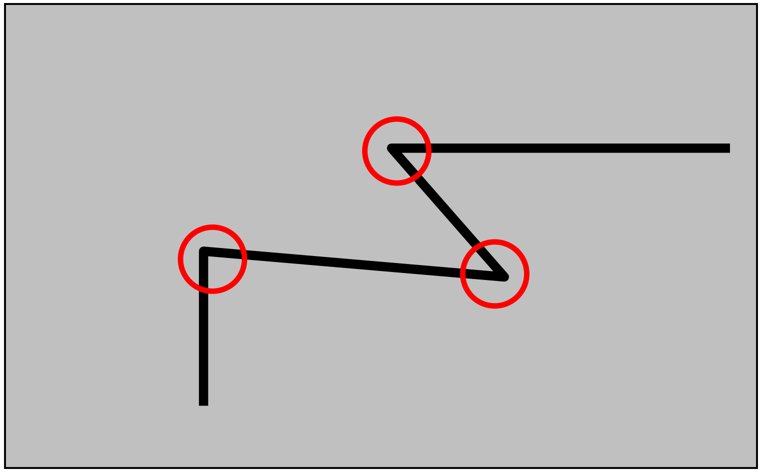
An introductory example:

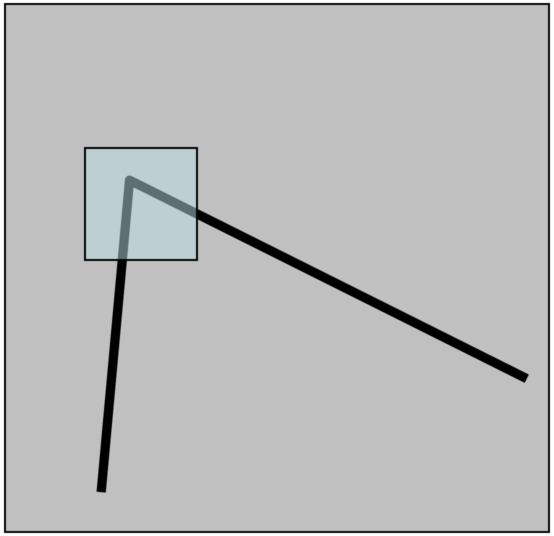
Harris corner detector



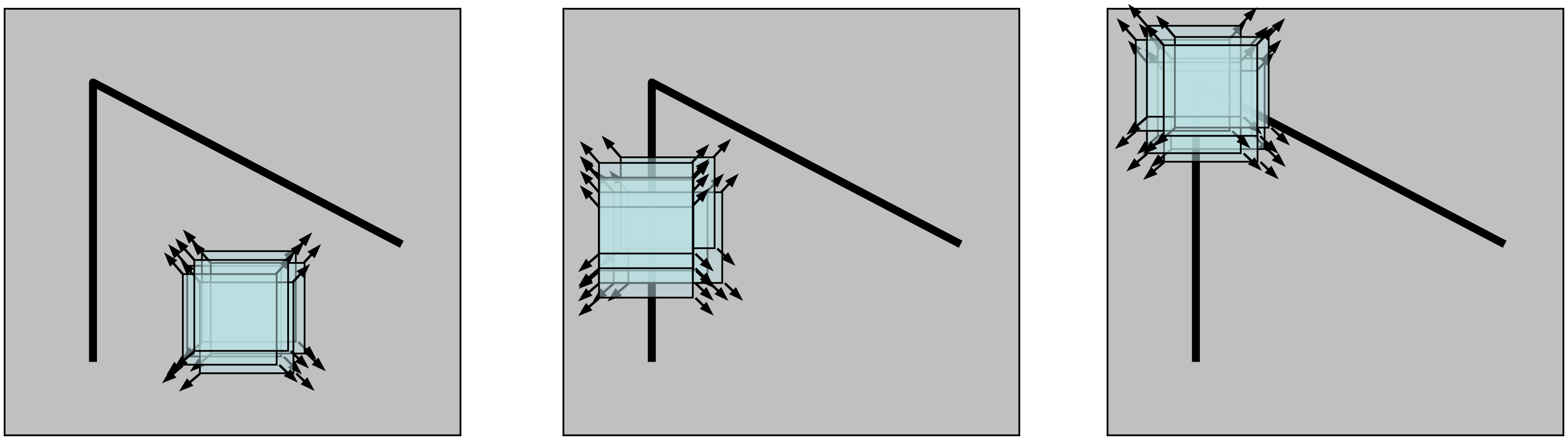
C.Harris, M.Stephens. “A Combined Corner and Edge Detector”. 1988

The Basic Idea

- We should easily recognize the point by looking through a small window
- Shifting a window in *any direction* should give *a large change* in intensity



Harris Detector: Basic Idea



"flat" region:
no change in
all directions

"edge":
no change along
the edge direction

"corner":
significant change
in all directions

Harris Detector: Mathematics

Change of intensity for the shift [u,v]:

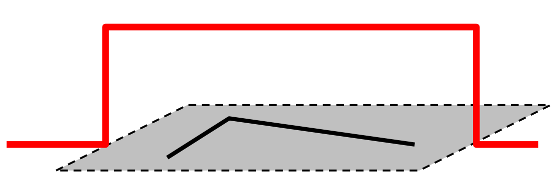
$$E(u,v)=\sum_{x,y} w(x,y)\left[I(x+u,y+v)-I(x,y)\right]^2$$

Window function

Shifted intensity

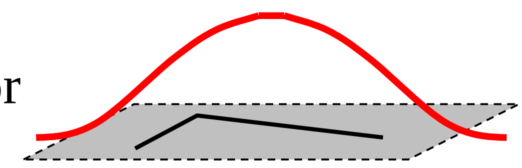
Intensity

Window function $w(x,y)$ =



1 in window, 0 outside

or



Gaussian

Harris Detector: Mathematics*

For small shifts $[u,v]$ we have a *bilinear* approximation:

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

where M is a 2×2 matrix computed from image derivatives:

