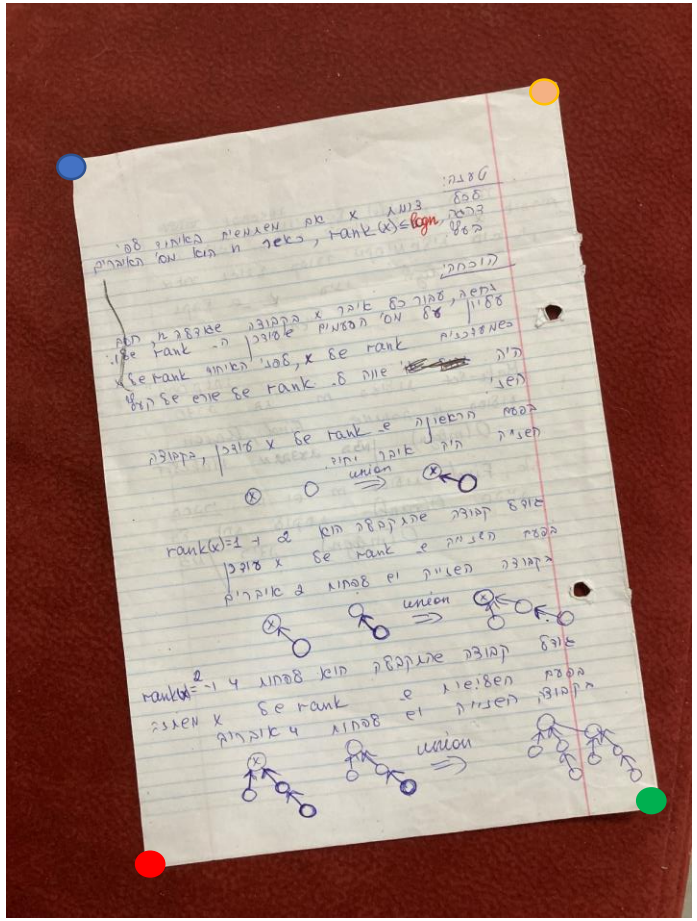
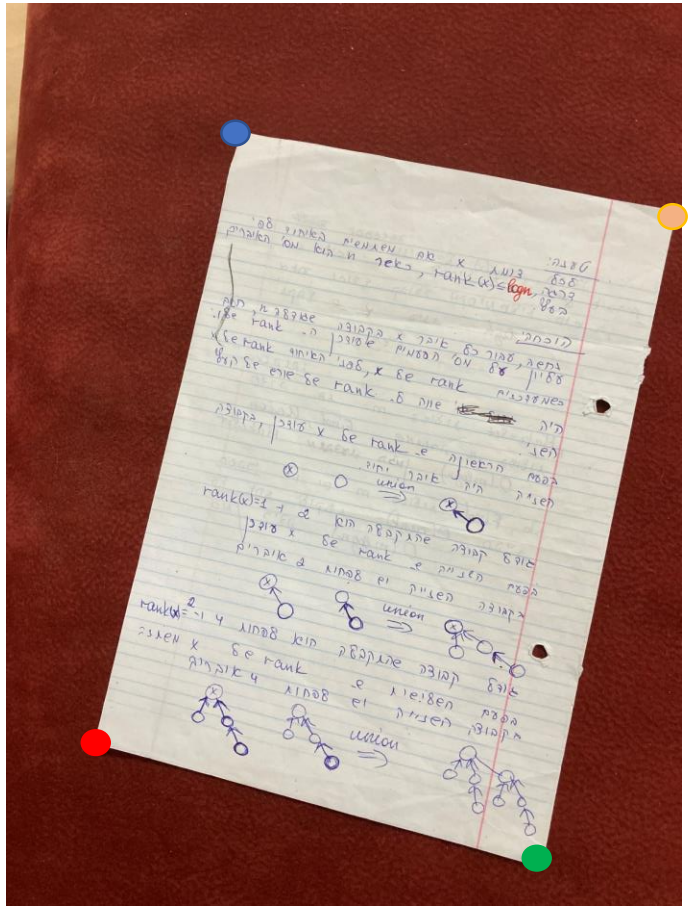
