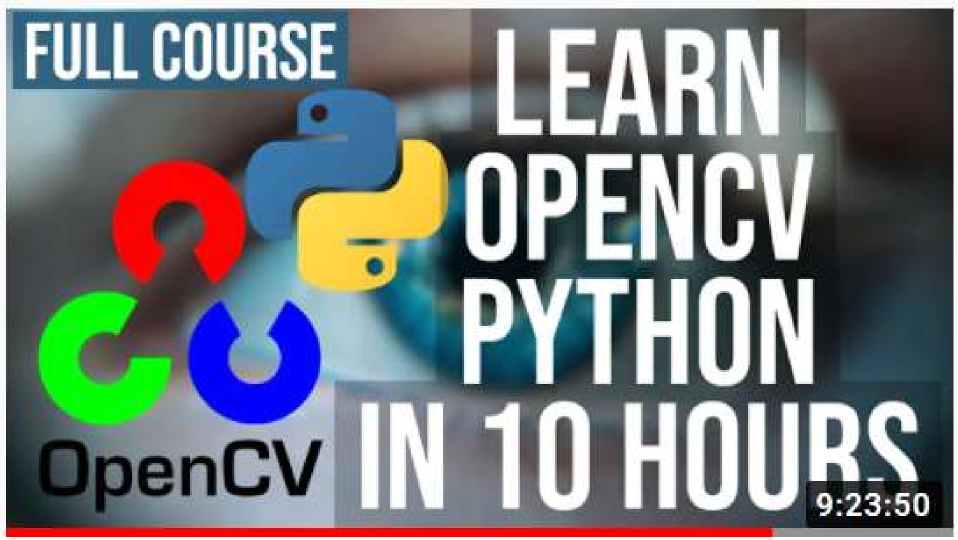
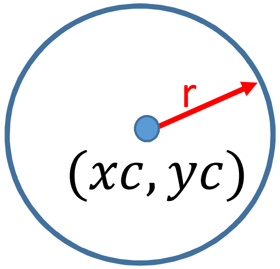
OpenCV Python Tutorial For Beginners (10 hrs)

https://www.youtube.com/watch?v=N81PCpADwKQ&t=27645s

https://www.youtube.com/watch?v=kdLM6AOd2vc&list=PLS1QulWo1RIa7D1O6skqDQ-JZ1GGHKK-K



34. Circle Detection using OpenCV Hough Circle Transform



$$(x-xc)^2+(y-yc)^2=r^2$$

circles=cv.HoughCircles(image, method, dp, minDist[, circles[, param1[, param2[, minRadius[, maxRadius]]]]])

image 8-bit, single-channel, grayscale input image.
circles Output vector of found circles.
method Detection method, see HoughModes. Currently, the only implemented method is HOUGH_GRADIENT
dp Inverse ratio of the accumulator resolution to the image resolution.
minDist Minimum distance between the centers of the detected circles.
param1 First method-specific parameter. In case of HOUGH_GRADIENT , it is the higher threshold of the two passed to the Canny edge detector (the lower one is twice smaller).
param2 Second method-specific parameter. In case of HOUGH_GRADIENT , it is the accumulator threshold for the circle centers at the detection stage.
minRadius Minimum circle radius.
maxRadius Maximum circle radius. If <= 0, uses the maximum image dimension. If < 0, returns centers without finding the radius.

cv.HoughCircles (image, circles, method, dp, minDist, param1 = 100, param2 = 100, minRadius = 0, maxRadius = 0)

Parameters:

image 8-bit, single-channel, grayscale input image.

circles output vector of found circles(cv.CV_32FC3 type). Each vector is encoded as a 3-element floating-point vector (x, y, radius) .

method detection method (see cv.HoughModes). Currently, the only implemented method is HOUGH_GRADIENT

dp inverse ratio of the accumulator resolution to the image resolution. For example, if dp = 1 , the accumulator has the same resolution as the input image. If dp = 2 , the accumulator has half as big width and height.

minDist minimum distance between the centers of the detected circles. If the parameter is too small, multiple neighbor circles may be falsely detected in addition to a true one. If it is too large, some circles may be missed.

cv.HoughCircles (image, circles, method, dp, minDist, param1 = 100, param2 = 100, minRadius = 0, maxRadius = 0)

Parameters:

Param1 first method-specific parameter. In case of HOUGH_GRADIENT , it is the higher threshold of the two passed to the Canny edge detector (the lower one is twice smaller).

Param2 second method-specific parameter. In case of HOUGH_GRADIENT , it is the accumulator threshold for the circle centers at the detection stage. The smaller it is, the more false circles may be detected. Circles, corresponding to the larger accumulator values, will be returned first.

minRadius minimum circle radius.

maxRadius maximum circle radius.

cv.HoughCircles (image, circles, method, dp, minDist, param1 = 100, param2 = 100, minRadius = 0, maxRadius = 0)

ex40.py

13 circles

```python
import numpy as np
import cv2 as cv
img = cv.imread('smarties.png')
output = img.copy()
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
gray = cv.medianBlur(gray, 5)
circles = cv.HoughCircles(gray, cv.HOUGH_GRADIENT, 1, 20,
                          param1=50, param2=30, minRadius=0, maxRadius=0)
detected_circles = np.uint16(np.around(circles))  # (1, 13, 3)
for (x, y ,r) in detected_circles[0, :]:
    cv.circle(output, (x, y), r, (0, 0, 0), 3)  # black circles
    cv.circle(output, (x, y), 2, (0, 255, 255), 3) # yellow centers

cv.imshow('output',output)
cv.waitKey(0)
cv.destroyAllWindows()
```

circles: array([[[376.5, 81.5, 26.6], ...,[320.5, 278.5, 67.4]]], dtype=float32) Shape (1, 13, 3)

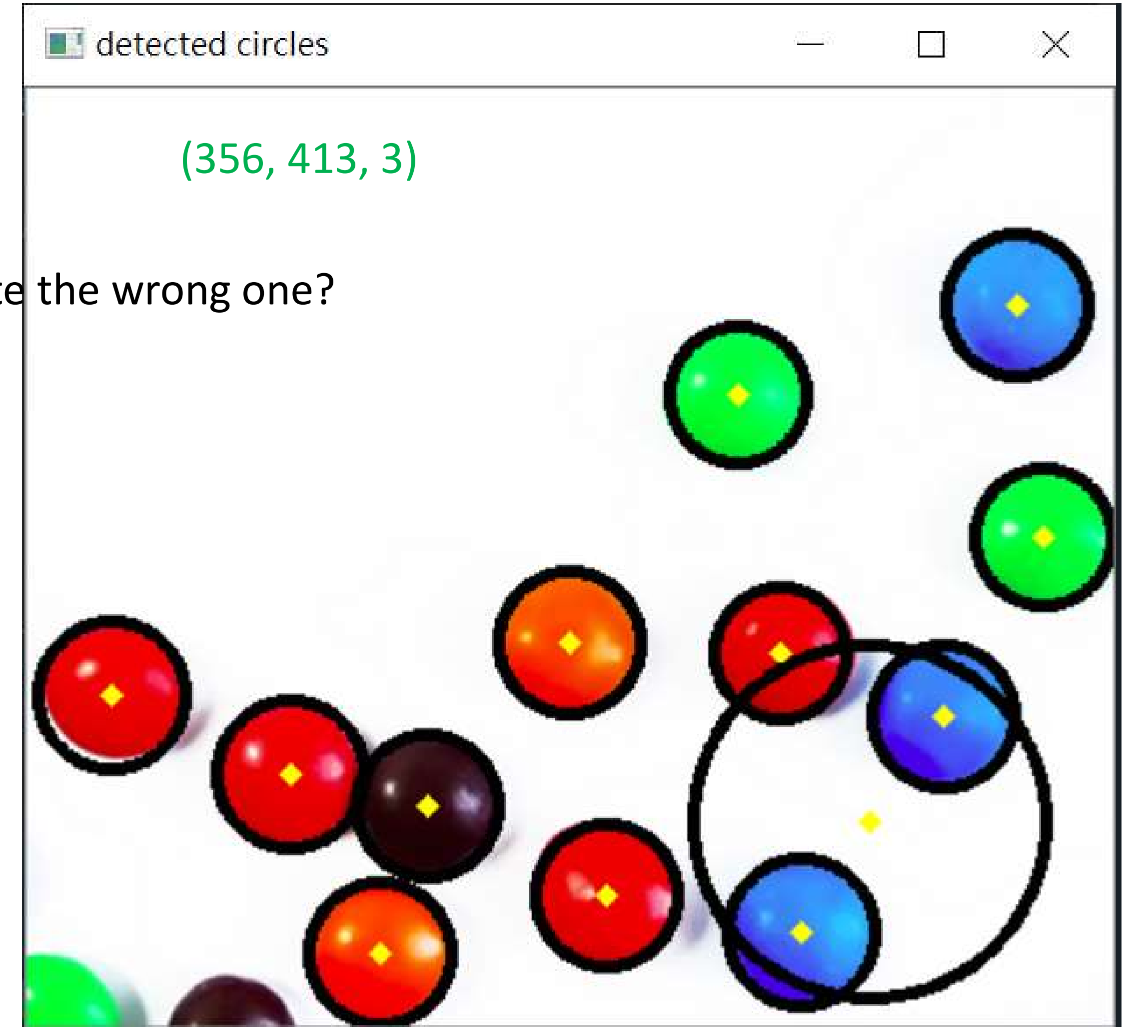
x, y, r

13 circles

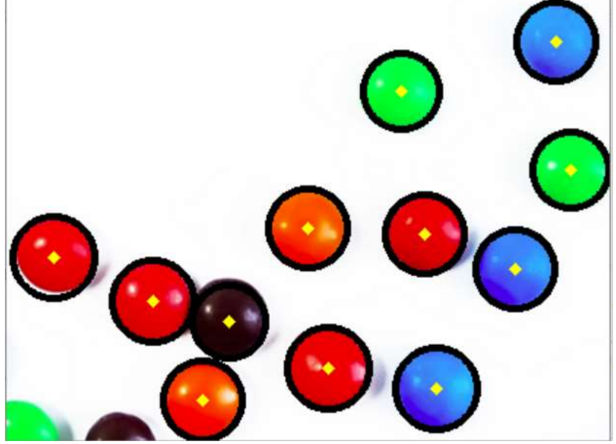
accuracy?