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

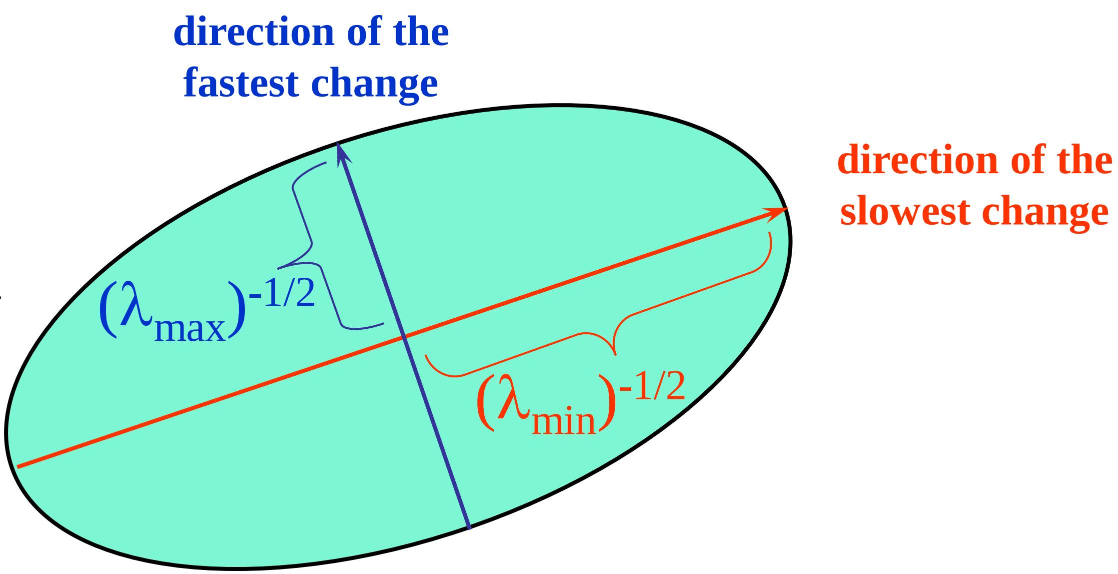
Harris Detector: Eigenvalue Analysis

Intensity change in shifting window: eigenvalue analysis

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

λ_1, λ_2 – eigenvalues of M

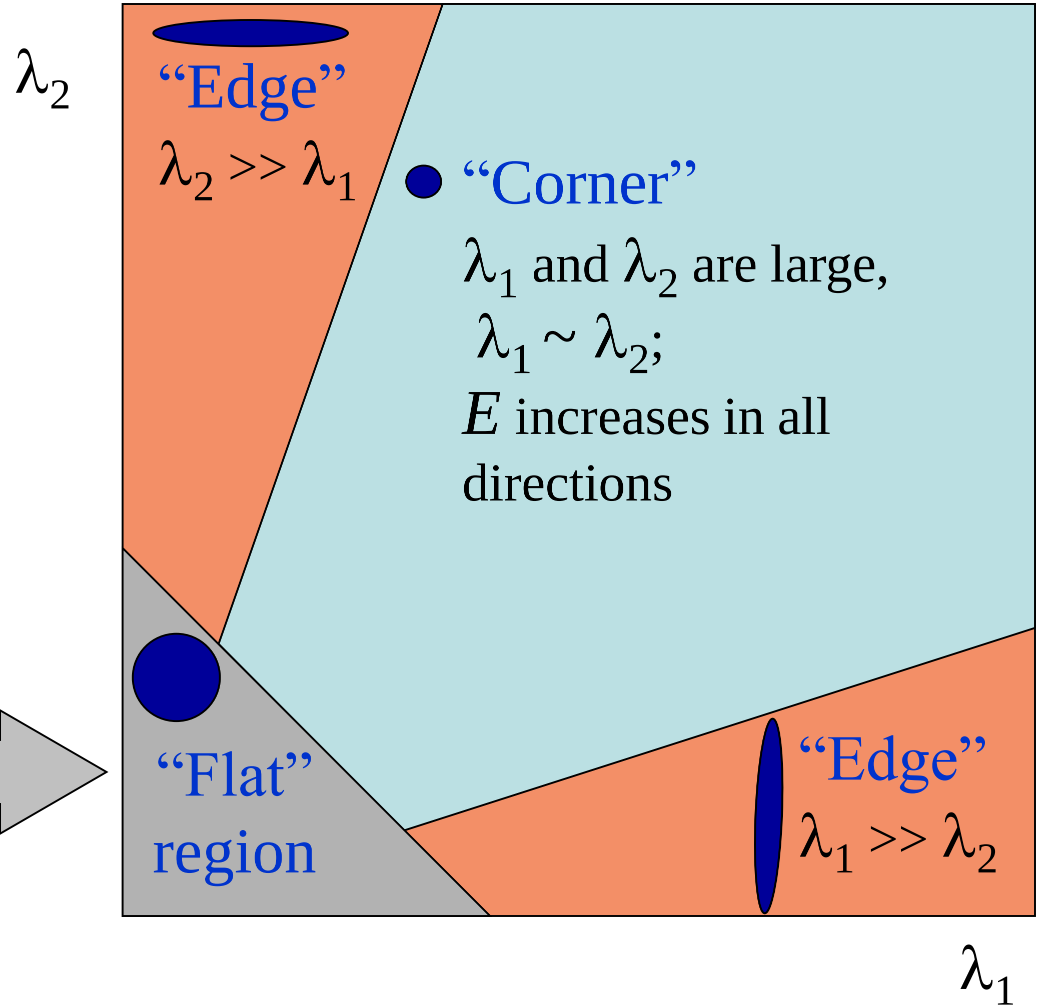
Ellipse $E(u,v)$ = const



Eigenvalues Carry Clues

Classification of image points using eigenvalues of M:

λ_1 and λ_2 are small; E is almost constant in all directions



Corner Response Measure

Measure of corner response:

$$R = \det M - k\left(\operatorname{trace} M\right)^2$$

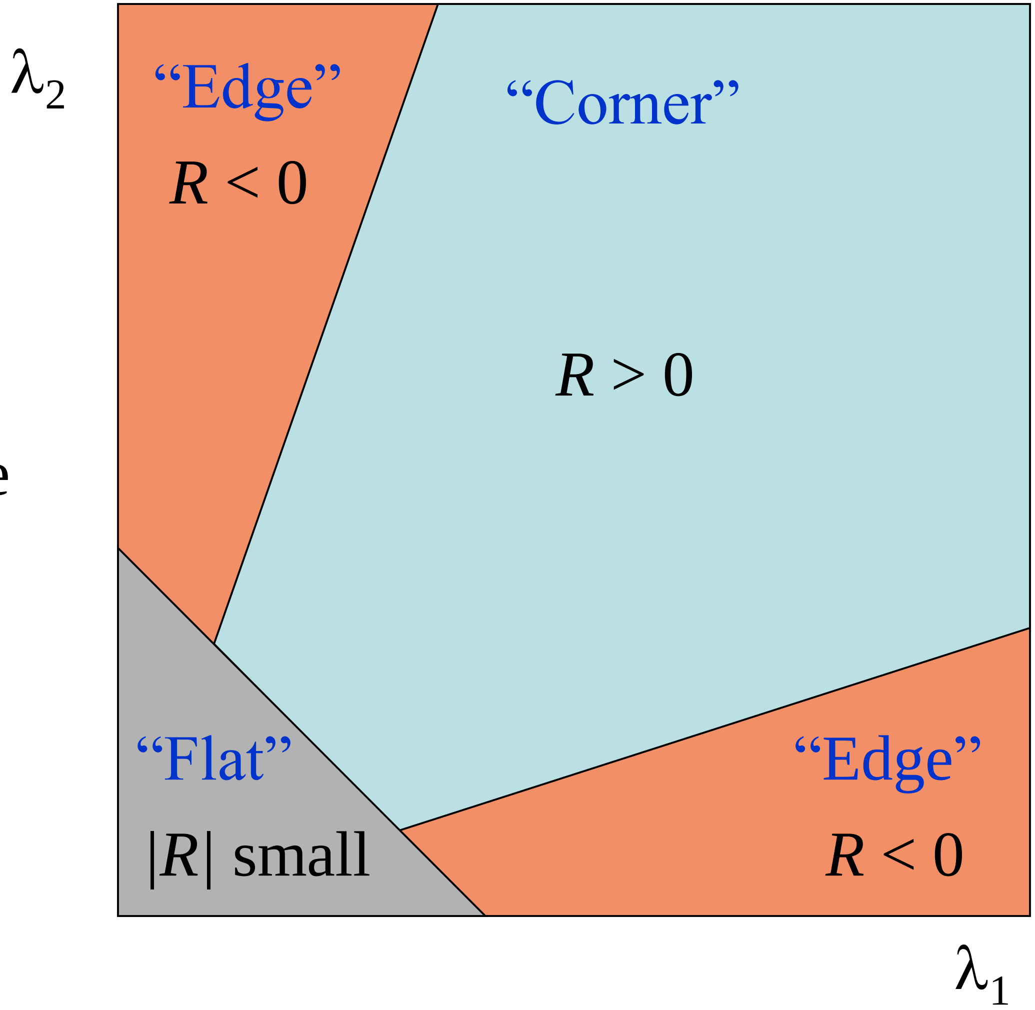
$$\det M = \lambda_1 \lambda_2$$
$$\operatorname{trace} M = \lambda_1 + \lambda_2$$

(k – empirical constant, k = 0.04-0.06)