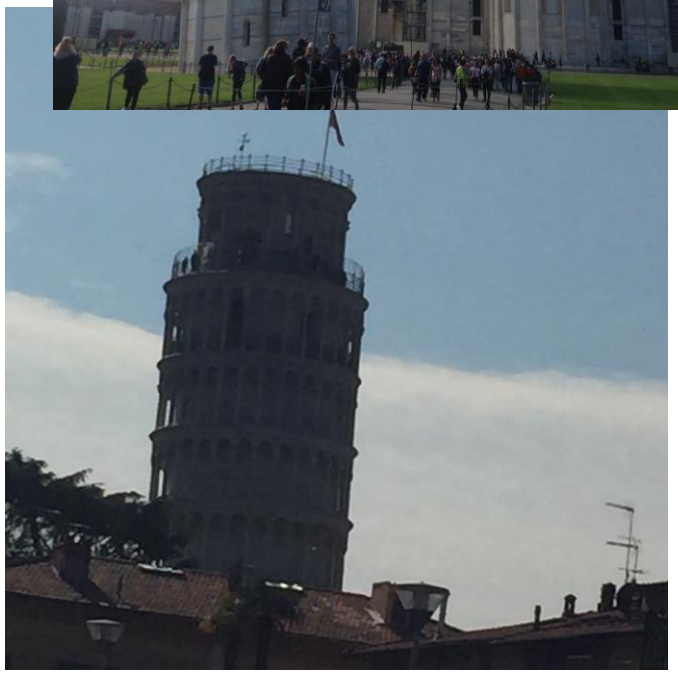
Keypoints Extraction and Matching