(12/13 accuracy)

Q: how to eliminate the wrong one?



12 circles



35 (36). Face (+Eyes) Detection using Haar Cascade Classifiers

匈牙利數學家Alfred Haar

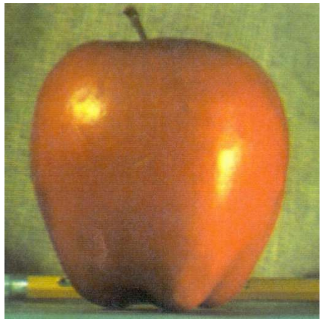
https://github.com/opencv/opencv/tree/master/data/haarcascades to download

haarcascade_frontalface_alt.xml (pretrained face detector)
haarcascade_eye_tree_eyeglasses.xml (pretrained eye detector)

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 (image of face) and negative images (image without face).

positive

negative



Haar feature-based cascade classifiers

Haar feature-based cascade classifiers

每一個基本的Haar-like特徵就是一個弱分類器，Cascade Classifier透過數個強分類器的組合（級聯），將每一張輸入的圖片依次通各個強分類器，前面的強分類器相對簡單，其包含的弱分類器也較少，後面的強分類器逐級複雜，只有通過前面的強分類檢測後的圖片才能進入後面的強分類器進行檢測，所以前面幾級的分類器已經過濾掉大部分不合格的圖片，只有通過了所有強分類器檢測的才是正確辨識出的圖片。

https://chtseng.wordpress.com/2018/06/15/opencv-cascade-object-detection/

(pretrained model)

```
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_alt.xml')

face_cascade.detectMultiScale

faces = face_cascade.detectMultiScale(image[, scaleFactor[, minNeighbors[, flags[, minSize[, maxSize]]]]])
```

Parameters

image	Matrix of the type CV_8U containing an image where objects are detected.
objects	Vector of rectangles where each rectangle contains the detected object, the rectangles may be partially outside the original image. (n, 4): x, y, w, h
scaleFactor	Parameter specifying how much the image size is reduced at each image scale.
minNeighbors	Parameter specifying how many neighbors each candidate rectangle should have to retain it.
flags	Parameter with the same meaning for an old cascade as in the function cvHaarDetectObjects. It is not used for a new cascade.
minSize	Minimum possible object size. Objects smaller than that are ignored. (w, h)
maxSize	Maximum possible object size. Objects larger than that are ignored. If `maxSize == minSize` model is evaluated on single scale.

objects = cv.CascadeClassifier.detectMultiScale(image, scaleFactor, minNeighbors)

image Matrix of the type CV_8U containing an image where objects are detected.
objects Vector of rectangles where each rectangle contains the detected object, the rectangles may be partially outside the original image.
scaleFactor Parameter specifying how much the image size is reduced at each image scale.
minNeighbors Parameter specifying how many neighbors each candidate rectangle should have to retain it.

ex41.py

faces.shape
Out[20]: (1, 4)

x, y, w, h
array([[243, 123, 247, 247]], dtype=int32)

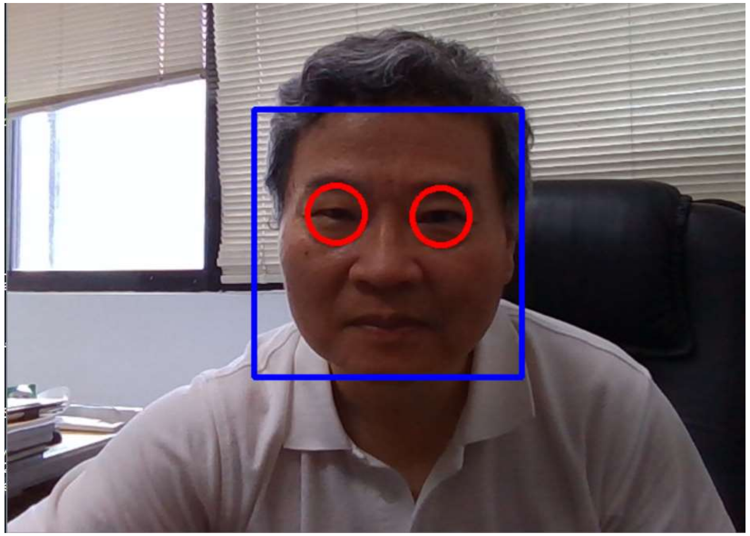
```python
import cv2

face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_alt.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye_tree_eyeglasses.xml')
# Read the input image
#img = cv2.imread('test.png')
cap = cv2.VideoCapture(0) #'test.mp4')

while cap.isOpened():
    _, img = cap.read()

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.1, 3) # default (1.1, 3)

    for (x, y , w ,h) in faces:
        cv2.rectangle(img, (x,y), (x+w, y+h), (255, 0 , 0), 3) # blue
```

ex41.py

eyes.shape
Out[32]: (2, 4)

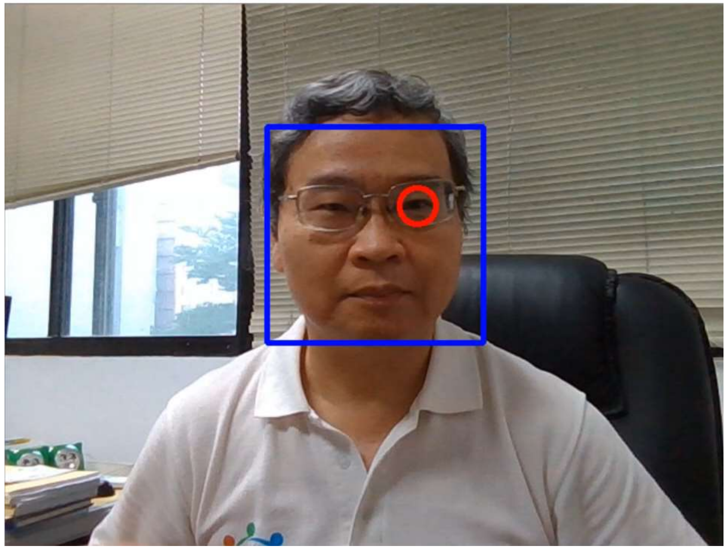
array([[45, 57, 55, 55],
       [149, 60, 55, 55]], dtype=int32)

```python
        faceROI = gray[y:y+h,x:x+w]
        #-- In each face, detect eyes
        eyes = eye_cascade.detectMultiScale(faceROI, 1.1, 3)
        for (x2,y2,w2,h2) in eyes:
            eye_center = (x + x2 + w2//2, y + y2 + h2//2)
            radius = int(round((w2 + h2)*0.25))
            frame = cv2.circle(img, eye_center, radius, (0, 0, 255 ), 3) #red

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

cap.release()
cv2.destroyAllWindows()
```

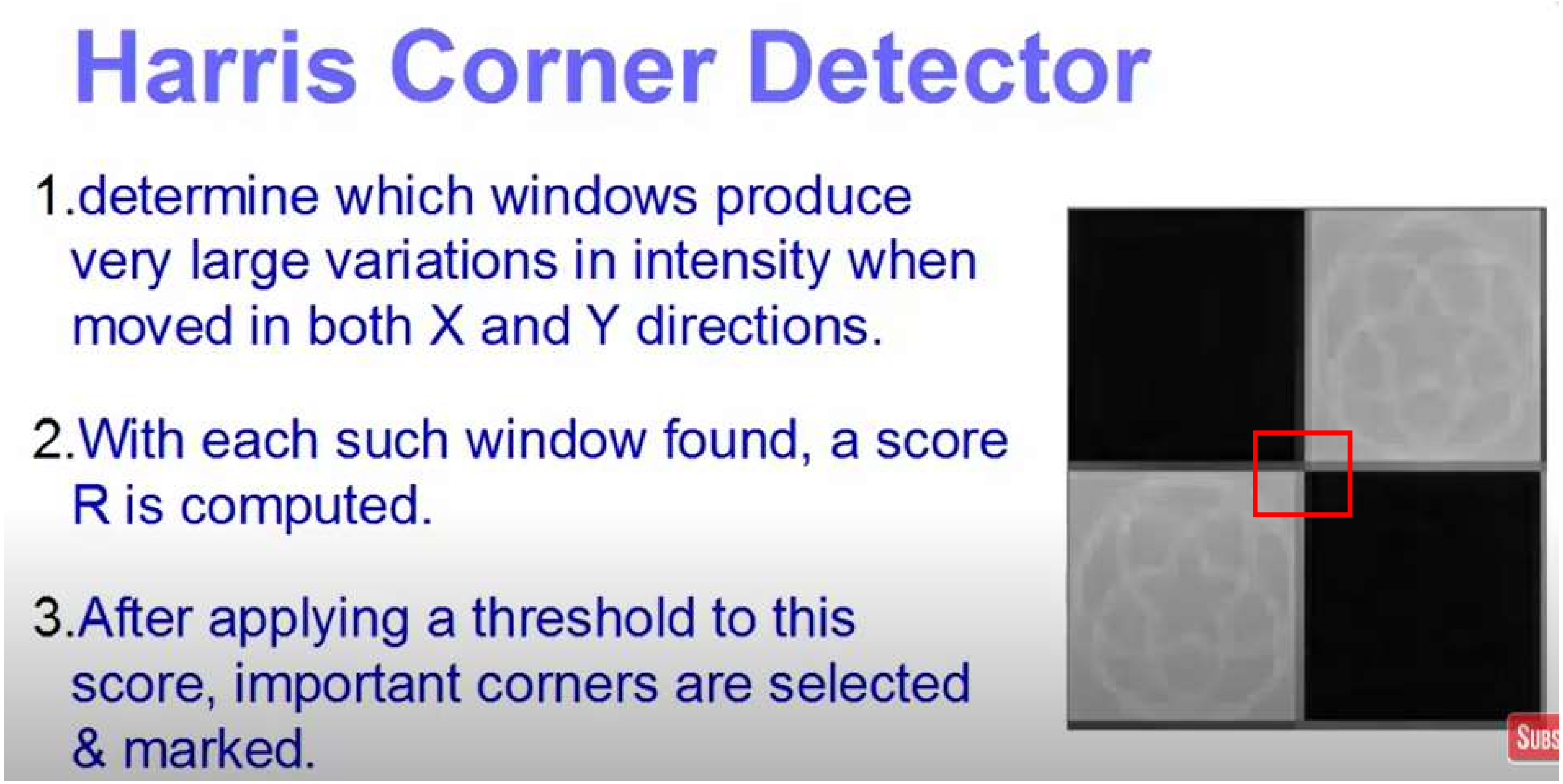


faces.shape
Out[20]: (1, 4)

eyes.shape
Out[32]: (2, 4)



37. Detect Corners with Harris Corner Detector



Chris Harris and Mike Stephens (1988). "A Combined Corner and Edge Detector". *Alvey Vision Conference*. **15**.