Note that other definition of corner response measure also exists

Harris Detector: Mathematics

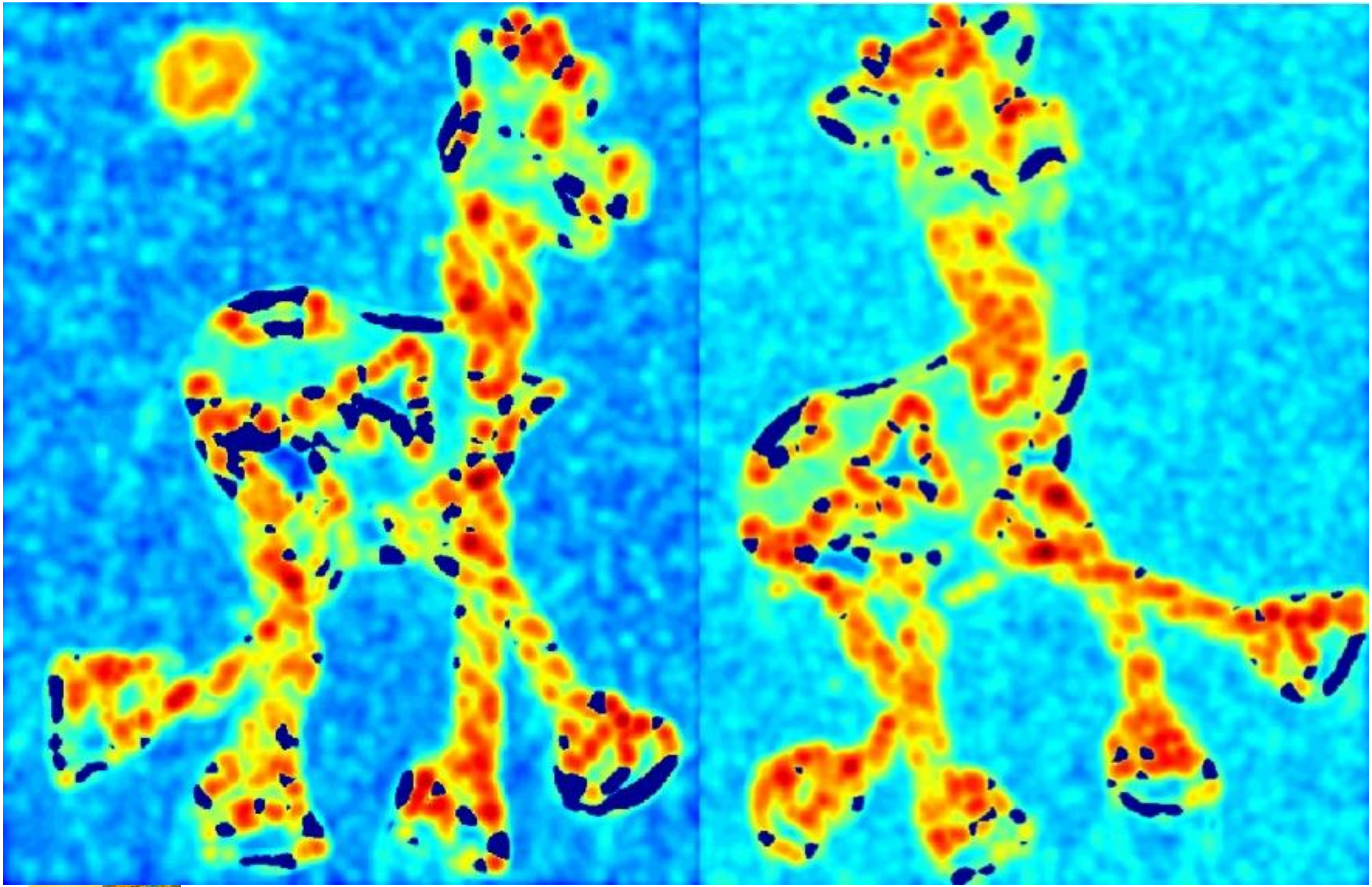
- R depends only on eigenvalues of M
- R is large for a corner
- R is negative with large magnitude for an edge
- $|R|$ is small for a flat region



