -1.</_>
           <_> 0 14 20 6 2.</_>
        </rects>
     </_>
   −<_>
      −<rects>
           <_> 9 1 4 15 -1.</_>
           <_> 9 6 4 5 3.</_>
        </rects>
```

opencv_visualisation

- It can be usefull to visualise the trained cascade, to see which features it selected and how complex its stages are. For this OpenCV supplies a opencv_visualisation application.
  - --image (required) : path to a reference image for your object model. This should be an annotation with dimensions [-w,-h] as passed to both opencv_createsamples and opencv_traincascade application.
  - --model (required) : path to the trained model, which should be in the folder supplied to the -data parameter of the opencv_traincascade application.
  - --data (optional) : if a data folder is supplied, which has to be manually created beforehand, stage output and a video of the features will be stored.
- opencv_visualisation --image=/data/object.png --model=/data/model.xml --data=/data/result/

OBJECT DETECT MATLAB

The cascade object detector uses the Viola-Jones algorithm to detect people's faces, noses, eyes, mouth, or upper body. You can also use the Image Labeler to train a custom classifier to use with this System object

To detect facial features or upper body in an image:

Create the vision.CascadeObjectDetector object and set its properties.

Call the object with arguments, as if it were a function.

```
detector = vision.CascadeObjectDetector
detector = vision.CascadeObjectDetector(model)
detector = vision.CascadeObjectDetector(XMLFILE)
detector = vision.CascadeObjectDetector(Name,Value)
```

OBJECT DETECT MATLAB

detector = vision.CascadeObjectDetector creates a detector to detect objects using the Viola-Jones algorithm.

detector = vision.CascadeObjectDetector(model) creates a detector configured to detect objects defined by the input character vector, model.

detector = vision.CascadeObjectDetector(XMLFILE) creates a detector and configures it to use the custom classification model specified with the XMLFILE input.

detector = vision.CascadeObjectDetector(Name,Value) sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, detector = vision.CascadeObjectDetector('ClassificationModel','Up

OBJECT DETECT MATLAB

Trained cascade classification model, specified as a character vector. The ClassificationModel property controls the type of object to detect. By default, the detector is configured to detect faces.

You can set this character vector to an XML file containing a custom classification model, or to one of the valid model character vectors listed below. You can train a custom classification model using the trainCascadeObjectDetector function. The function can train the model using Haar-like features, histograms of oriented gradients (HOG), or local binary patterns (LBP).

OBJECT DETECT MATLAB

'FrontalFaceCART'(Default),'FrontalFaceLBP‘

'UpperBody‘, 'EyePairBig‘, 'EyePairSmall‘, 'LeftEye'

'RightEye‘, 'LeftEyeCART‘, 'RightEyeCART‘, 'ProfileFace‘

'Mouth‘, 'Nose‘

MinSize — Size of smallest detectable object

[] (default) | two-element vector

MaxSize — Size of largest detectable object

[] (default) | two-element vector

ScaleFactor — Scaling for multiscale object detection

1.1 (default) | scalar

MergeThreshold — Detection threshold

4 (default) | integer

UseROI — Use region of interest false (default) | false

OBJECT DETECT MATLAB

bbox = detector(I)

bbox = detector(I,roi)

bbox = detector(I) returns an M-by-4 matrix, bbox, that defines M bounding boxes containing the detected objects. The detector performs multiscale object detection on the input image, I.

bbox = detector(I,roi) detects objects within the rectangular search region specified by roi. Set the 'UseROI' property to true

to use this syntax.