Harris Corner Detector

1. determine which windows produce very large variations in intensity when moved in both X and Y directions.

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

$$E(u,v) \approx [\,u \quad v\,]\, M \begin{bmatrix} u \\ v \end{bmatrix}$$

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$$

Harris Corner Detector

2. With each such window found, a score R is computed.

$$R = det(M) - k(trace(M))^2$$

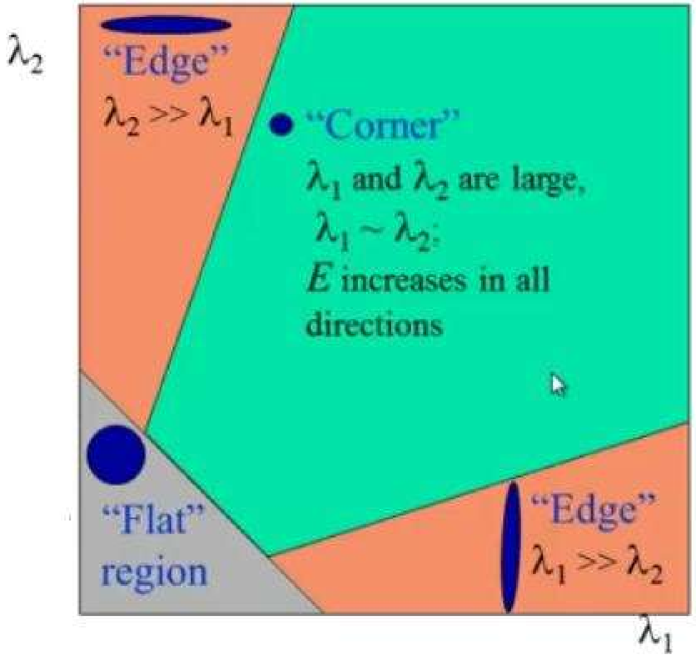
where $$R = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2$$

- $det(M) = \lambda_1 \lambda_2$
- $trace(M) = \lambda_1 + \lambda_2$
- λ_1 and λ_2 are the eigen values of M

Harris Corner Detector

3. After applying a threshold to this score, important corners are selected & marked.

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



```
dst = cv.cornerHarris(img, 2, 3, 0.04)
```

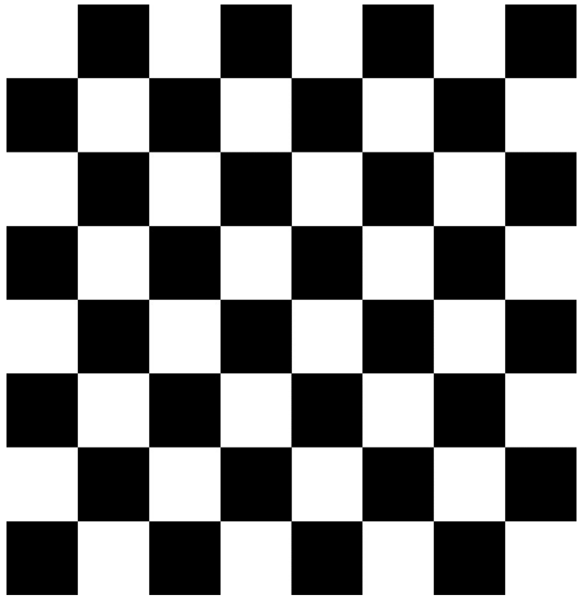
img - Input image, it should be grayscale and float32 type.
blockSize - It is the size of neighbourhood considered for corner detection
ksize - Aperture parameter of Sobel derivative used.
k - Harris detector free parameter in the equation.

```
dst=cv2.dilate(src, kernel[, dst[, anchor[, iterations[, borderType[, borderValue]]]]])
    cv2.dilate(img,kernel,iterations = 1)
```

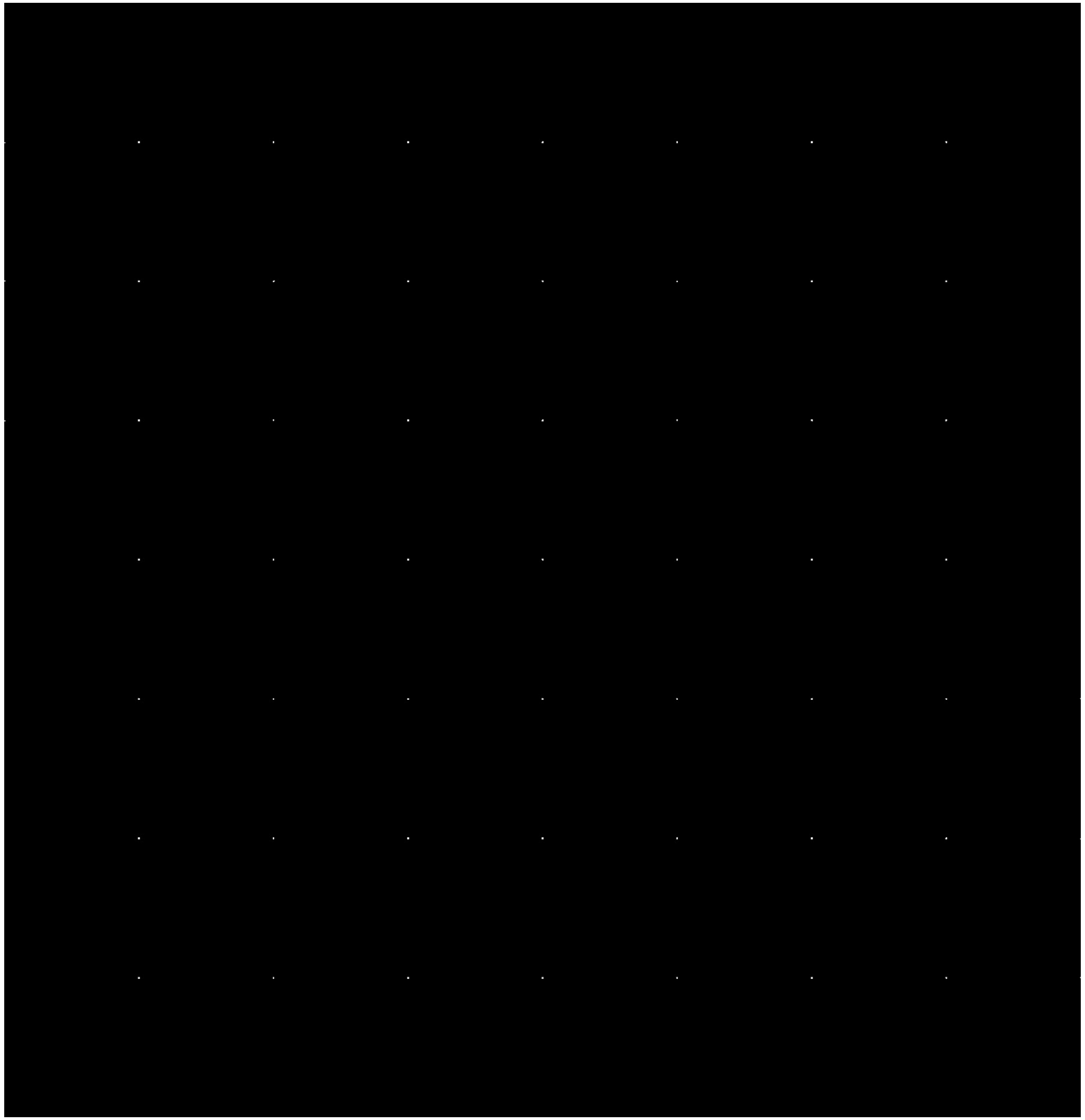
```python
import numpy as np
import cv2 as cv

img = cv.imread('blox.jpg') #chessboard.png') # (3723, 3595, 3)
cv.imshow('img', img)

gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
gray = np.float32(gray)

dst1 = cv.cornerHarris(gray, 2, 3, 0.04) # (3723, 3595)
#result is dilated for marking the corners, not important
dst = cv.dilate(dst1, None)

cv.imshow('dst', dst)

img[dst > 0.01 * dst.max()] = [0, 0, 255] # red

cv.imshow('corner', img)

if cv.waitKey(0) & 0xff == 27:
    cv.destroyAllWindows()

# cv.imwrite('corner.png',img)
```