Harris Detector: Workflow

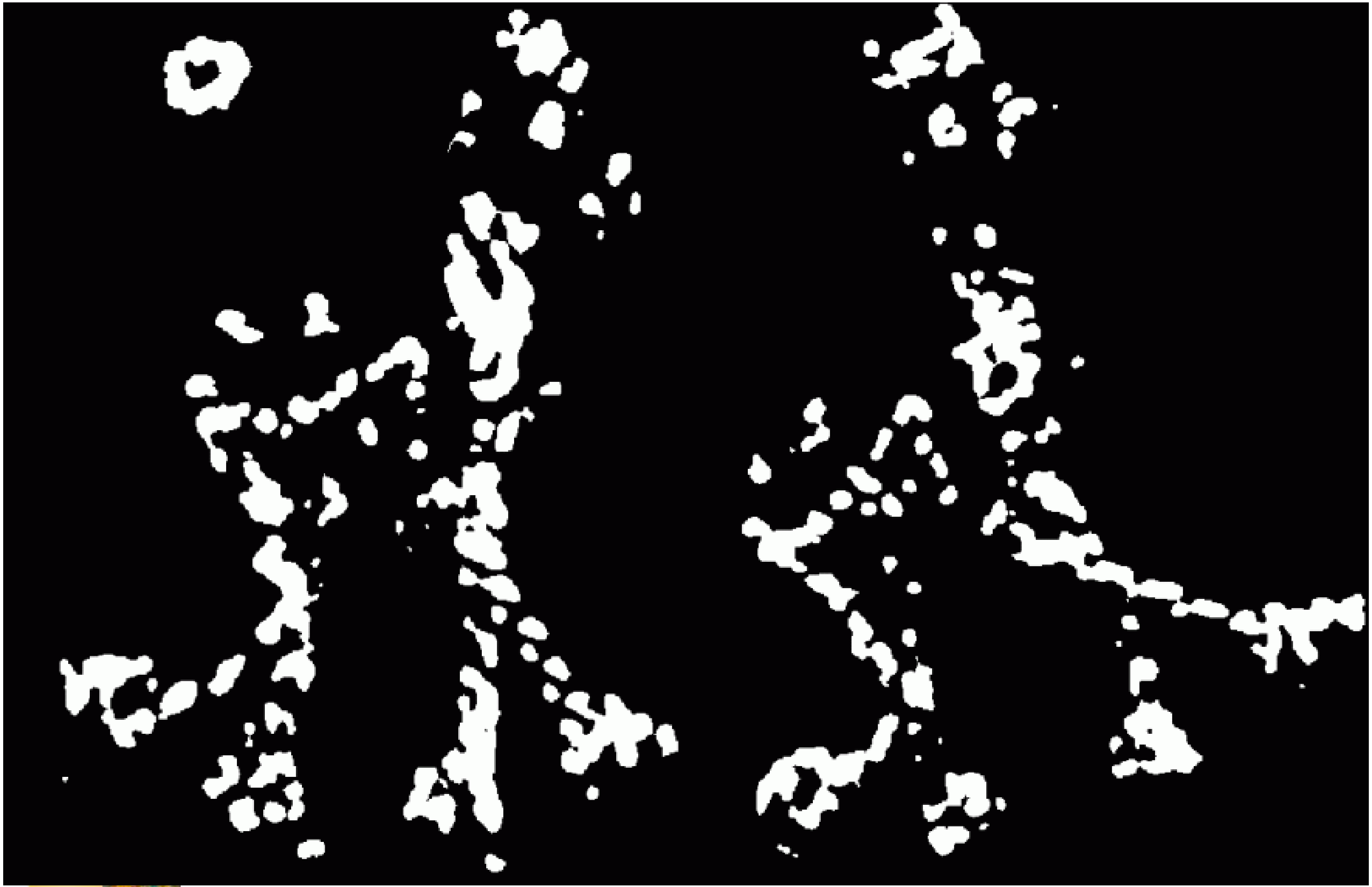
Harris Detector: Workflow

Compute corner response R



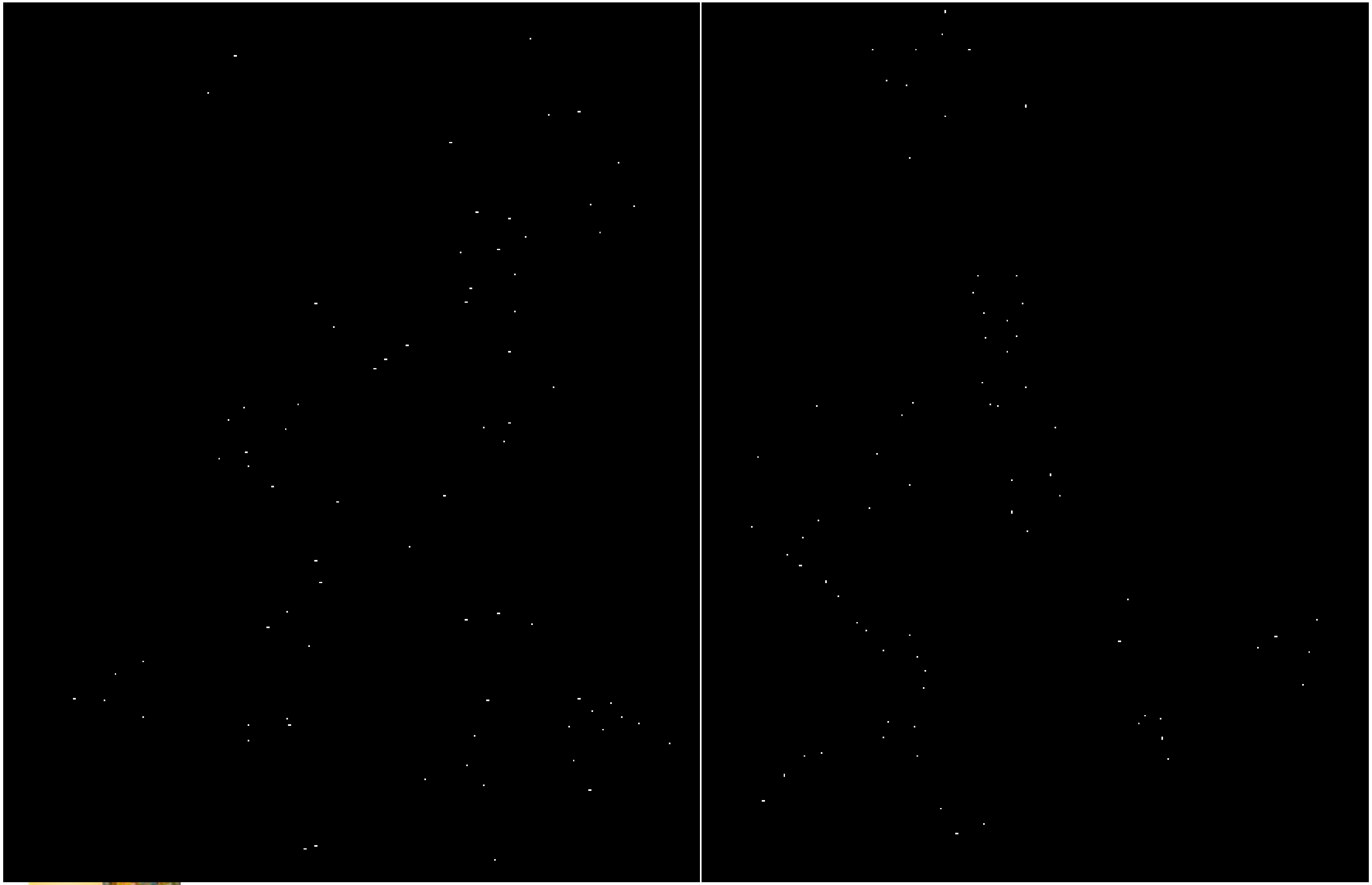
Harris Detector: Workflow

Find points with large corner response: R>threshold



Harris Detector: Workflow

Take only the points of local maxima of R



Harris Detector: Final Results

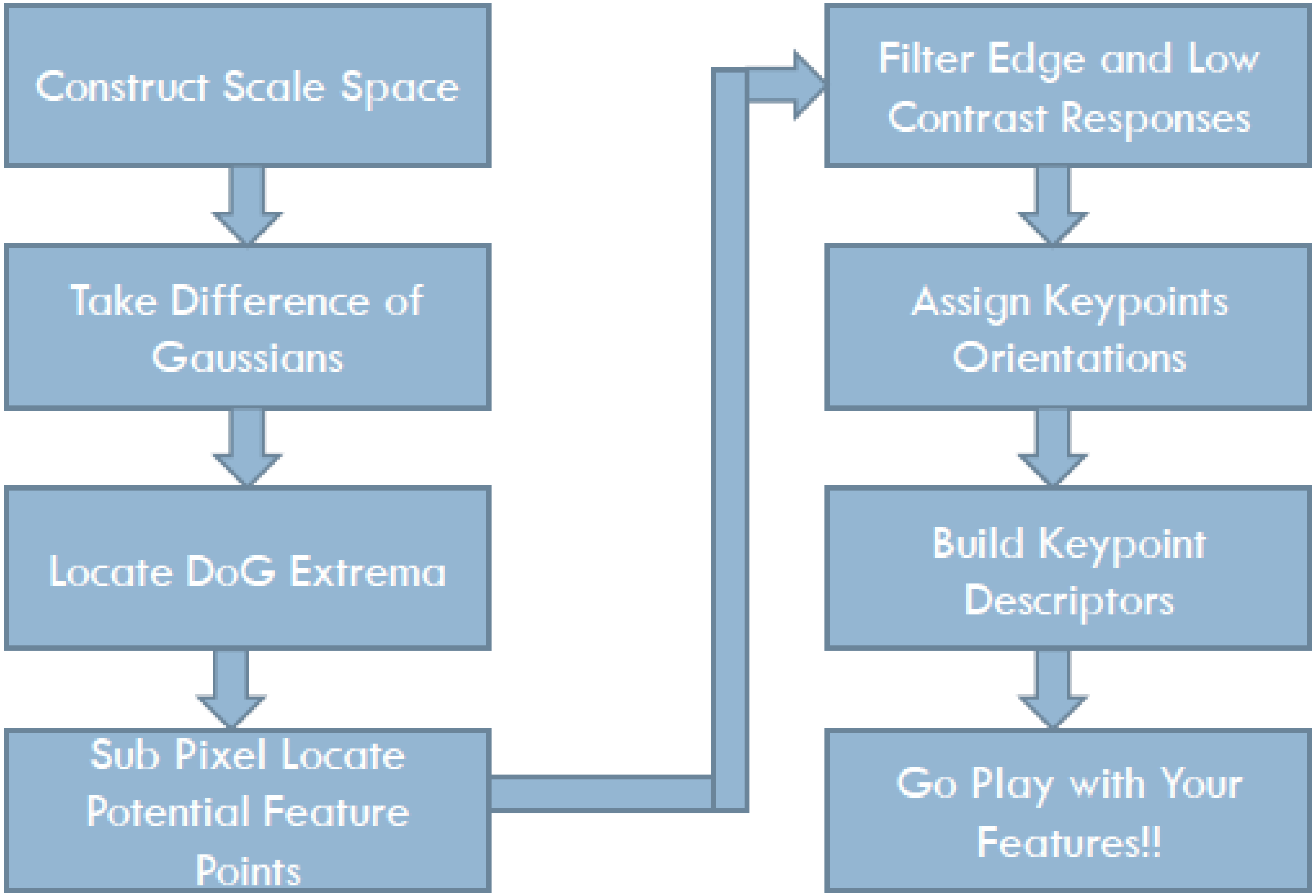
```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('images/checkerboard.png',0)
blockSize = 2;  apertureSize = 3;  k = 0.04;
harris = cv2.cornerHarris(img, blockSize, apertureSize, k,
cv2.BORDER_DEFAULT);

plt.rcParams["figure.figsize"] = [16,9]
plt.subplot(121),plt.imshow(img,cmap='gray'),plt.title('Original')
plt.subplot(122),plt.imshow(harris,cmap='gray'),plt.title('Harris')
plt.show()
```

Roadmap

- Beyond lines
  - How do we define objects of general categories such as faces, pedestrians, vehicles and so on
- Corner detection
  - Harris corner detector
- Keypoint detection*
  - Scale-invariant feature transform (SIFT)[1]
- Applications into the real world
  - Where opportunities and challenges abound

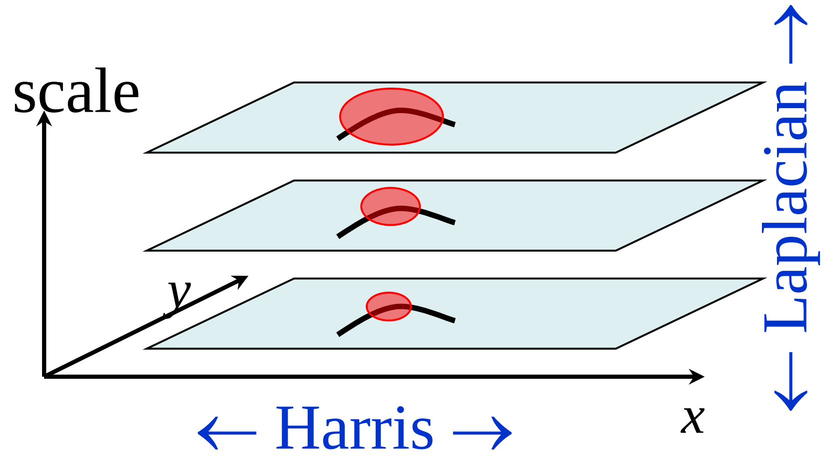
[1]http://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf