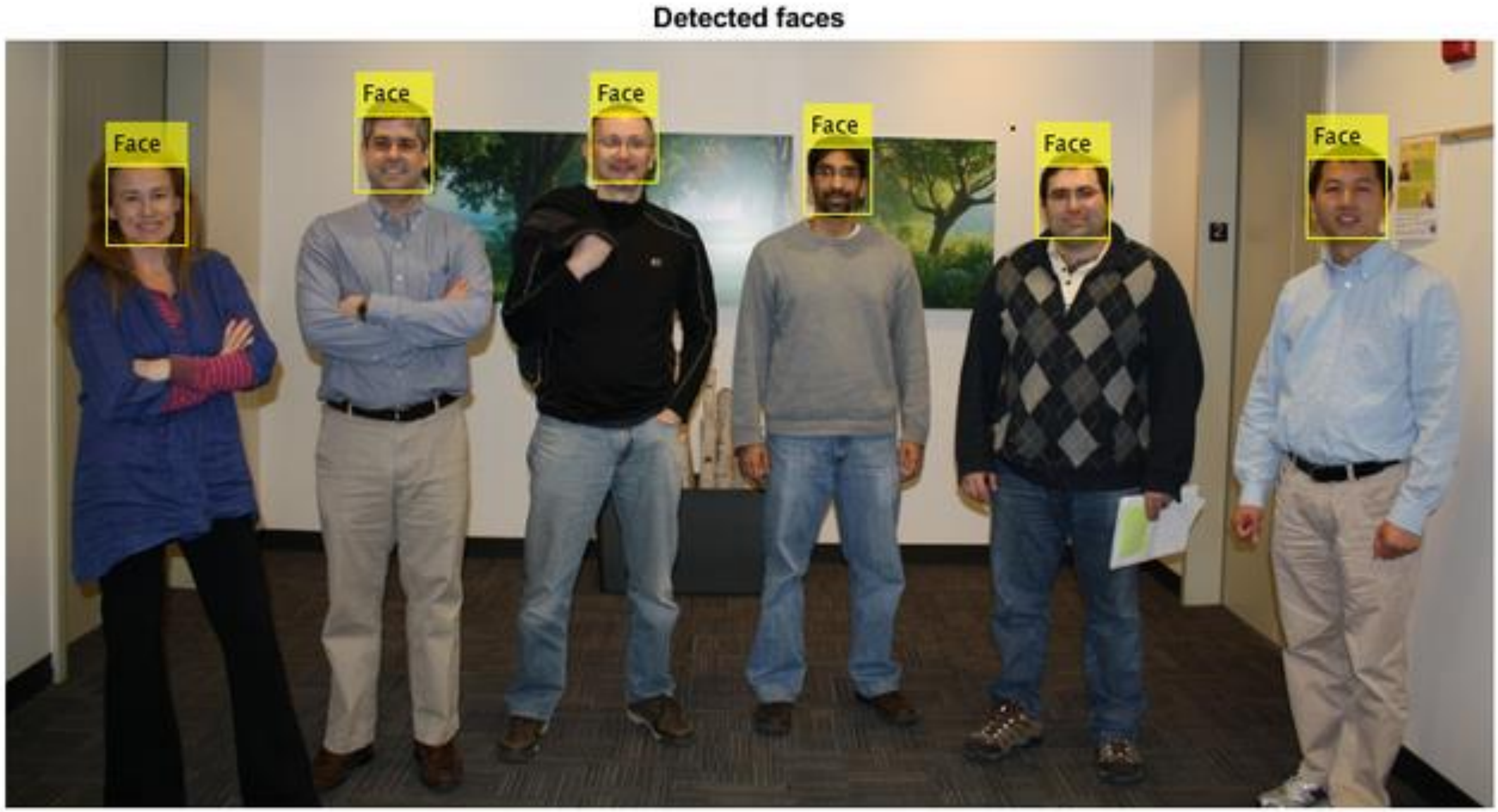
I — Input image grayscale | truecolor (RGB)

model — Classification model 'FrontalFaceCART' (default) | character string

FACE DETECT MATLAB

```
close all;
faceDetector = vision.CascadeObjectDetector;
%Read the input image.
I = imread('visionteam.jpg');
%Detect faces.
bboxes = faceDetector(I);
%Annotate detected faces.