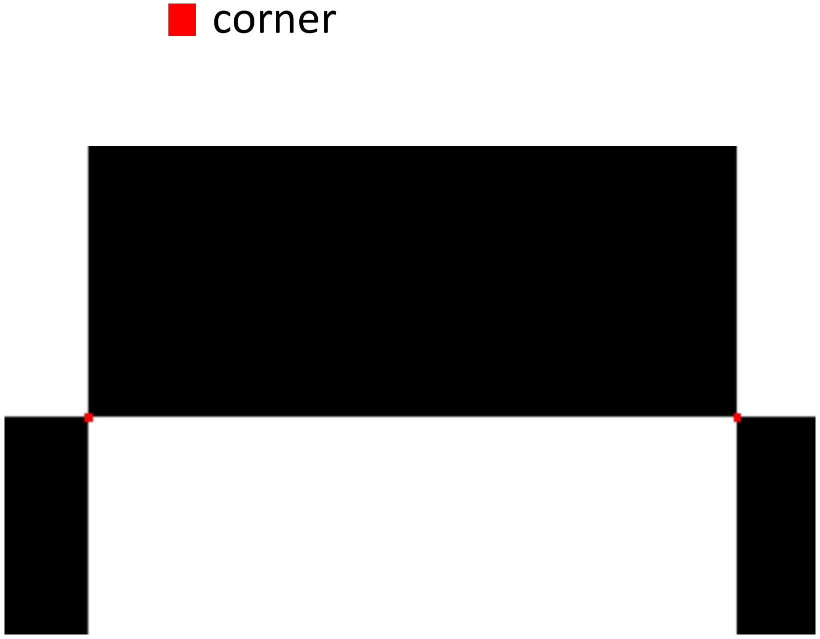
ex42.py

chessboard.png

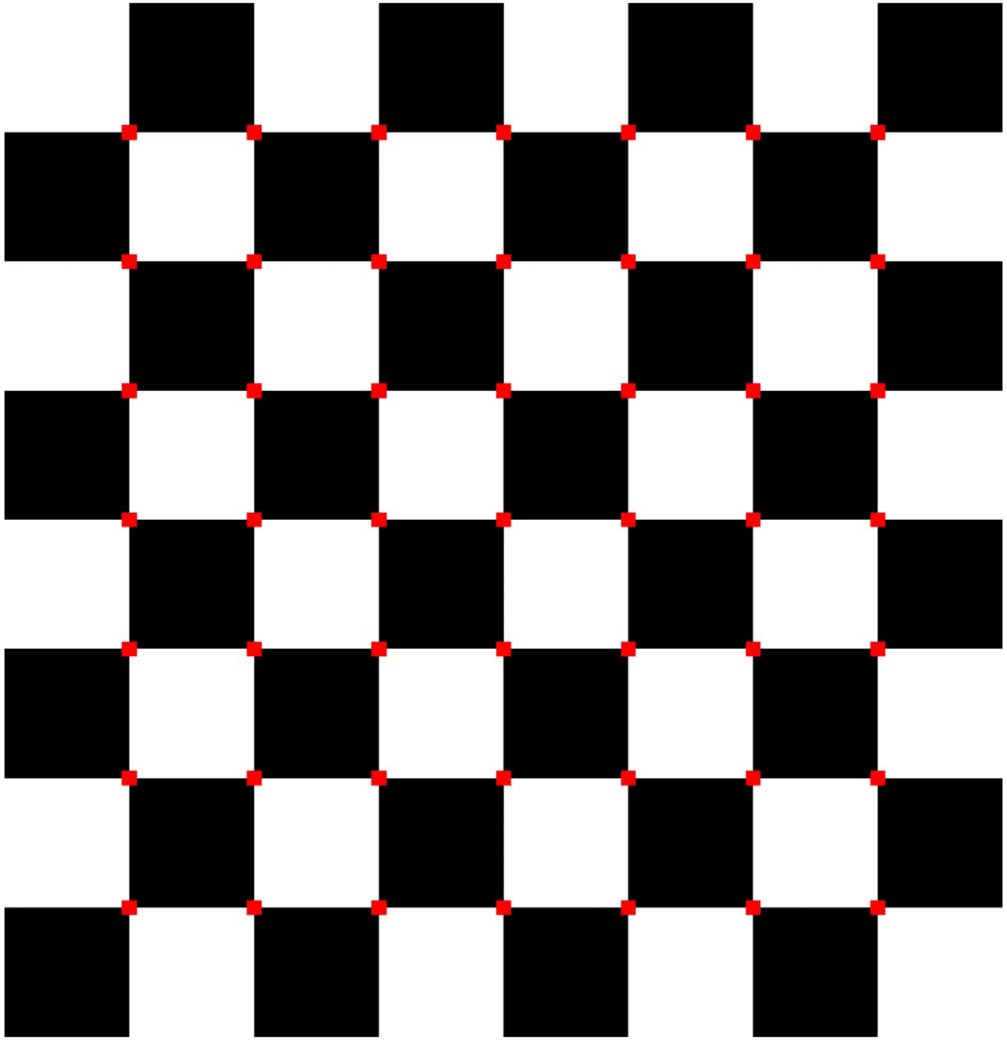


cv2.cornerHarris





cv.cornerHarris(gray, 50, 3, 0.04)



38. Detect Corners with Shi Tomasi Corner Detector

J. Shi and C. Tomasi. Good Features to Track,. 9th IEEE Conference on Computer Vision and Pattern Recognition. Springer. June 1994.

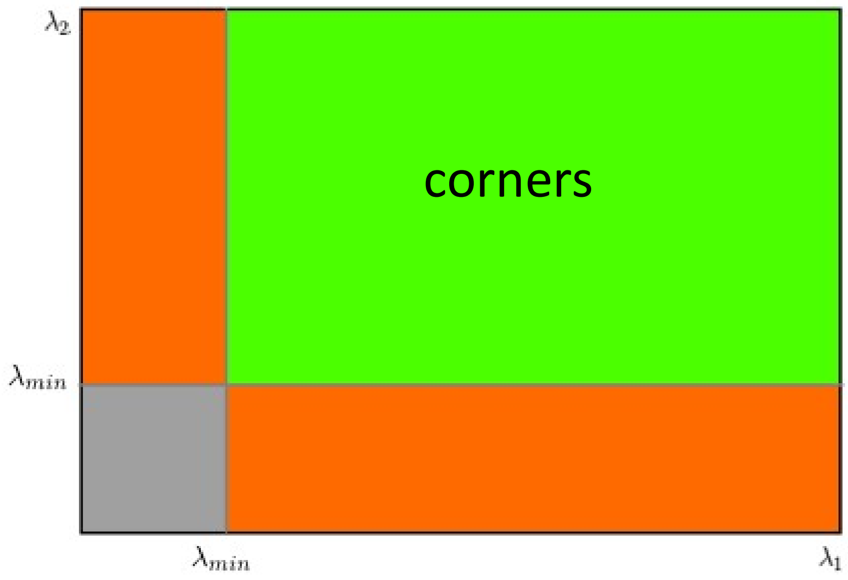
The scoring function in Harris Corner Detector was given by:

$$R = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2$$

Instead of this, Shi-Tomasi proposed:

$$R = min(\lambda_1, \lambda_2)$$

$R_{threshold}$
$= \lambda_{min}$



corners = cv2.goodFeaturesToTrack(gray, 100, 0.01, 10)

cv2.goodFeaturesToTrack().

It finds N strongest corners in the image by Shi-Tomasi method (or Harris Corner Detection, if you specify it).

As usual, image should be a grayscale image. gray

Then you specify number of corners you want to find. 100

Then you specify the quality level, which is a value between 0-1, which denotes the minimum quality of corner below which everyone is rejected. 0.01

Then we provide the minimum euclidean distance between corners detected. 10

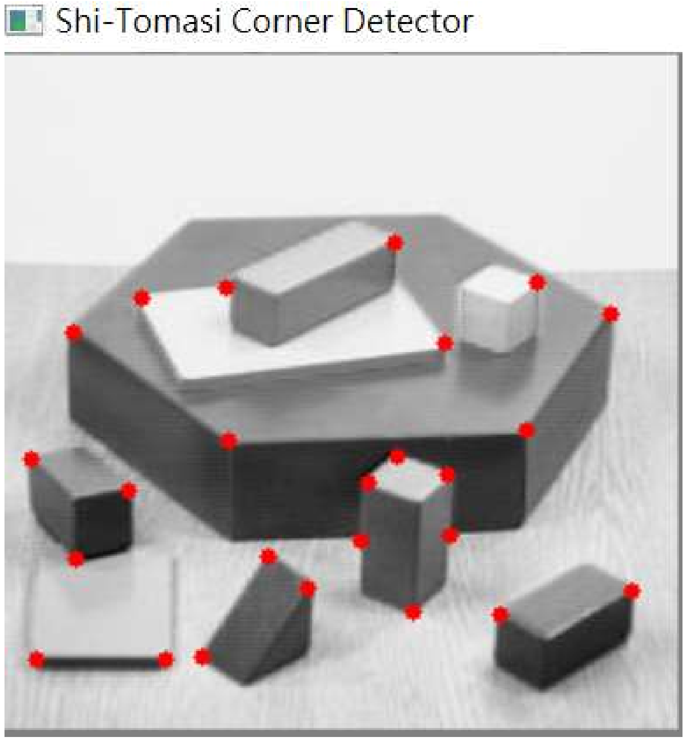
numpy.int0 這個其實就是numpy.int64

ex43.py

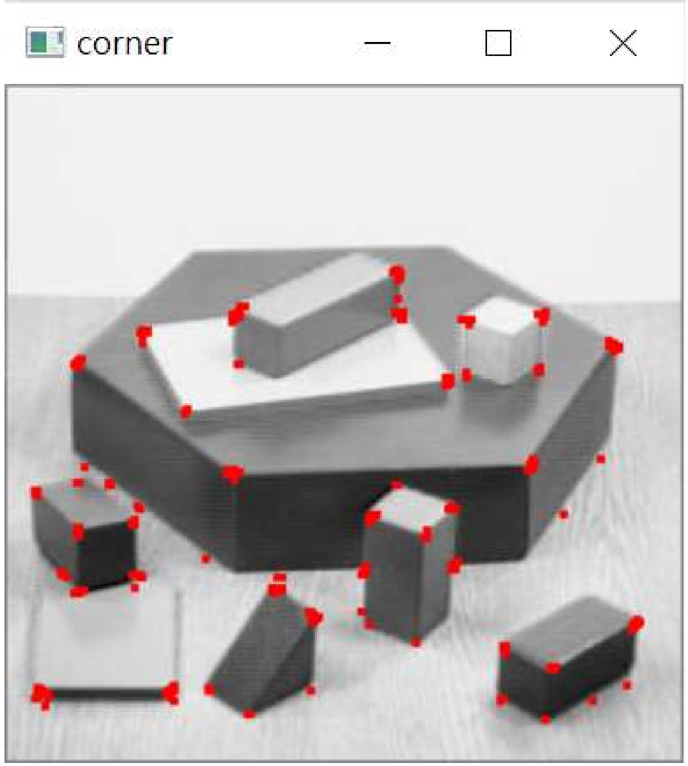
```python
import numpy as np
import cv2 as cv

img = cv.imread('blox.jpg')

gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

corners = cv.goodFeaturesToTrack(gray, 25, 0.01, 10)

corners = np.int0(corners)  # (25, 1, 2)

for i in corners:       # i.shape = (1,2)
    x, y = i.ravel()    # i.ravel().shape = (2,)
    cv.circle(img, (x, y), 3, [0, 0, 255], -1) # red

cv.imshow('Shi-Tomasi Corner Detector', img)

if cv.waitKey(0) & 0xff == 27:
    cv.destroyAllWindows()
```