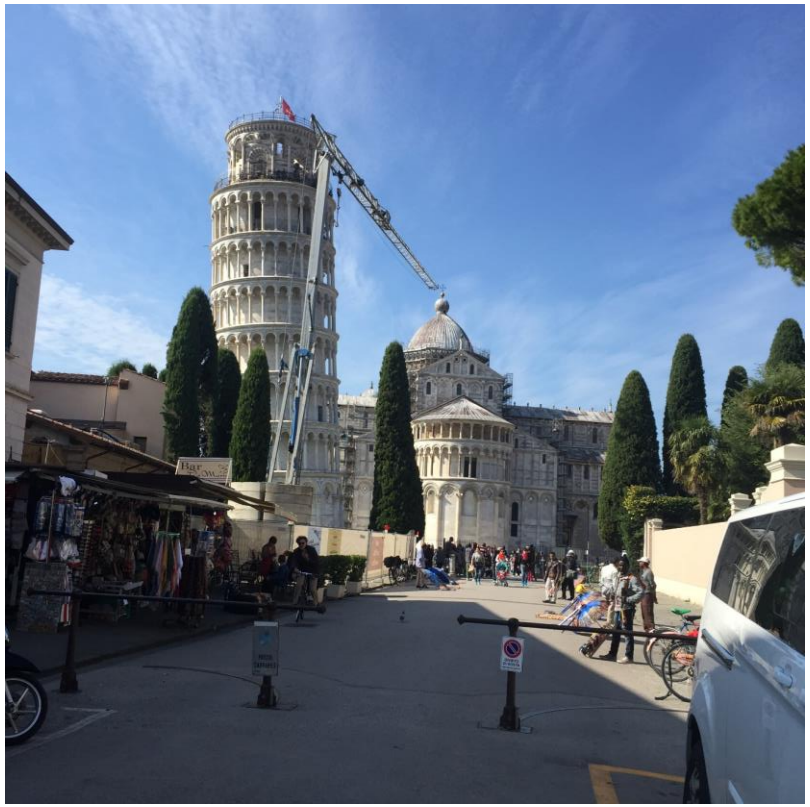
Introduction





Introduction

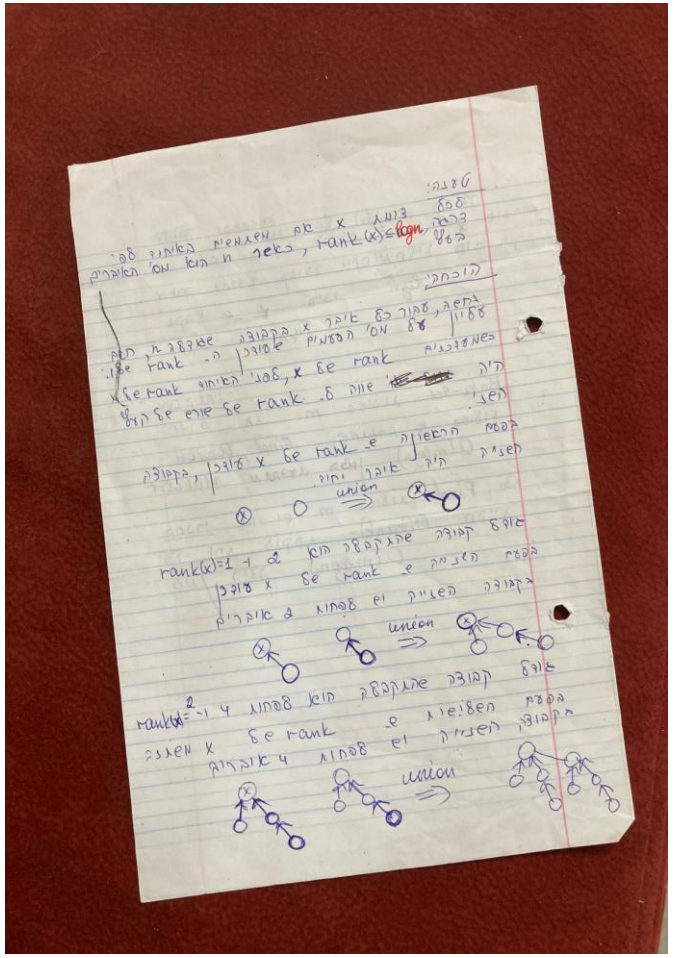




Keypoints

- In this lesson, we will learn about keypoints (interesting point or local features) extraction and matching
  - Keypoint is a region of an image that is unique and easily recognizable
  - Keypoint feature is an image pattern which differs from its immediate neighborhood.
  - Keypoint feature lies in specific locations of the images, such as building corners, doorways, eye of the dog, etc. In other words, keypoint is a location which differentiates an object in an image