IFaces = insertObjectAnnotation(I,'rectangle',bboxes,'Face');
figure, imshow(IFaces),title('Detected faces');
```

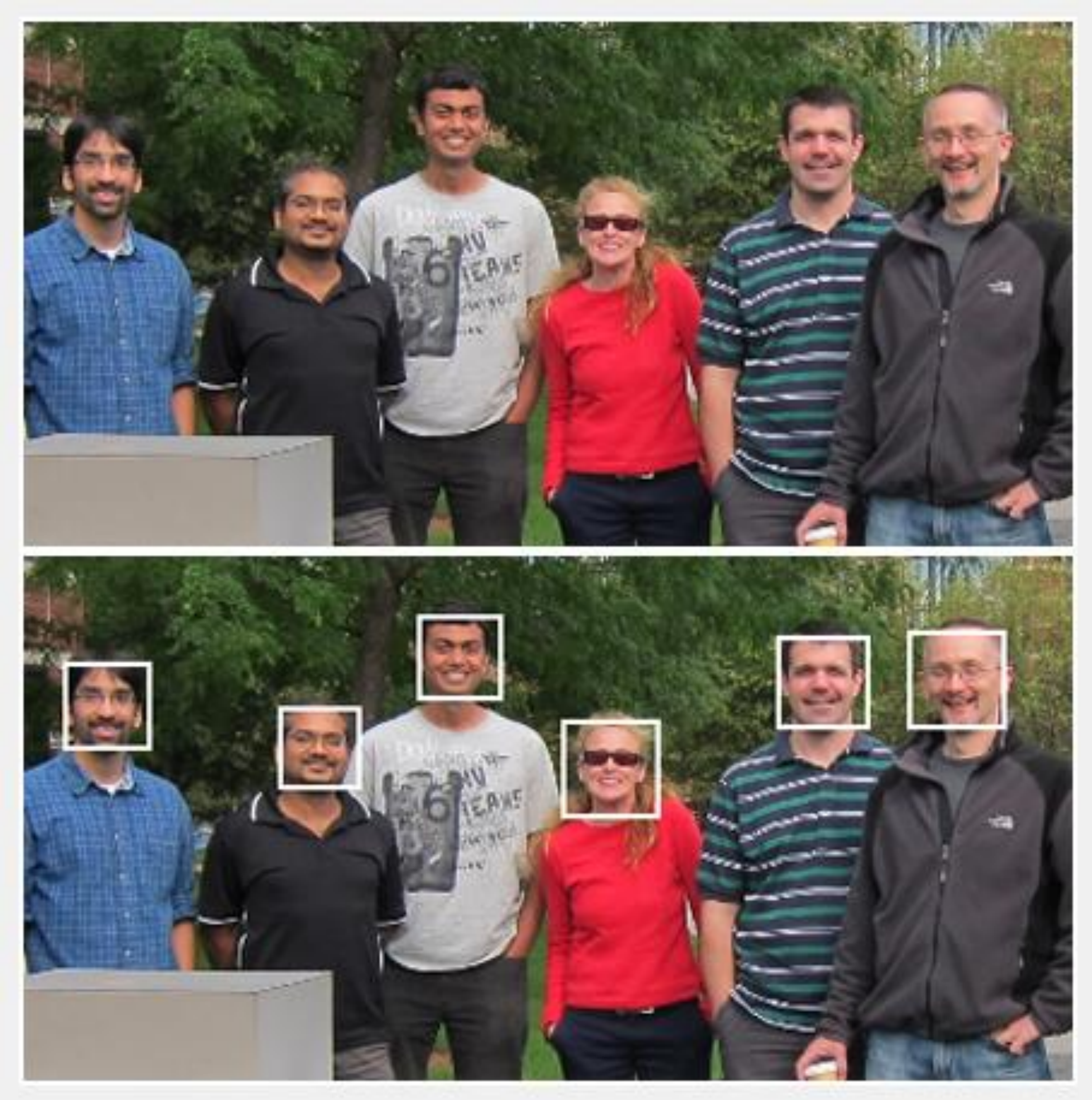
FACE DETECT MATLAB



FACE DETECT MATLAB

```
close all;
faceDetector = vision.CascadeObjectDetector;
shapeInserter = vision.ShapeInserter('BorderColor','Custom','CustomBorderColor',[0 255 255]);
I = imread('d:/faces.jpg');
imshow(I);shg;
bbox = step(faceDetector, I);
% Draw boxes around detected faces and display results