Overview of the SIFT Algorithm

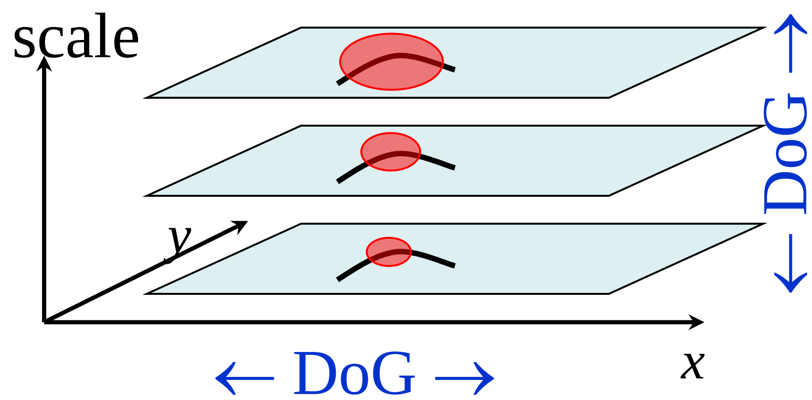


Finding Keypoints – Scale, Location

- Harris-Laplacian[1]
  Find local maximum of:
  - Laplacian in scale
  - Harris corner detector in space (image coordinates)

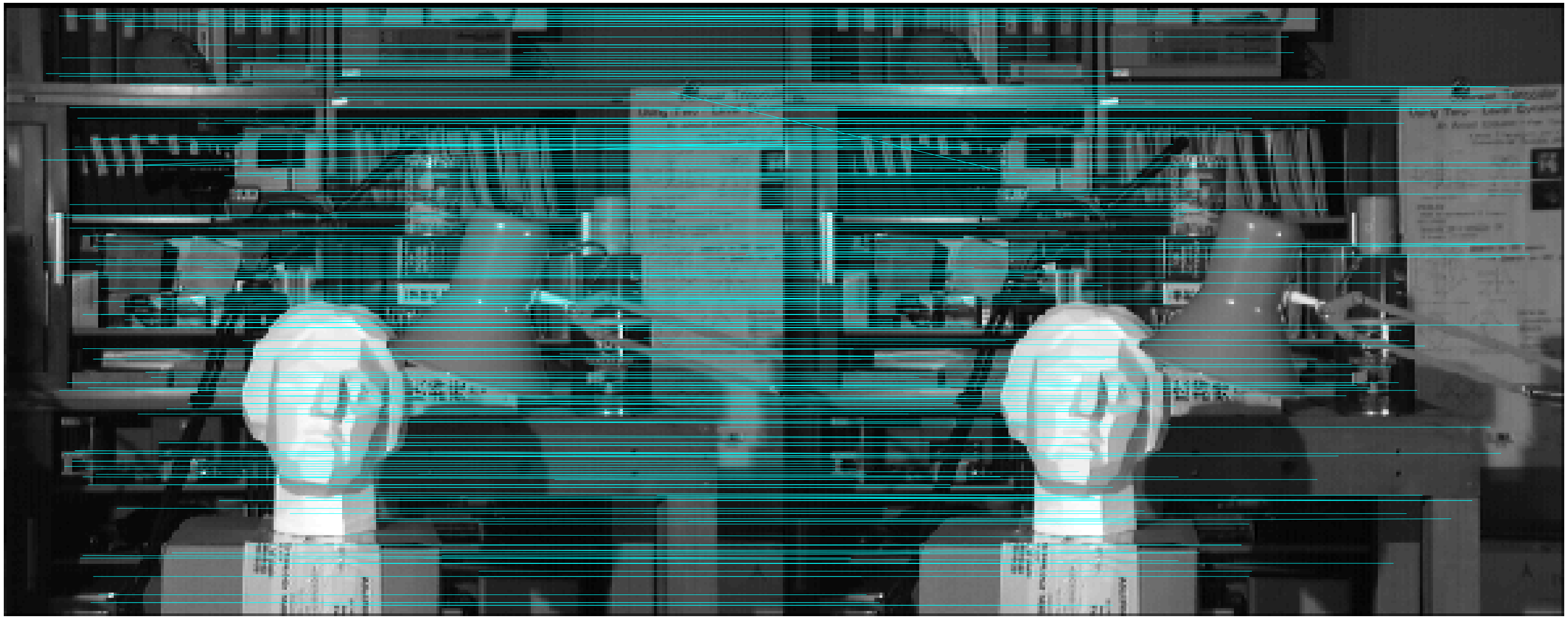


- SIFT[2]
  Find local maximum of:
  – Difference of Gaussians in space and scale



[1] K.Mikolajczyk, C.Schmid. "Indexing Based on Scale Invariant Interest Points". ICCV 2001
[2] D.Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". IJCV 2004

Image Example

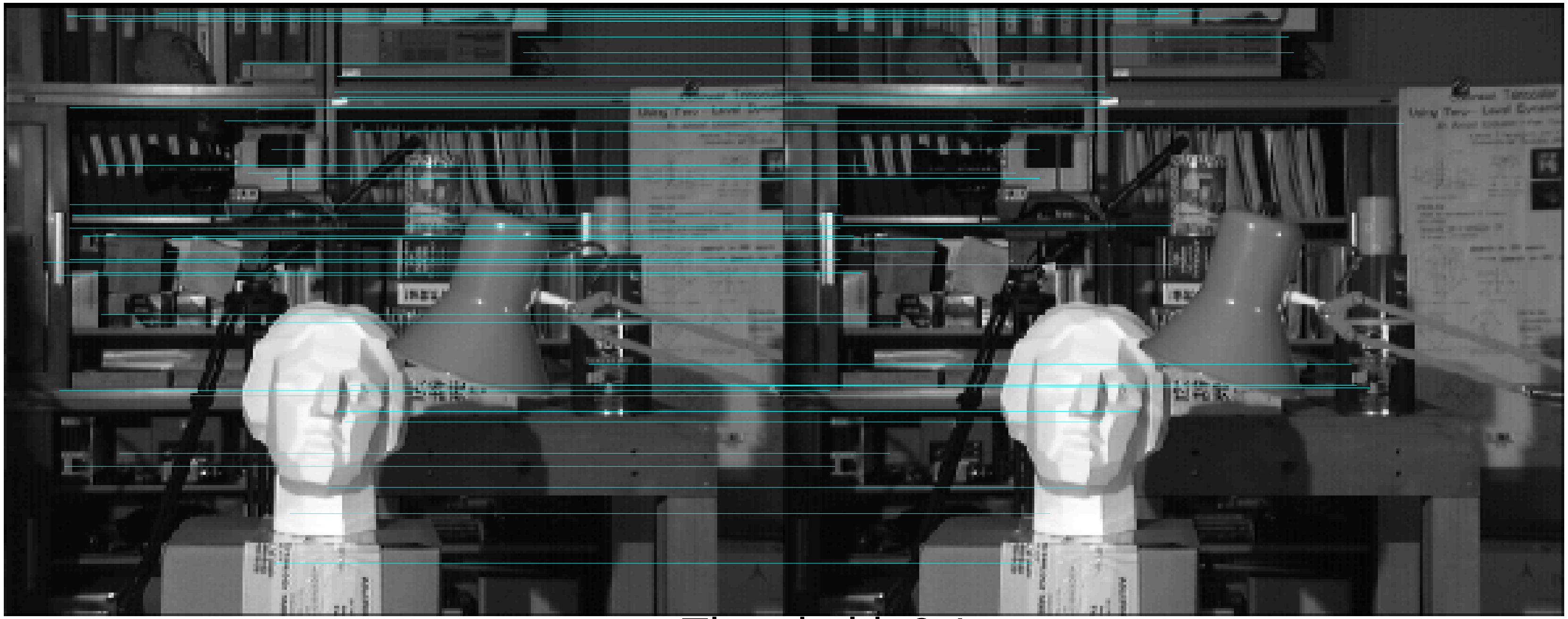


Threshold=0.6

Left view

Right view

Image Example



Threshold=0.1

Left view Right view

```
import cv2
import numpy as np
img = cv2.imread('cameraman2.tif')
gray= cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
sift = cv2.xfeatures2d.SIFT_create()
kp = sift.detect(gray,None)
img=cv2.drawKeypoints(gray,kp,img)
```