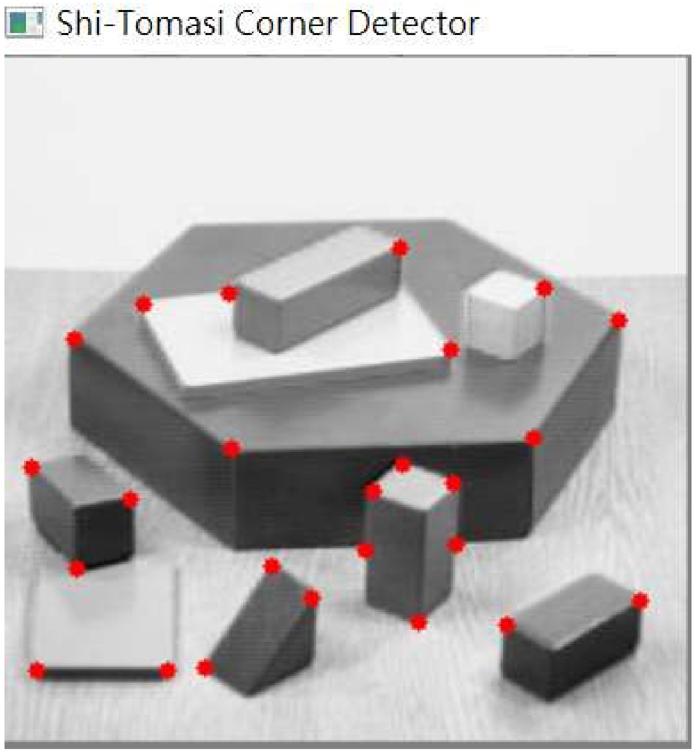
i= [[155 211]]

i.ravel() = [155 211]



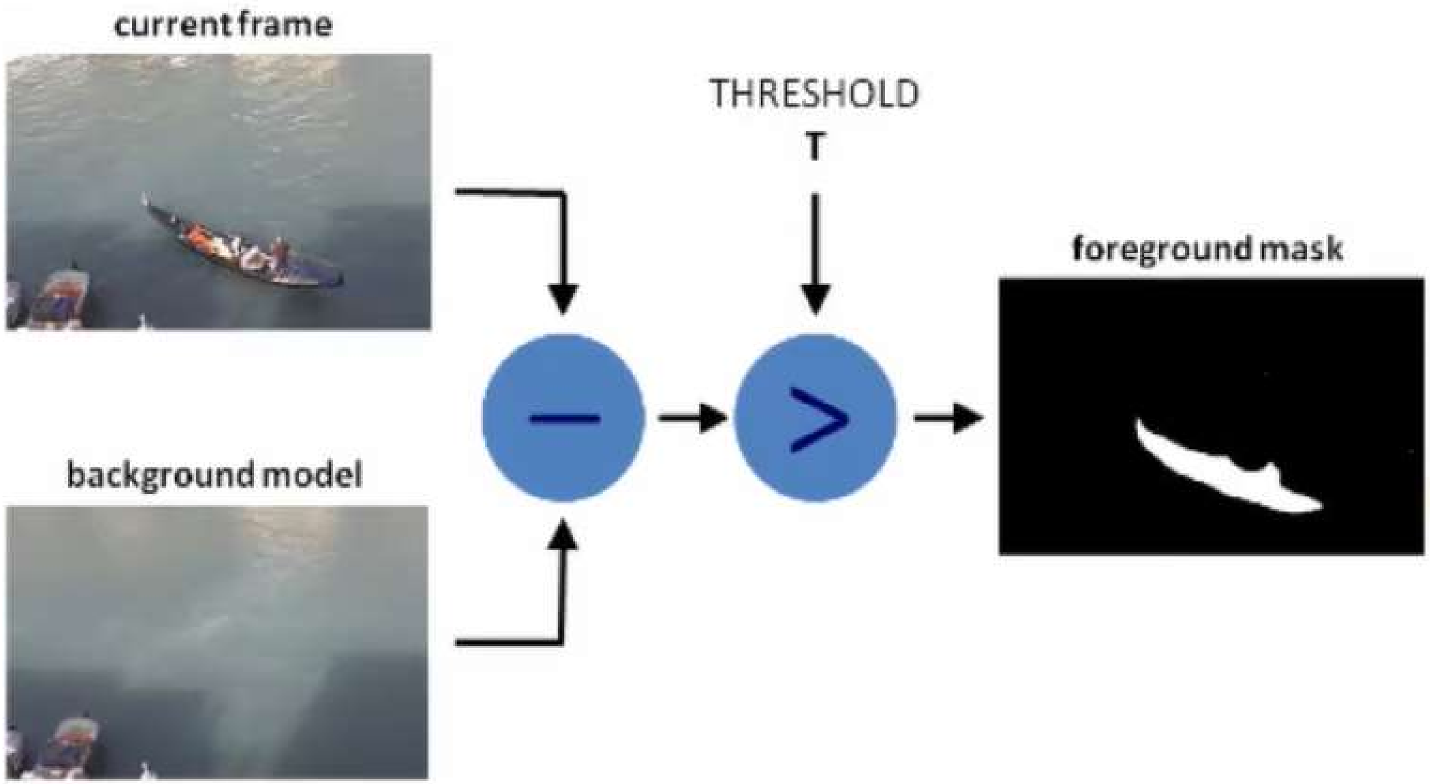
Harris





39. How to Use Background Subtraction Methods

To detect moving objects in a static scene.



Optical flow ?

1. Gaussian Mixture-based Background/Foreground Segmentation Algorithm

retval=cv.createBackgroundSubtractorMOG2([, history[, varThreshold[, detectShadows]]])

500 16, True

Parameters

history Length of the history.

varThreshold Threshold on the squared Mahalanobis distance between the pixel and the model to decide whether a pixel is well described by the background model. This parameter does not affect the background update.

detectShadows If true, the algorithm will detect shadows and mark them. It decreases the speed a bit, so if you do not need this feature, set the parameter to false. default = True

2. K-nearest neighbours - based Background/Foreground Segmentation Algorithm

retval=cv.createBackgroundSubtractorKNN([, history[, dist2Threshold[, detectShadows]]])

500 400.0, True

Parameters

history Length of the history.

dist2Threshold Threshold on the squared distance between the pixel and the sample to decide whether a pixel is close to that sample. This parameter does not affect the background update.

detectShadows If true, the algorithm will detect shadows and mark them. It decreases the speed a bit, so if you do not need this feature, set the parameter to false. default = True

Member functions:

createBackgroundSubtractorMOG2.apply

Computes a foreground mask

cv.BackgroundSubtractorMOG2.apply(image[, fgmask[, learningRate]])

-1

image Next video frame. Floating point frame will be used without scaling and should be in range [0,255].

fgmask The output foreground mask as an 8-bit binary image.

learningRate The value between 0 and 1 that indicates how fast the background model is learnt.
Negative parameter value makes the algorithm to use some automatically chosen learning rate.
0 means that the background model is not updated at all,
1 means that the background model is completely reinitialized from the last frame.

createBackgroundSubtractorMOG2.getHistory()

Returns the number of last frames that affect the background model.