I_faces = step(shapeInserter, I, int32(bbox));
imshow(I_faces), title('Detected faces');
```

OBJECT DETECT MATLAB

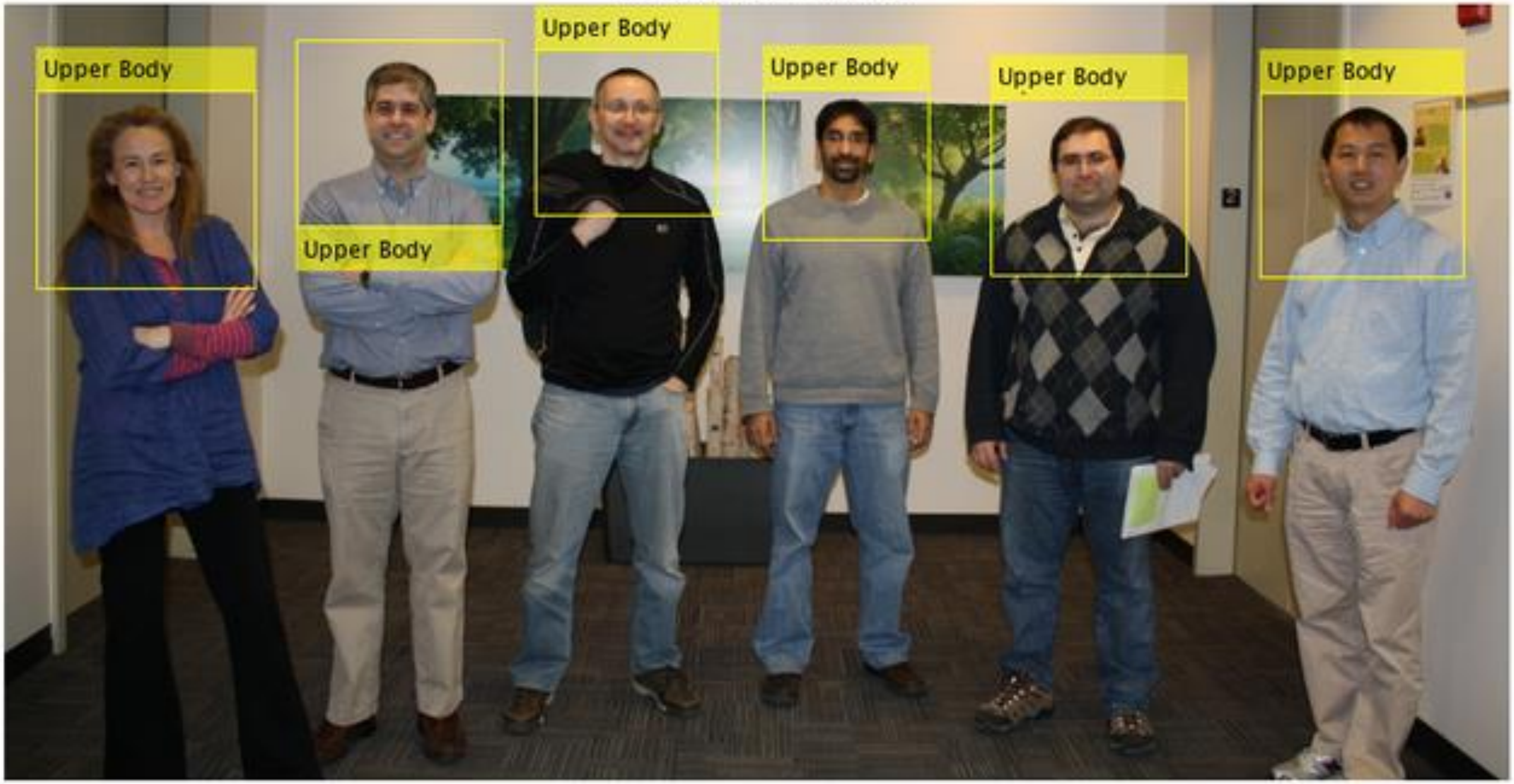


UPPER BODY DETECTOR MATLAB

```
%Create a body detector object and set properties.
bodyDetector = vision.CascadeObjectDetector('UpperBody');
bodyDetector.MinSize = [60 60];
bodyDetector.MergeThreshold = 10;
%Read input image and detect upper body.
I2 = imread('visionteam.jpg');
bboxBody = bodyDetector(I2);
%Annotate detected upper bodies.