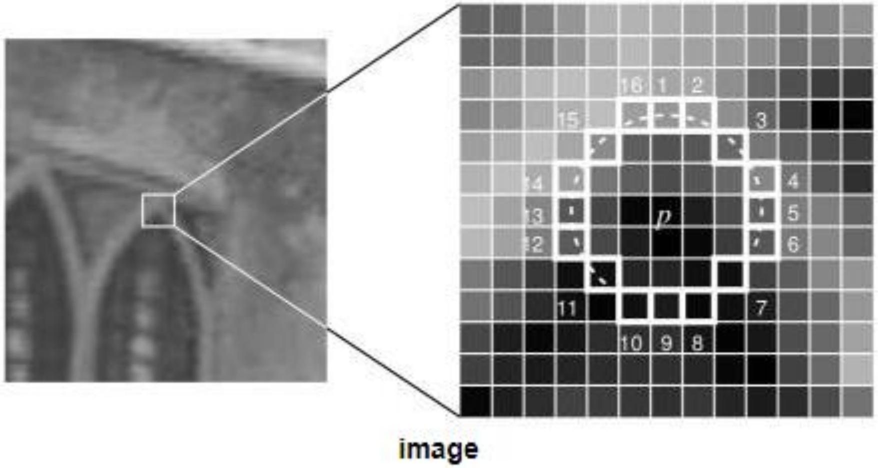
Output

100 Strongest Points

100 Strongest Feature Points from Box Image

FAST Algorithm

- FAST (Features from Accelerated Segment Test) algorithm is proposed for fast feature extraction (proposed by Rodsten in 2006)
- First select a pixel p and threshold t
- Consider a 16 pixels around p to perform the test
- Pixel p is a corner, if there exists a set of n contiguous pixels in the circle are all brighter than Ip + t or darker than Ip – t
- A high speed test was proposed to examines only 4 pixels at 1, 9, 5 and 13



```
import numpy as np
import cv2
from matplotlib import pyplot as plt
img = cv2.imread('images/FeatureBasedObjectDetectionExample_01.png',0)
# Initiate FAST object with default values
fast = cv2.FastFeatureDetector_create()
# find and draw the keypoints
kp = fast.detect(img,None)
img2 = cv2.drawKeypoints(img, kp, None, color=(255,0,0))
# Print all default params
print( "Threshold: {}".format(fast.getThreshold()) )
print( "nonmaxSuppression:{}".format(fast.getNonmaxSuppression()) )
print( "neighborhood: {}".format(fast.getType()) )
print( "Total Keypoints with nonmaxSuppression: {}".format(len(kp)) )
cv2.imwrite('fast_true.png',img2)
# Disable nonmaxSuppression
fast.setNonmaxSuppression(0)
kp = fast.detect(img,None)
print( "Total Keypoints without nonmaxSuppression: {}".format(len(kp)) )
img3 = cv2.drawKeypoints(img, kp, None, color=(255,0,0))
cv2.imshow("Keypoint", img3)
cv2.waitKey(0)
```

ORB Algorithm

- ORB: an efficient alternate to SIFT or SURF
- First, it is used FAST to find keypoints
- Then, apply Harris corner to find top N points among them
- After that, it uses multi-scale pyramid to produce multi-scale features

ORB

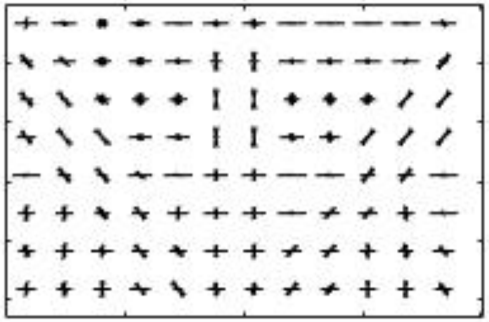
```
import numpy as np
import cv2
from matplotlib import pyplot as plt
img = cv2.imread('images/FeatureBasedObjectDetectionExample_01.png',0)
# Initiate ORB detector
orb = cv2.ORB_create()
# find the keypoints with ORB
kp = orb.detect(img,None)
# compute the descriptors with ORB
kp, des = orb.compute(img, kp)
# draw only keypoints location,not size and orientation
img2 = cv2.drawKeypoints(img, kp, None, color=(0,255,0), flags=0)
plt.imshow(img2), plt.show()
```

Histogram Of Gradient

- This feature descriptors used in computer vision and image processing for the purpose of object detection
- The technique counts occurrences of gradient orientation in localized portions of an image
- This method is similar to scale-invariant feature transform descriptors but differs in that it is computed on a dense grid of uniformly spaced cells and uses overlapping local contrast normalization for improved accuracy

Result

Results

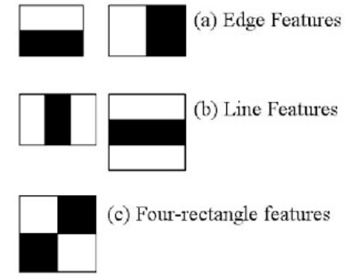


HOG

```
import cv2
import numpy as np

def inside(r, q):
    rx, ry, rw, rh = r
    qx, qy, qw, qh = q
    return rx > qx and ry > qy and rx + rw < qx + qw and ry + rh < qy + qh


def draw_detections(img, rects, thickness = 1):
    for x, y, w, h in rects:
        # the HOG detector returns slightly larger rectangles than the real objects.
        # so we slightly shrink the rectangles to get a nicer output.
        pad_w, pad_h = int(0.15*w), int(0.05*h)
        cv2.rectangle(img, (x+pad_w, y+pad_h), (x+w-pad_w, y+h-pad_h), (0, 255, 0), 
thickness)
```

```
image = cv2.imread('images/pedestrian.jpg',0)
hog = cv2.HOGDescriptor()
hog.setSVMDetector(
cv2.HOGDescriptor_getDefaultPeopleDetector() )
found,w=hog.detectMultiScale(image, winStride=(8,8),
padding=(32,32), scale=1.05)
draw_detections(image,found)
print(found)
cv2.imshow('feed',image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

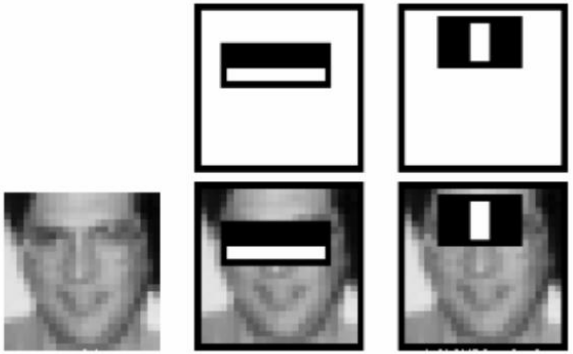
HAAR Detection



image

- Proposed by Viola and Jones in 2001
- Each feature is a single value obtained by subtracting sum of pixels under white and black
- Each feature is focused on different property e.g., the region of the eye is darker than the nose.
- The concept of cascade classifiers, if the windows fails on the first stage,

  we discard it.



image

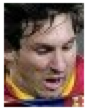
HAAR

```
import numpy as np
import cv2
face_cascade = cv2.CascadeClassifier('cascades/haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('cascades/haarcascade_eye.xml')
img = cv2.imread('images/woman.tif')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

faces = face_cascade.detectMultiScale(gray, 1.3, 5)
for (x,y,w,h) in faces:
    cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
    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.imshow('img',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Template Matching

- Search for similar object in other pictures

  - cv2.TM_CCOEFF



image

image

Template Matching

- It uses a sliding windows to search the image and compare
- There are many methods for comparison including