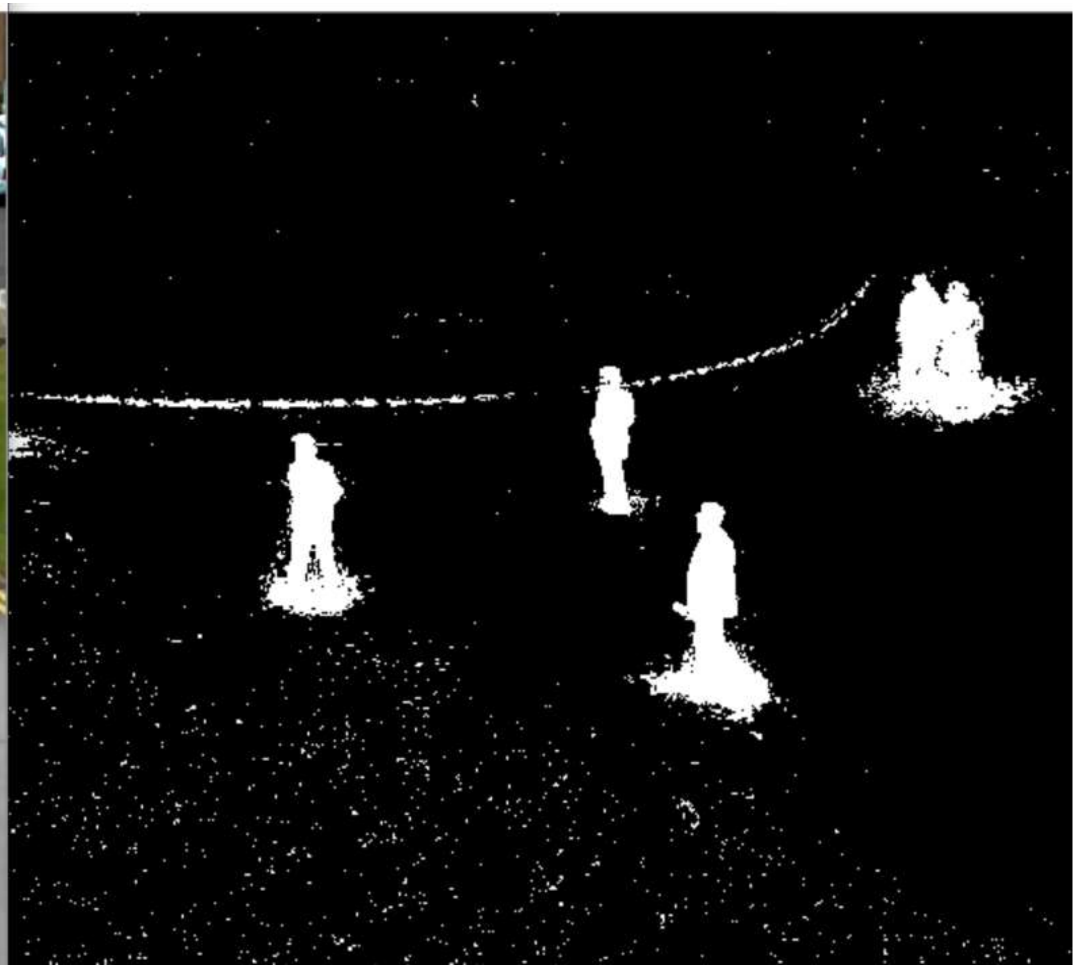
….

….

ex44.py

```python
import cv2 as cv

cap = cv.VideoCapture('vtest.avi')
#cap = cv.VideoCapture(0)

fgbg = cv.createBackgroundSubtractorMOG2(detectShadows=False) #shadow shown in gray
#fgbg = cv.createBackgroundSubtractorKNN(detectShadows=False)

while True:
    ret, frame = cap.read()
    if frame is None:
        break

    fgmask = fgbg.apply(frame) #The output foreground mask as an 8-bit binary image

    cv.imshow('Frame', frame)
    cv.imshow('FG MASK Frame', fgmask)

    keyboard = cv.waitKey(30)
    if keyboard == 113 or keyboard == 27:  # 'q' or 'esc'
        print(fgbg.getHistory())
        break
cap.release()
cv.destroyAllWindows()
```

No shadows



Try detectShadows and cap = cv.VideoCapture(0)

40. Mean Shift Object Tracking

Video used : https://www.bogotobogo.com/python/OpenCV_Python/images/mean_shift_tracking/slow_traffic_small.mp4

In OpenCV, for HSV color space
hue range is [0,179], saturation range is [0,255], and value range is [0,255].

Object Tracking using color thresholding (顏色二值化)

ex45.py

```
green = np.uint8([[[0,255,0 ]]]) #(1,1,3)
hsv_green = cv.cvtColor(green,cv.COLOR_BGR2HSV)
print( hsv_green )
# [[[ 60 255 255]]]

lower_green = np.array([50,100,100])
upper_green = np.array([70,255,255])
                        h   s   v
```

```python
import numpy as np
import cv2 as cv
lower_green = np.array([50,100,100])
upper_green = np.array([70,255,255])

cap = cv.VideoCapture('slow_traffic_small.mp4')

while(1):
    # Take each frame
    ret, frame = cap.read()
    if ret == True:
      # Convert BGR to HSV
      hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)
      # Threshold the HSV image to get only gren colors
      mask = cv.inRange(hsv, lower_green, upper_green)
      # Bitwise-AND mask and original image
      res = cv.bitwise_and(frame,frame, mask= mask)
      cv.imshow('frame',frame)
      cv.imshow('mask',mask)
      cv.imshow('res',res)
      k = cv.waitKey(30) & 0xff
      if k == 27:       # esc
          break
    else:
       break

cap.release()
cv.destroyAllWindows()
```

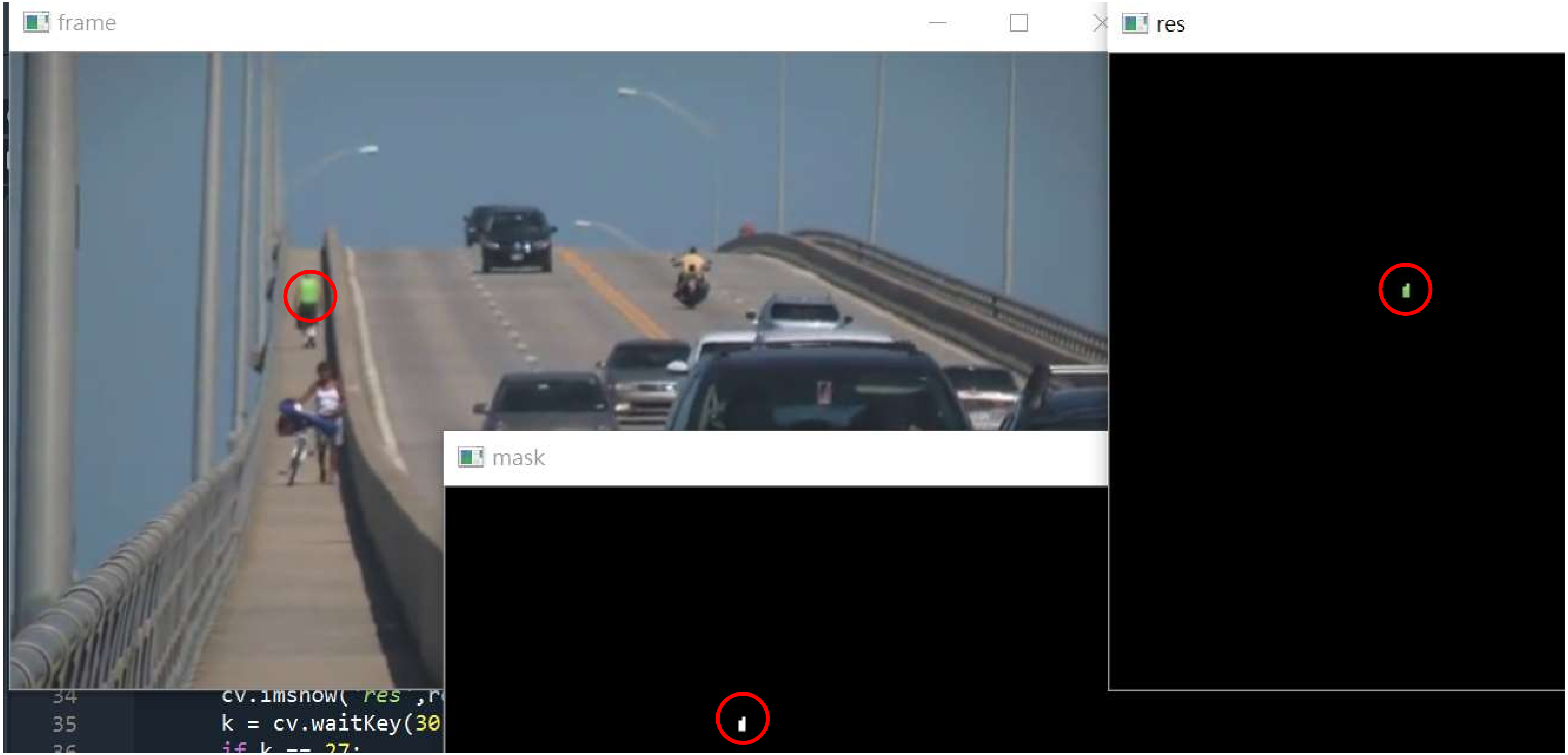
ex45.py

dst=cv.inRange(src, lowerb, upperb[, dst])

Checks if array elements lie between the elements of two other arrays.

dst (I) is set to 255 (all 1 -bits) if src (I) is within the specified 1D, 2D, 3D, ... box and 0 otherwise.

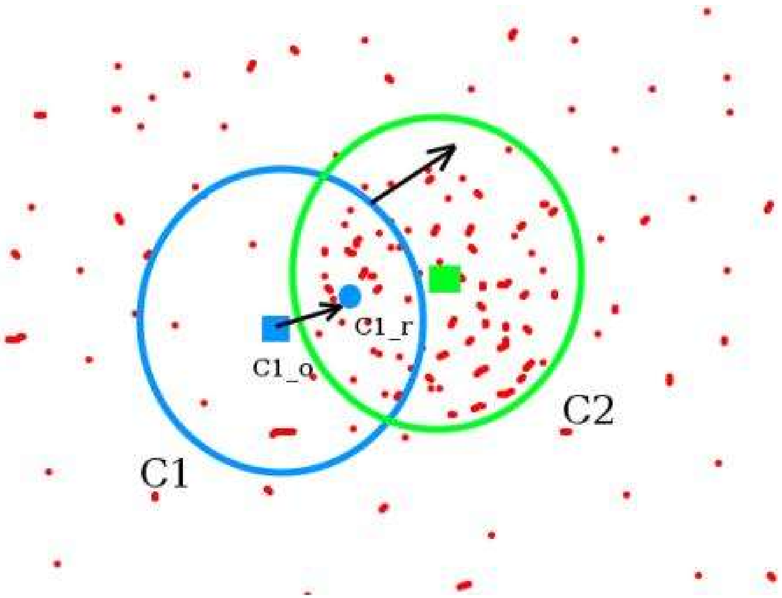
255 (decimal): 1 1 1 1 1 1 1 1 (binary)