IBody = insertObjectAnnotation(I2,'rectangle',bboxBody,'Upper Body');
figure,imshow(IBody),title('Detected upper bodies');
```

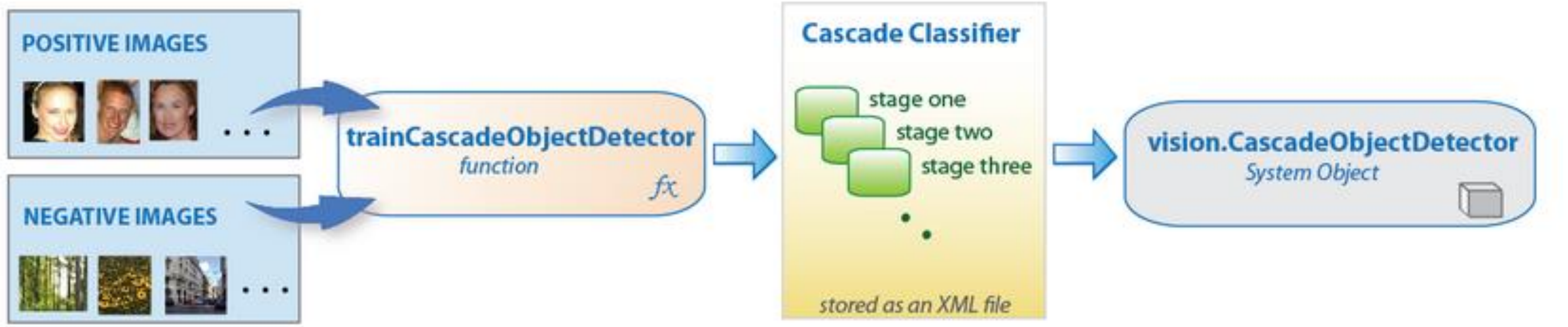
UPPER BODY DETECTOR MATLAB

Detected upper bodies



Train a Cascade Object Detector MATLAB

- The vision.CascadeObjectDetector System object comes with several pretrained classifiers for detecting frontal faces, profile faces, noses, eyes, and the upper body. However, these classifiers are not always sufficient for a particular application. Computer Vision System Toolbox™ provides the trainCascadeObjectDetector function to train a custom classifier.



Train a Cascade Object Detector MATLAB

The Computer Vision System Toolbox cascade object detector can detect object categories whose aspect ratio does not vary significantly. Objects whose aspect ratio remains fixed include faces, stop signs, and cars viewed from one side.

The vision.CascadeObjectDetector System object detects objects in images by sliding a window over the image. The detector then uses a cascade classifier to decide whether the window contains the object of interest. The size of the window varies to detect objects at different scales, but its aspect ratio remains fixed. The detector is very sensitive to out-of-plane rotation, because the aspect ratio changes for most 3-D objects. Thus, you need to train a detector for each orientation of the object. Training a single detector to handle all orientations will not work.

Train a Cascade Object Detector MATLAB

The cascade classifier consists of stages, where each stage is an ensemble of weak learners. The weak learners are simple classifiers called decision stumps. Each stage is trained using a technique called boosting. Boosting provides the ability to train a highly accurate classifier by taking a weighted average of the decisions made by the weak learners.

Each stage of the classifier labels the region defined by the current location of the sliding window as either positive or negative. Positive indicates that an object was found and negative indicates no objects were found. If the label is negative, the classification of this region is complete, and the detector slides the window to the next location. If the label is positive, the classifier passes the region to the next stage. The detector reports an object found at the current window location when the final stage classifies the region as positive.

Train a Cascade Object Detector MATLAB

The stages are designed to reject negative samples as fast as possible. The assumption is that the vast majority of windows do not contain the object of interest. Conversely, true positives are rare and worth taking the time to verify.

A true positive occurs when a positive sample is correctly classified. A false positive occurs when a negative sample is mistakenly classified as positive. A false negative occurs when a positive sample is mistakenly classified as negative.

To work well, each stage in the cascade must have a low false negative rate. If a stage incorrectly labels an object as negative, the classification stops, and you cannot correct the mistake.