a. **method=CV_TM_SQDIFF**

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

b. **method=CV_TM_SQDIFF_NORMED**

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

c. **method=CV_TM_CCORR**

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

d. **method=CV_TM_CCORR_NORMED**

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

e. **method=CV_TM_CCOEFF**

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

where

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

f. **method=CV_TM_CCOEFF_NORMED**

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

Template Matching

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('images/messi5.jpg',0)
img2 = img.copy()
template = cv2.imread('images/messi_face.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']
```

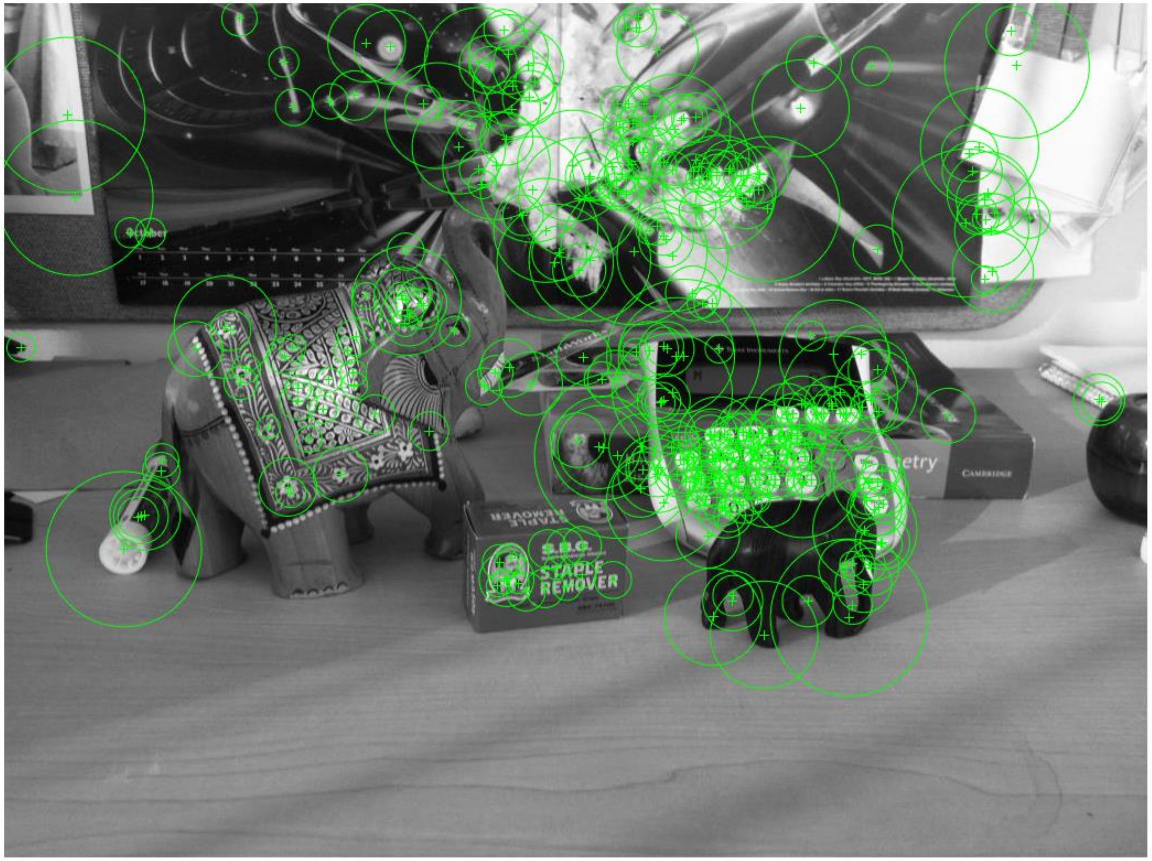
```python
for meth in methods:
    img = img2.copy()
    method = eval(meth)
    # Apply template Matching
    res = cv2.matchTemplate(img,template,method)
    min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)
    # If the method is TM_SQDIFF or TM_SQDIFF_NORMED, take minimum
    if method in [cv2.TM_SQDIFF,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

Matching

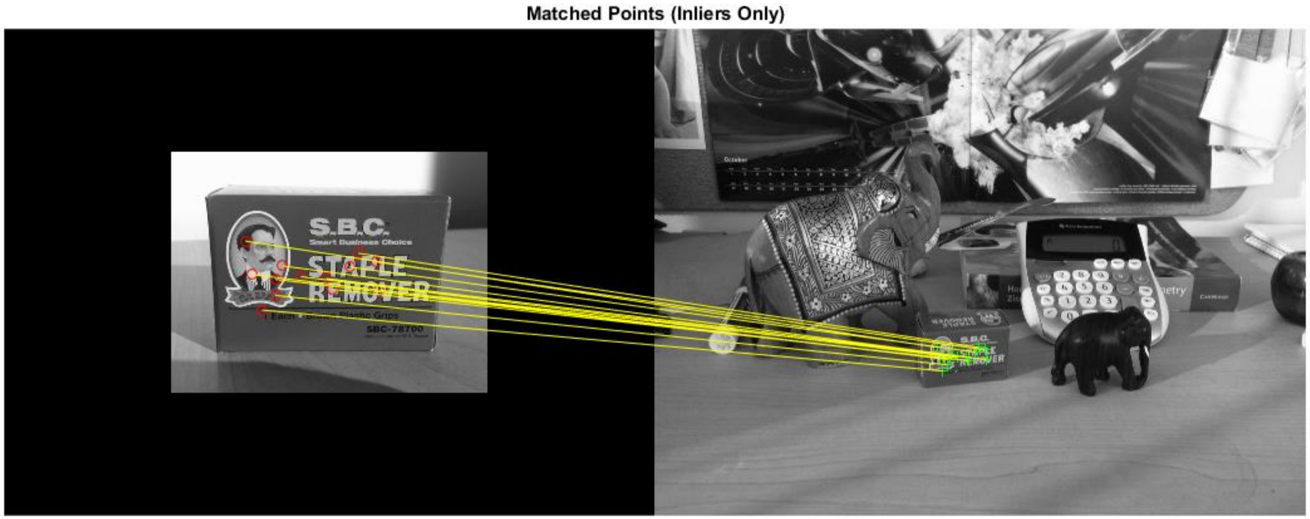
- Searching for keypoints from one image into another image
- The algorithm for matching can be :
  - Brute Force
  - FLANN (Fast Library for Approximate Nearest Neighbors)

300 Strong points from Scene images

300 Strongest Feature Points from Scene Image



Match Points



Output

Detected Box

Brute Force

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
img1 = cv2.imread('images/stapleRemover.jpg',0)          # queryImage
img2 = cv2.imread('images/clutterDesk.jpg',0) # trainImage
# Initiate ORB detector
orb = cv2.ORB_create()
# find the keypoints and descriptors with ORB
kp1, des1 = orb.detectAndCompute(img1,None)
kp2, des2 = orb.detectAndCompute(img2,None)
```

```
# create BFMatcher object
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
# Match descriptors.
matches = bf.match(des1,des2)
# Sort them in the order of their distance.
matches = sorted(matches, key = lambda x:x.distance)
# print(matches)
# Draw first 10 matches.
img3 = img2.copy()
img3 = cv2.drawMatches(img1,kp1,img2,kp2,matches[:5], img3)
# plt.imshow(img3),plt.show()

plt.rcParams["figure.figsize"] = [16,9]
plt.subplot(2,2,1),plt.imshow(img1,'gray')
plt.subplot(2,2,2),plt.imshow(img2,'gray')
plt.subplot(2,2,3),plt.imshow(img3,'gray')
#  plt.subplot(2,2,4),plt.imshow(des2,'gray')
plt.show()
```

FLANN

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
img1 = cv2.imread('images/stapleRemover.jpg',0)          # queryImage
img2 = cv2.imread('images/clutterDesk.jpg',0) # trainImage
# Initiate ORB detector
orb = cv2.ORB_create()
# find the keypoints and descriptors with ORB
kp1, des1 = orb.detectAndCompute(img1,None)
kp2, des2 = orb.detectAndCompute(img2,None)

# FLANN parameters
FLANN_INDEX_LSH = 6
index_params= dict(algorithm = FLANN_INDEX_LSH,
                   table_number = 6, # 12
                   key_size = 12,     # 20
                   multi_probe_level = 1) #2
```

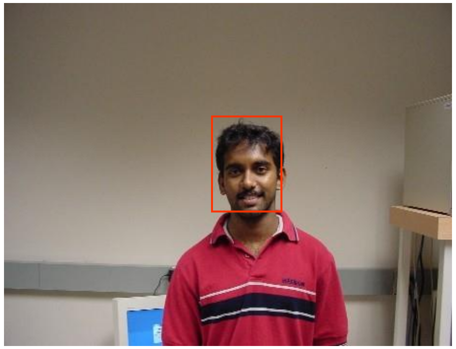
```
search_params = dict(checks=10)   # or pass empty dictionary
flann = cv2.FlannBasedMatcher(index_params,search_params)
matches = flann.knnMatch(des1,des2,k=2)
# Need to draw only good matches, so create a mask
matchesMask = [[0,0] for i in range(len(matches))]
# ratio test as per Lowe's paper
for i,(m,n) in enumerate(matches):
    if m.distance < 0.7*n.distance:
        matchesMask[i]=[1,0]
draw_params = dict(matchColor = (0,255,0),
                   singlePointColor = (255,0,0),
                   matchesMask = matchesMask,
                   flags = 0)
img3 = cv2.drawMatchesKnn(img1,kp1,img2,kp2,matches,None,**draw_params)

plt.rcParams["figure.figsize"] = [16,9]
plt.subplot(2,2,1),plt.imshow(img1,'gray')
plt.subplot(2,2,2),plt.imshow(img2,'gray')
plt.subplot(2,2,3),plt.imshow(img3,'gray')
#  plt.subplot(2,2,4),plt.imshow(des2,'gray')
plt.show()
```

Summary of Object Detection

- Existing approaches toward object detection involve a lot of tinkering (no winning theory)
- Feature extraction (corner or keypoint) has shown practical success in various applications – but how do we define features?
- The gap between the very best machine vision system and human vision system on object detection/recognition performance is still significant