frame
res
mask
35 k = cv.waitKey(30

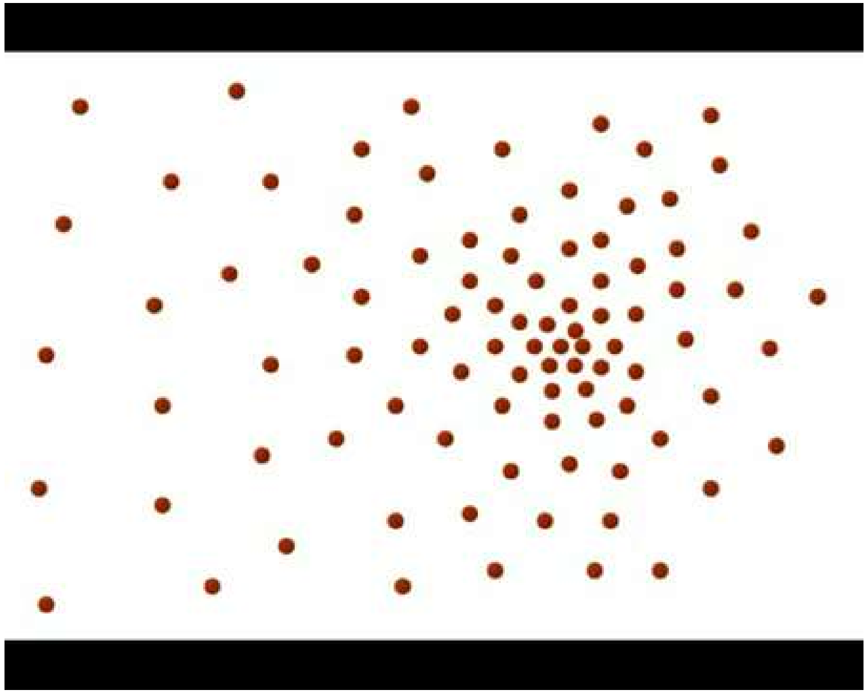
40. Mean Shift Object Tracking

https://docs.opencv.org/3.4/d7/d00/tutorial_meanshift.html

We will learn about the Meanshift and Camshift algorithms to track objects in videos.



Meanshift




ROI
final_image

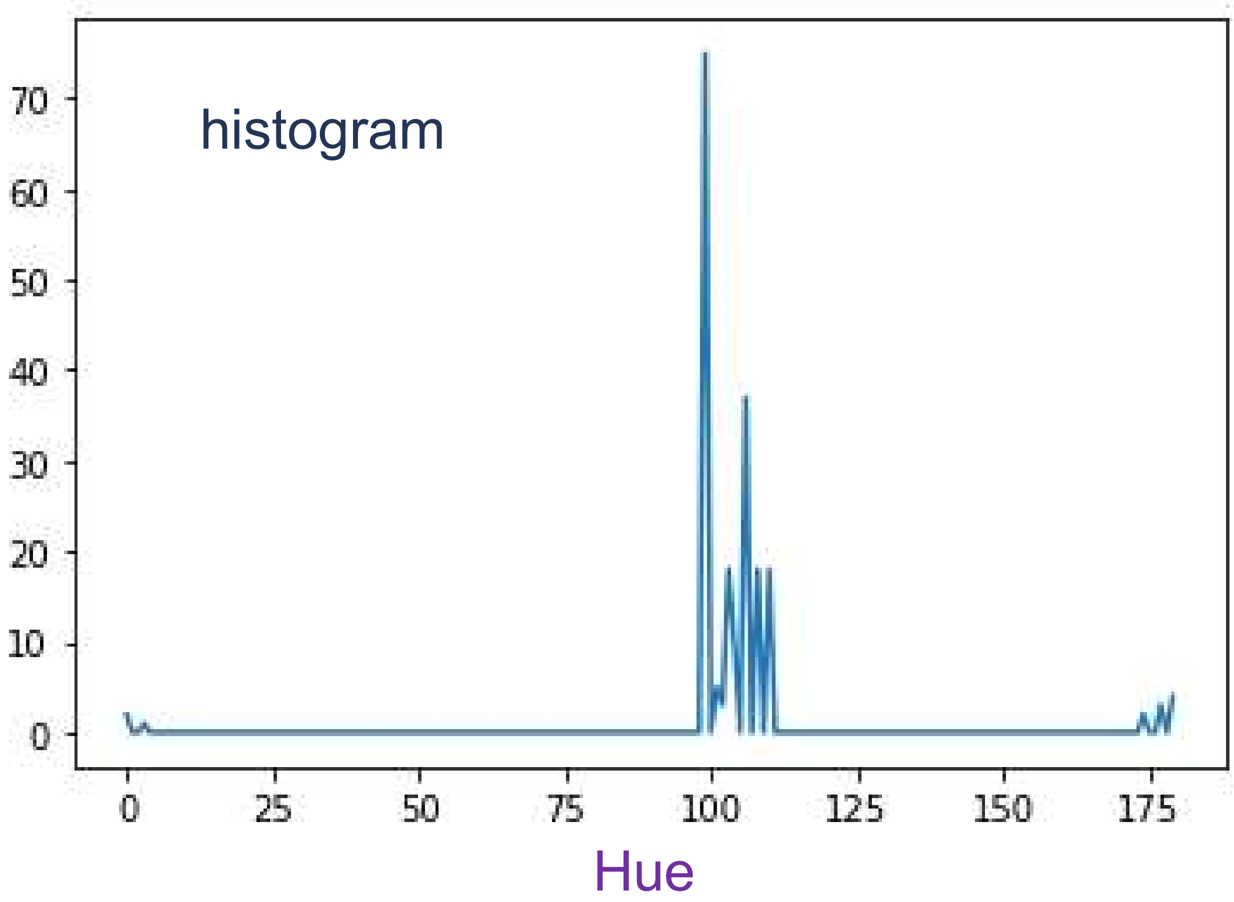
To use meanshift in OpenCV:
first we need to setup the target (ROI),
find its histogram so that we can backproject the target on each frame for calculation of meanshift.
We also need to provide an initial location of window.
For histogram, only Hue is considered here. Also, to avoid false values due to low light, low light values are discarded using **cv.inRange()** function.

channel

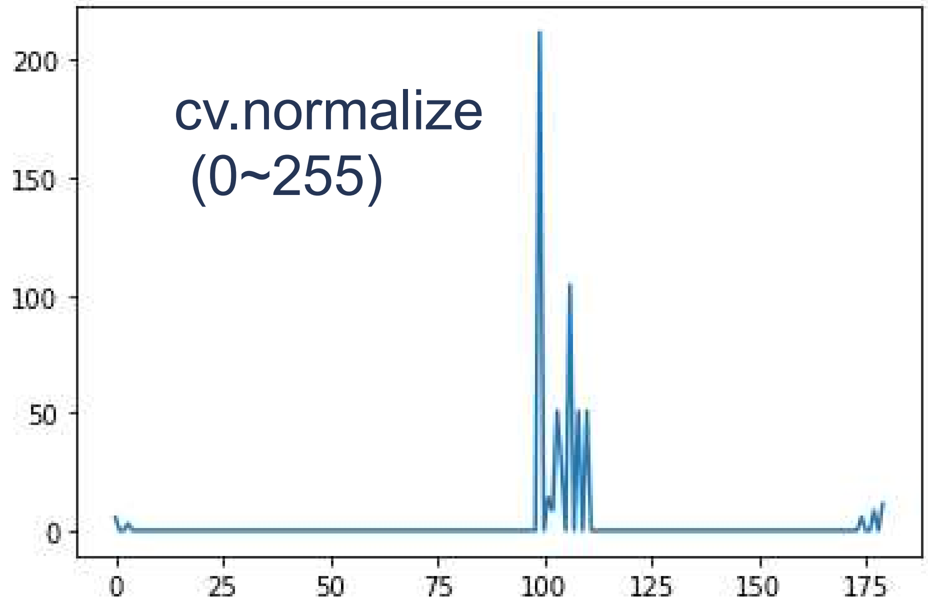
```
cv2.calcHist([src], [0], None, [histSize], [histRange])
    roi_hist = cv.calcHist([hsv_roi], [0], mask, [180], [0, 180]) # shape (180,1)
```

```
dst=cv.normalize(src, dst[, alpha[, beta[, norm_type[, dtype[, mask]]]]])
    cv.normalize(roi_hist, roi_hist, 0, 255, cv.NORM_MINMAX)
```

alpha, beta : the lower and upper range boundary in case of the range normalization



ROI (shape (50, 100))



The class defining termination criteria for iterative algorithms

cv.TERM_CRITERIA_EPS - stop the algorithm iteration if specified accuracy, *epsilon*, is reached.

cv.TERM_CRITERIA_COUNT - the maximum number of iterations at which the iterative algorithm stops

```
term_crit = ( cv.TERM_CRITERIA_EPS | cv.TERM_CRITERIA_COUNT, 10, 1) # tuple (3,10,1)
```

termination criteria, either 10 iterations or move by at least 1 pt

Back Projection:

- Back Projection is a way of recording how well the pixels of a given image fit the distribution of pixels in a histogram model.
- To make it simpler: For Back Projection, you calculate the histogram model of a feature and then use it to find this feature in an image.

```
dst=cv.calcBackProject(images, channels, hist, ranges, scale[, dst])
```

Let's say you have gotten a skin histogram (Hue-Saturation) based on the image below. The histogram besides is going to be our *model histogram* (which we know represents a sample of skin tonality). You applied some mask to capture only the histogram of the skin area:

Now, let's imagine that you get another hand image (Test Image) like the one below: (with its respective histogram)

- What we want to do is to use our *model histogram* (that we know represents a skin tonality) to detect skin areas in our Test Image. Here are the steps
    1. In each pixel of our Test Image (i.e. $p(i, j)$), collect the data and find the correspondent bin location for that pixel (i.e. $(h_{i,j}, s_{i,j})$).
    2. Lookup the *model histogram* in the correspondent bin - $(h_{i,j}, s_{i,j})$ - and read the bin value.
    3. Store this bin value in a new image (*BackProjection*). Also, you may consider to normalize the *model histogram* first, so the output for the Test Image can be visible for you.
    4. Applying the steps above, we get the following BackProjection image for our Test Image:

    5. In terms of statistics, the values stored in *BackProjection* represent the *probability* that a pixel in *Test Image* belongs to a skin area, based on the *model histogram* that we use. For instance in our Test image, the brighter areas are more probable to be skin area (as they actually are), whereas the darker areas have less probability (notice that these "dark" areas belong to surfaces that have some shadow on it, which in turns affects the detection).

dst=cv.calcBackProject(images, channels, hist, ranges, scale[, dst])

images	Source arrays. They all should have the same depth, CV_8U, CV_16U or CV_32F , and the same size. Each of them can have an arbitrary number of channels.
nimages	Number of source images.
channels	The list of channels used to compute the back projection. The number of channels must match the histogram dimensionality. The first array channels are numerated from 0 to images[0].channels()-1 , the second array channels are counted from images[0].channels() to images[0].channels() + images[1].channels()-1, and so on.
hist	Input histogram that can be dense or sparse.
backProject	Destination back projection array that is a single-channel array of the same size and depth as images[0] .
ranges	Array of arrays of the histogram bin boundaries in each dimension. See calcHist .
scale	Optional scale factor for the output back projection.
uniform	Flag indicating whether the histogram is uniform or not (see above).

```
dst = cv.calcBackProject([hsv], [0], roi_hist, [0, 180], 1)
```

```
retval, window = cv.meanShift(probImage, window, criteria)
```

probImage Back projection of the object histogram. See calcBackProject for details.

window Initial search window.

criteria Stop criteria for the iterative search algorithm. returns : Number of iterations CAMSHIFT took to converge. The function implements the iterative object search algorithm. It takes the input back projection of an object and the initial position. The mass center in window of the back projection image is computed and the search window center shifts to the mass center. The procedure is repeated until the specified number of iterations criteria.maxCount is done or until the window center shifts by less than criteria.epsilon. The algorithm is used inside CamShift and, unlike CamShift , the search window size or orientation do not change during the search. You can simply pass the output of calcBackProject to this function. But better results can be obtained if you pre-filter the back projection and remove the noise. For example, you can do this by retrieving connected components with findContours , throwing away contours with small area (contourArea), and rendering the remaining contours with drawContours.

ex46.py

```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt

cap = cv.VideoCapture('slow_traffic_small.mp4')
# take first frame of the video
ret, frame = cap.read()

# setup initial location of window
x, y, width, height = 300, 200, 100, 50
track_window = (x, y ,width, height)
# set up the ROI for tracking
roi = frame[y:y+height, x : x+width]
hsv_roi = cv.cvtColor(roi, cv.COLOR_BGR2HSV)

mask = cv.inRange(hsv_roi, np.array((0., 60., 32.)), np.array((180., 255., 255)))
# mask.shape = (50, 100)
roi_hist = cv.calcHist([hsv_roi], [0], mask, [180], [0, 180]) # shape (180,1)
plt.plot(roi_hist)
cv.normalize(roi_hist, roi_hist, 0, 255, cv.NORM_MINMAX)

# Setup the termination criteria, either 10 iteration or move by atleast 1 pt
term_crit = ( cv.TERM_CRITERIA_EPS | cv.TERM_CRITERIA_COUNT, 10, 1) # tuple (3,10,1)
cv.imshow('roi',roi)
```

ex46.py

```python
while(1):
    ret, frame = cap.read()
    if ret == True:

        hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)
        dst = cv.calcBackProject([hsv], [0], roi_hist, [0, 180], 1) #(360, 640)
        # apply meanshift to get the new location
        ret, track_window = cv.meanShift(dst, track_window, term_crit)
        # Draw it on image
        x,y,w,h = track_window
        final_image = cv.rectangle(frame, (x,y), (x+w, y+h), 255, 3)

        cv.imshow('dst', dst)
        cv.imshow('final_image',final_image)
        k = cv.waitKey(30) & 0xff
        if k == 27:    # esc
            break
    else:
        break

cap.release()
cv.destroyAllWindows()
```

dst

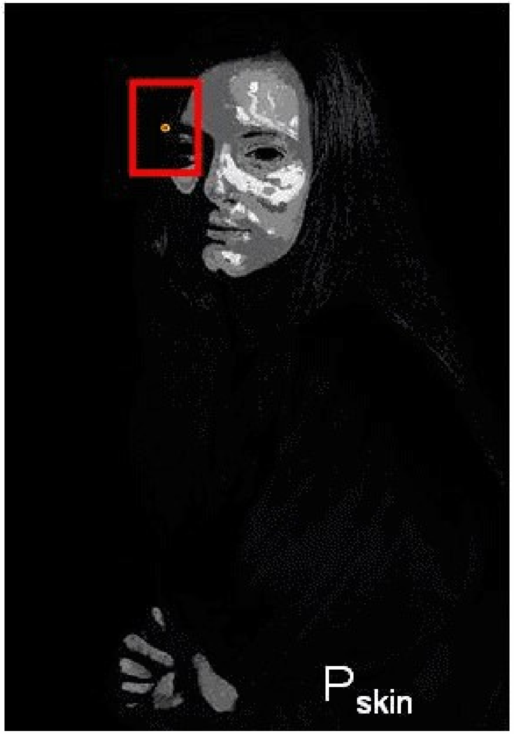
41. Object Tracking Camshift Method

CAMshift (Continuously Adaptive Meanshift) published by Gary Bradsky in his paper "Computer Vision Face Tracking for Use in a Perceptual User Interface" in 1998.

The meanshift problem is that the window always has the same size whether the car is very far or very close to the camera. That is not good. We need to adapt the window size with size and rotation of the target.

It applies meanshift first. Once meanshift converges, it updates the size of the window as, $s = 2 \times \sqrt{\frac{M_{00}}{256}}$.

It also calculates the orientation of the best fitting ellipse to it. Again it applies the meanshift with new scaled search window and previous window location. The process continues until the required accuracy is met.



Mean shift window initialization

```
ret, track_window = cv.CamShift(dst, track_window, term_crit) # ret= ((x, y), (w, h), θ )

pts = cv.boxPoints(ret)  #查找旋轉矩形的4個頂點, shape=(4,2)
```

((415.0, 226.5), (52.80326461791992, 166.11111450195312), 87.1136474609375)
(331, 199, 168, 55)
[[330.72034 204.31413]
 [496.62073 195.9496]
 [499.27966 248.68587]
 [333.37927 257.0504]]

(center (x,y), (width, height), angle of rotation)

非填充多邊形：polylines()

```
# cv2.polylines(img, [pts], isClosed, color[, thickness[, lineType[, shift]]])
# img – 要畫的圖片
# pts – 多邊形的頂點
# isClosed – 是否閉合線段
# color – 顏色
# thickness – 線段寬度
```

ex47.py

```python
import numpy as np
import cv2 as cv
cap = cv.VideoCapture('slow_traffic_small.mp4')
# take first frame of the video
ret, frame = cap.read()
# setup initial location of window
x, y, width, height = 300, 200, 100, 50
track_window = (x, y ,width, height)
# set up the ROI for tracking
roi = frame[y:y+height, x : x+width]
hsv_roi = cv.cvtColor(roi, cv.COLOR_BGR2HSV)
mask = cv.inRange(hsv_roi, np.array((0., 60., 32.)), np.array((180., 255., 255)))
roi_hist = cv.calcHist([hsv_roi], [0], mask, [180], [0, 180])
cv.normalize(roi_hist, roi_hist, 0, 255,cv.NORM_MINMAX)
# Setup the termination criteria, either 10 iteration or move by atleast 1 pt
term_crit = ( cv.TERM_CRITERIA_EPS | cv.TERM_CRITERIA_COUNT, 10, 1)
cv.imshow('roi',roi)
```

ex47.py

```python
while(1):
    ret, frame = cap.read()
    if ret == True:

        hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)
        dst = cv.calcBackProject([hsv], [0], roi_hist, [0, 180], 1)
        # apply camshift to get the new location
        ret, track_window = cv.CamShift(dst, track_window, term_crit)
        #print(ret)
        #print(track_window)
        # Draw it on image
        pts = cv.boxPoints(ret)
        print(pts)
        pts = np.int0(pts)
        final_image = cv.polylines(frame, [pts], True, (0, 255, 0), 2)
                                                             # green

        cv.imshow('dst', dst)
        cv.imshow('final_image',final_image)
        k = cv.waitKey(30) & 0xff
        if k == 27:
            break
    else:
        break

cap.release()
cv.destroyAllWindows()
```

CAMshift Object Tracking

ex47.py

PyTesseract: Python Optical Character Recognition | Using Tesseract OCR with Python

In this video we will talk about PyTesseарct. Python-tesseract is an optical character recognition (OCR) tool for python. That is, it will recognize and “read” the text embedded in images.

Image to Text with Python - pytesseract.

Installing Tesseract

1. Download tesseract-ocr-setup-3.02.02.exe and install
2. pip install pytesseract
3. Include this line in python script

https://tesseract-ocr.github.io/tessdoc/Downloads.html

pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files (x86)\Tesseract-OCR\tesseract'

A comprehensive guide to OCR with Tesseract, OpenCV and Python
https://nanonets.com/blog/ocr-with-tesseract/

ex48.py

```
import pytesseract
import cv2 as cv

img = cv.imread('plate.jfif')
cv.imshow('car plate',img)

pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files (x86)\Tesseract-OCR\tesseract'

# get grayscale image
def get_grayscale(image):
    return cv.cvtColor(image, cv.COLOR_BGR2GRAY)

# noise removal
def remove_noise(image):
    return cv.medianBlur(image,5)

# thresholding
def thresholding(image):
    return cv.threshold(image, 0, 255, cv.THRESH_BINARY + cv.THRESH_OTSU)[1]   # [1] = image
```

ex48.py

```python
gray = get_grayscale(img)
thresh = thresholding(gray)
blur = remove_noise(thresh)

text = pytesseract.image_to_string(img)
print(text)

if cv.waitKey(0) & 0xff == 27:
    cv.destroyAllWindows()
```

ex48.py

image

v\iv~9999j

grayscale

VW9999

thresh

VW9999

blur

vvv~9999