trainCascadeObjectDetector

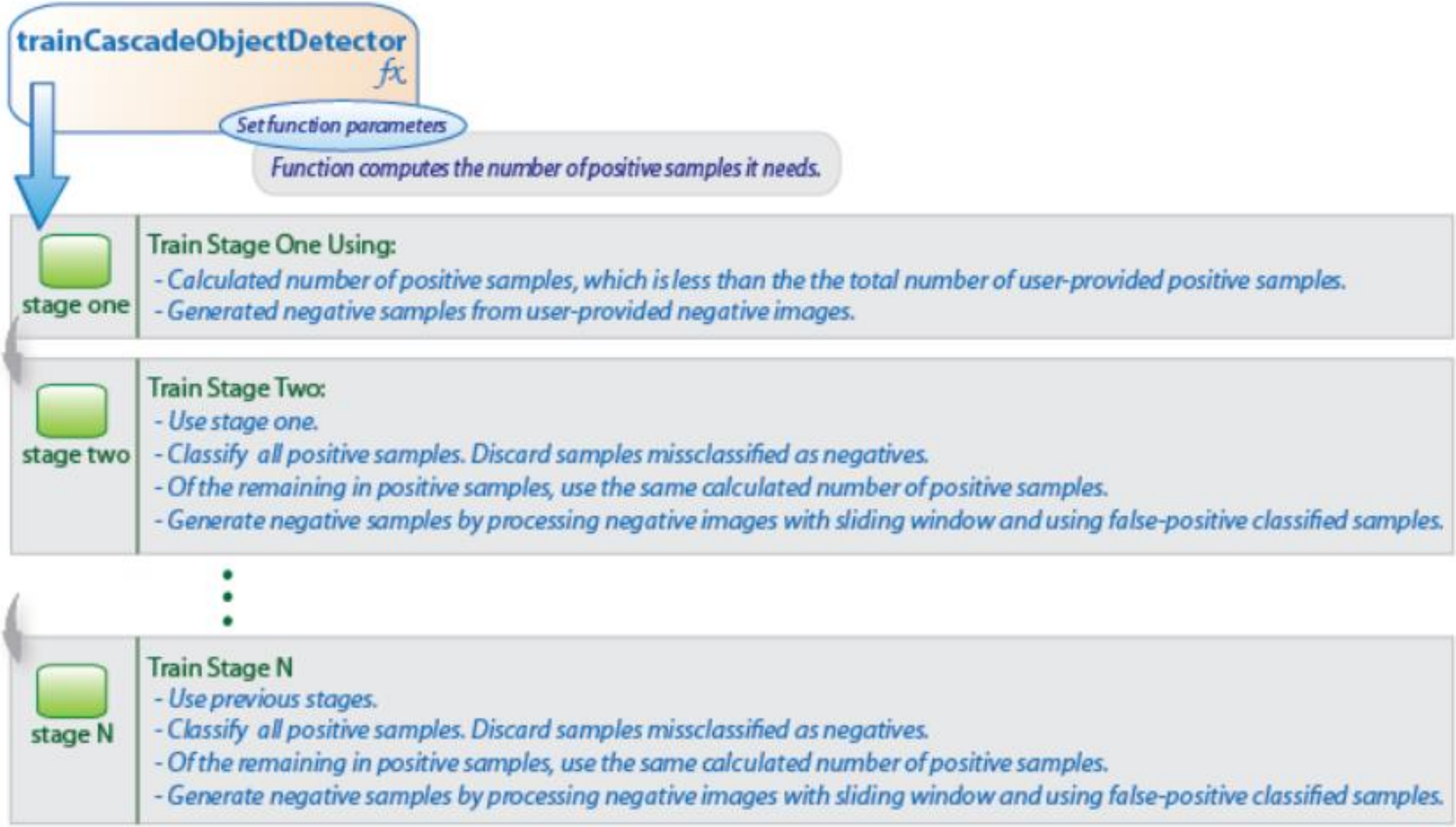
However, each stage can have a high false positive rate. Even if the detector incorrectly labels a nonobject as positive, you can correct the mistake in subsequent stages.

The overall false positive rate of the cascade classifier is fs, where f is the false positive rate per stage in the range (0 1), and s is the number of stages. Similarly, the overall true positive rate is ts, where t is the true positive rate per stage in the range (0 1]. Thus, adding more stages reduces the overall false positive rate, but it also reduces the overall true positive rate.

Train a Cascade Object Detector MATLAB

Cascade classifier training requires a set of positive samples and a set of negative images. You must provide a set of positive images with regions of interest specified to be used as positive samples. You can use the Image Labeler to label objects of interest with bounding boxes. The Image Labeler outputs a table to use for positive samples. You also must provide a set of negative images from which the function generates negative samples automatically. To achieve acceptable detector accuracy, set the number of stages, feature type, and other function parameters.

Train a Cascade Object Detector MATLAB



Function computes the number of positive samples it needs.