- Applications into the real world
  - Where opportunities and challenges abound

Face Detection



Face detection problem: do you see any human faces in an image? (trivial for human eyes but difficult for computers)

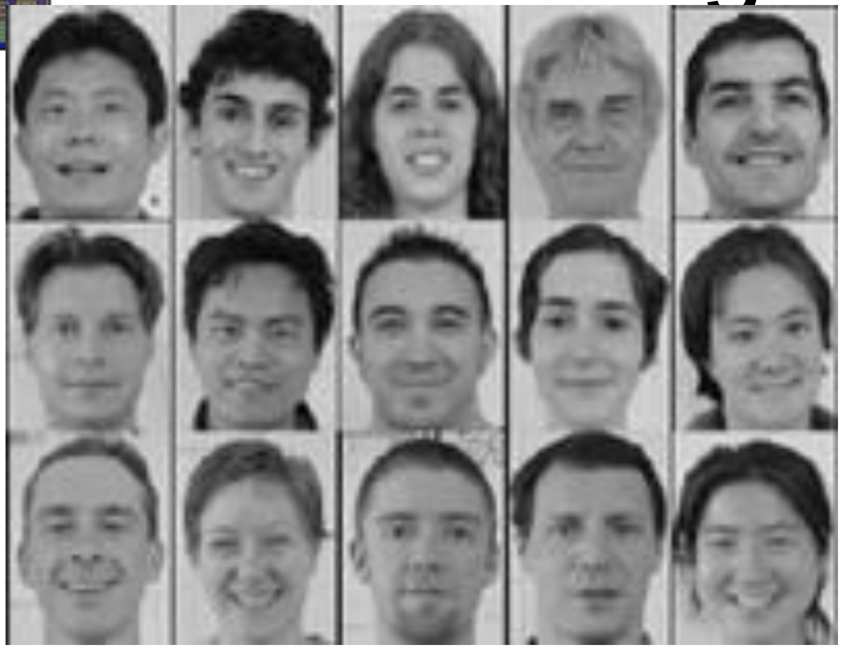
Challenges with FD

pose variation

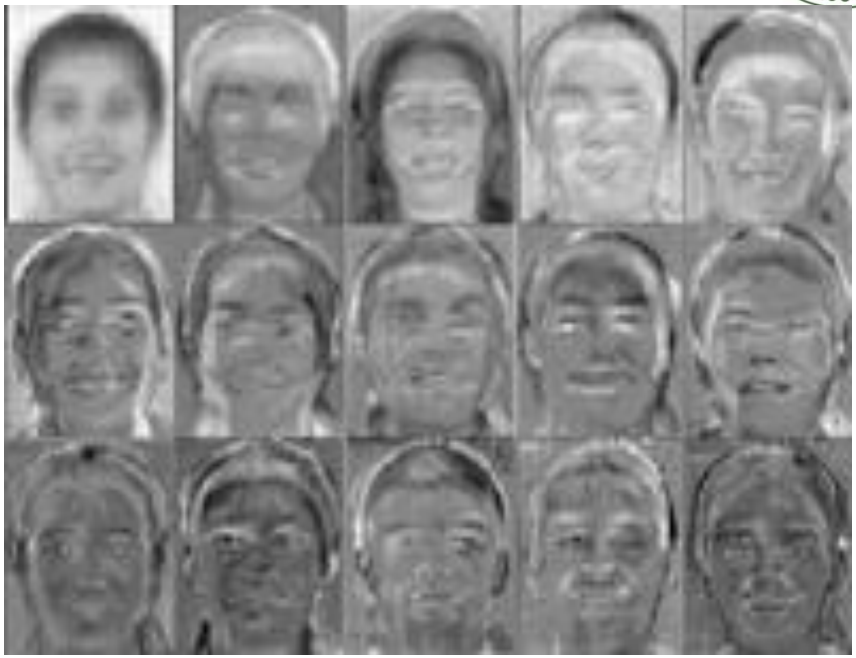
lighting condition variation

facial expression variation

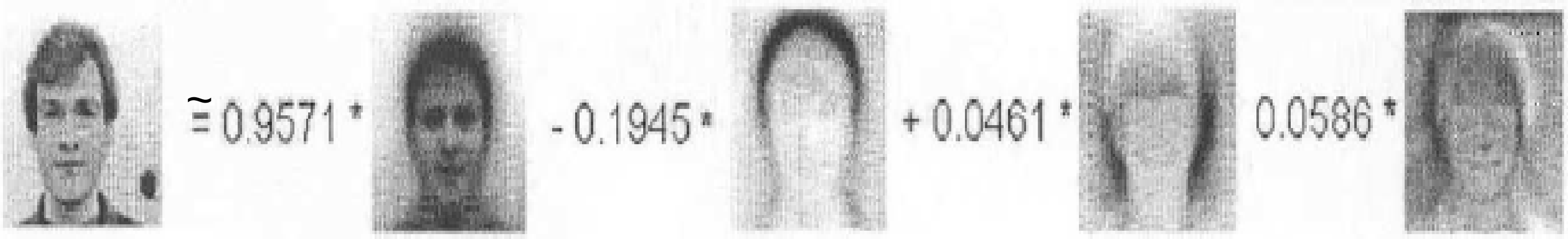
Eigenfaces



Training data

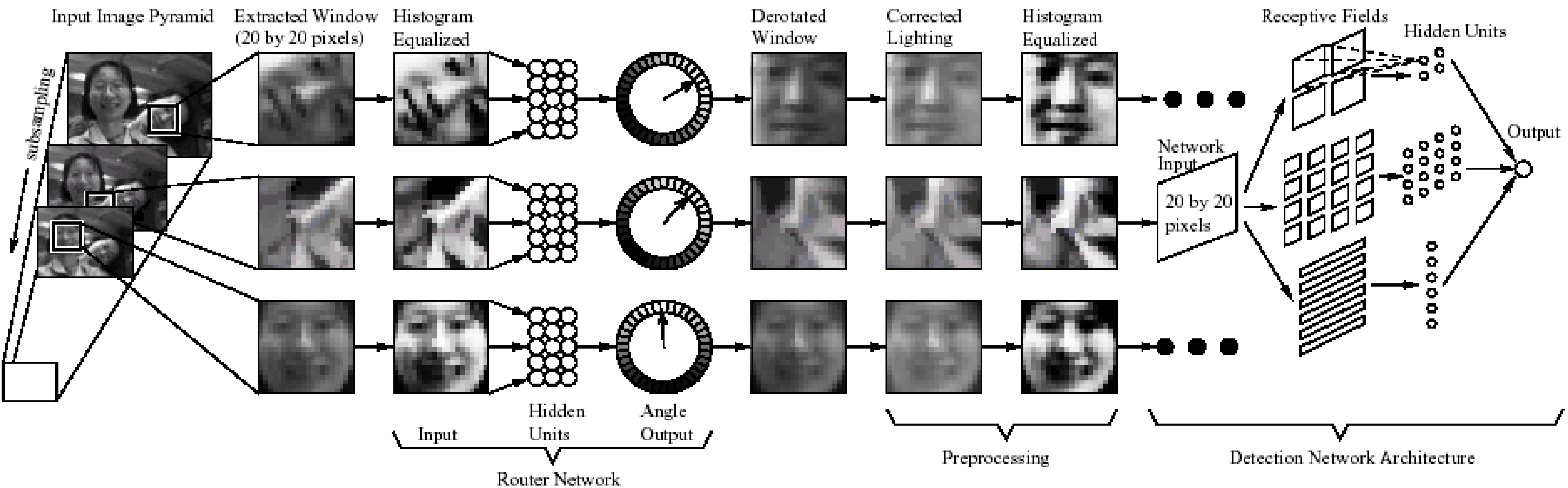


Eigenface images



w_1 w_2 w_3 w_4

Neural Network based FD



Cited from "Neural network based face detection", by Henry A. Rowley, Ph.D. thesis, CMU, May 1999

FD Examples

Pedestrian Detection

- Why do we need to detect pedestrians?
  - Security monitoring
  - Intelligent vehicles
  - Video database search
- Challenges
  - Uncertainty with pedestrian profile, viewing distance and angle, deformation of human limb

Learning-based Pedestrian Detection

Thermal (Infrared) Imaging

Visible-spectrum image

Infrared image

Image Examples

Vehicle Detection

- Intelligent vehicles aim at improving the driving safety by machine vision techniques

http://www.mobileye.com/visionRange.shtml

Questions?