Train Stage One Using:
- Calculated number of positive samples, which is less than the the total number of user-provided positive samples.
- Generated negative samples from user-provided negative images.

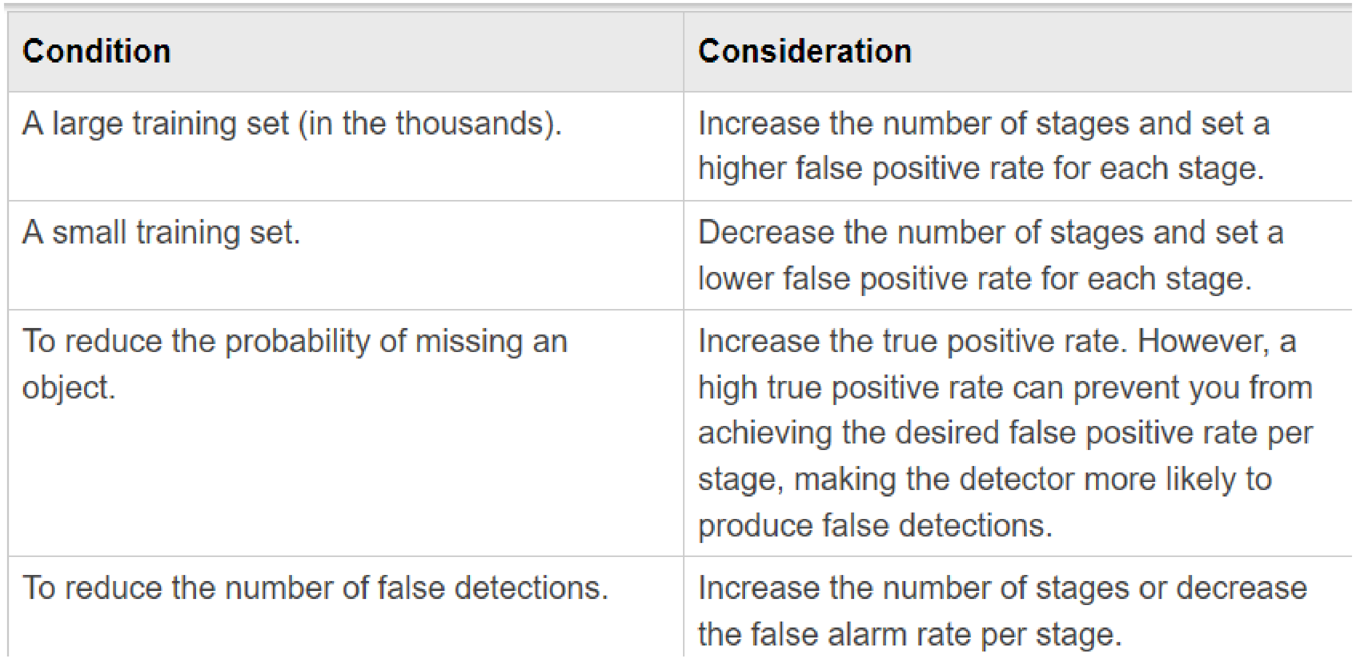
Train Stage Two:
- Use stage one.
- Classify all positive samples. Discard samples missclassified as negatives.
- Of the remaining in positive samples, use the same calculated number of positive samples.
- Generate negative samples by processing negative images with sliding window and using false-positive classified samples.

Train Stage N
- Use previous stages.
- Classify all positive samples. Discard samples missclassified as negatives.
- Of the remaining in positive samples, use the same calculated number of positive samples.
- Generate negative samples by processing negative images with sliding window and using false-positive classified samples.

Train a Cascade Object Detector MATLAB

Considerations when Setting Parameters

Select the function parameters to optimize the number of stages, the false positive rate, the true positive rate, and the type of features to use for training. When you set the parameters, consider these tradeoffs.



Condition	Consideration
A large training set (in the thousands).	Increase the number of stages and set a higher false positive rate for each stage.
A small training set.	Decrease the number of stages and set a lower false positive rate for each stage.
To reduce the probability of missing an object.	Increase the true positive rate. However, a high true positive rate can prevent you from achieving the desired false positive rate per stage, making the detector more likely to produce false detections.
To reduce the number of false detections.	Increase the number of stages or decrease the false alarm rate per stage.

Train a Cascade Object Detector MATLAB

Feature Types Available for Training

Choose the feature that suits the type of object detection you need. The trainCascadeObjectDetector supports three types of features: Haar, local binary patterns (LBP), and histograms of oriented gradients (HOG). Haar and LBP features are often used to detect faces because they work well for representing fine-scale textures. The HOG features are often used to detect objects such as people and cars. They are useful for capturing the overall shape of an object. For example, in the following visualization of the HOG features, you can see the outline of the bicycle.

Train a Cascade Object Detector MATLAB

You might need to run the trainCascadeObjectDetector function multiple times to tune the parameters. To save time, you can use LBP or HOG features on a small subset of your data. Training a detector using Haar features takes much longer. After that, you can run the Haar features to see if the accuracy improves.

Supply Positive Samples

To create positive samples easily, you can use the Image Labeler app. The Image Labeler provides an easy way to label positive samples by interactively specifying rectangular regions of interest (ROIs).

You can also specify positive samples manually in one of two ways. One way is to specify rectangular regions in a larger image. The regions contain the objects of interest.

Train a Cascade Object Detector MATLAB

The other approach is to crop out the object of interest from the image and save it as a separate image. Then, you can specify the region to be the entire image. You can also generate more positive samples from existing ones by adding rotation or noise, or by varying brightness or contrast.

Supply Negative Images

Negative samples are not specified explicitly. Instead, the trainCascadeObjectDetector function automatically generates negative samples from user-supplied negative images that do not contain objects of interest. Before training each new stage, the function runs the detector consisting of the stages already trained on the negative images.

Train a Cascade Object Detector MATLAB

Any objects detected from these image are false positives, which are used as negative samples. In this way, each new stage of the cascade is trained to correct mistakes made by previous stages.

As more stages are added, the detector's overall false positive rate decreases, causing generation of negative samples to be more difficult. For this reason, it is helpful to supply as many negative images as possible. To improve training accuracy, supply negative images that contain backgrounds typically associated with the objects of interest. Also, include negative images that contain nonobjects similar in appearance to the objects of interest. For example, if you are training a stop-sign detector, include negative images that contain road signs and shapes similar to a stop sign.

imageLabeler

- Define rectangular regions of interest (ROI) labels, pixel ROI labels, and scene labels, and use these labels to interactively label your ground truth data.
- The Image Labeler app enables you to label ground truth data in a collection of images. Using the app, you can:
- Use built-in detection or tracking algorithms to label your ground truth data.
- Write, import, and use your own custom automation algorithm to automatically label ground truth.
- Evaluate the performance of your label automation algorithms using a visual summary. Export the labeled ground truth as a groundTruth object.

StopSign Detect Matlab

StopSign Detect Matlab e:/../StopsignDetector1.m

```
%Positive data  in folder /vision/visiondata/stopSignImages/
%Negative data  in folder /vision/visiondata/nonStopSigns/
%Use Image Labeler app imageLabeler to label positive samples,
the result is stopSignsAndCars.mat, a 41x4 matrix
%Load positive samples from MAT file. The file contains a table
specifying bounding boxes for several object categories.
load('stopSignsAndCars.mat');
%Select the bounding boxes for stop signs from the table.
positiveInstances = stopSignsAndCars(:,1:2);
Add the image folder to the MATLAB path.
imDir = fullfile(matlabroot,'toolbox','vision','visiondata',...
'stopSignImages'); addpath(imDir);
```

StopSign Detect Matlab

```
negativeFolder = fullfile(matlabroot,'toolbox','vision','visiondata',...
    'nonStopSigns');
negativeImages = imageDatastore(negativeFolder);
trainCascadeObjectDetector('stopSignDetector.xml',positiveInstances,
... negativeFolder,'FalseAlarmRate',0.1,'NumCascadeStages',5);
%Use the newly trained classifier to detect a stop sign in an image.
detector = vision.CascadeObjectDetector('stopSignDetector.xml');
img = imread('stopSignTest.jpg');
%Detect a stop sign.
bbox = step(detector,img);
%Insert bounding box rectangles and return the marked image.
 detectedImg = insertObjectAnnotation(img,'rectangle',bbox,'stop
sign'); figure; imshow(detectedImg); rmpath(imDir);
```