Keypoints

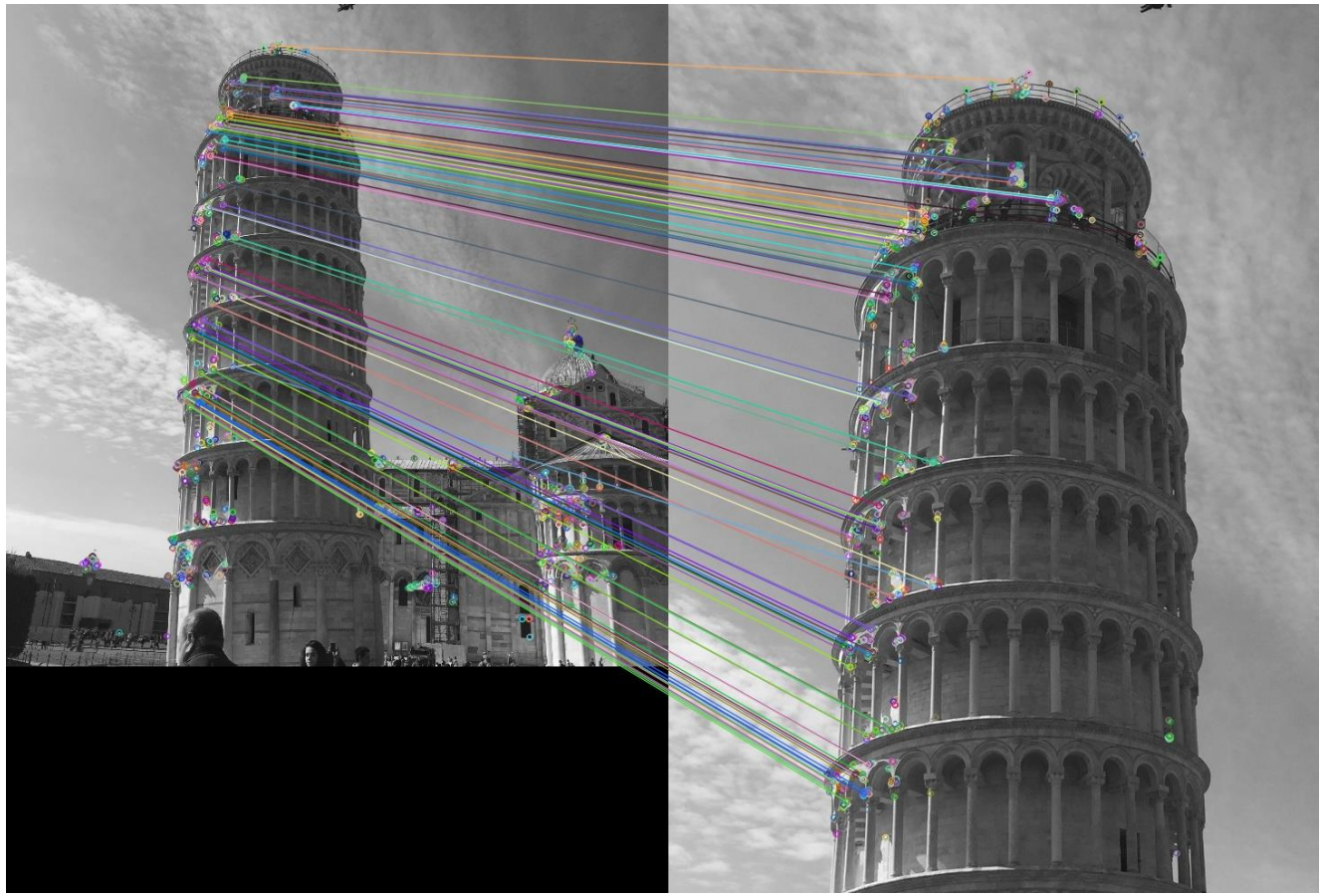


Where to look for a keypoint?

- Keypoint is the point at which the direction of the boundary of the object changes abruptly, or its is an intersection point between two or more edge segments.

Why are keypoints special?

- The keypoints are special, because even if the image changes (rotates, shrinks/expands, or is translated), we should be able to find the **same** keypoints in a modified image when comparing with the original image



Application Of Keypoints Extraction And Matching

- Automate object/motion tracking
- Image retrieval
- Object or scene recognition
- 3D structure reconstruction from multiple images
- Robot navigation

Main Component Of Keypoint Extraction And Matching

- **Detection:** Detect the **keypoint** – find (x,y) coordinates
- **Description:** Describe and quantify the region of the image surrounding the keypoint. We typically end up with a descriptor vector for each keypoint ("signature").
- **Matching:** Descriptors are compared across the images, to identify similar features. For two images we may get a set of pairs $(\boldsymbol{X_i}, \boldsymbol{Y_i}) \leftrightarrow (\boldsymbol{X'_i}, \boldsymbol{Y'_i})$, where $(\boldsymbol{X_i}, \boldsymbol{Y_i})$ is a keypoint in one image and $(\boldsymbol{X'_i}, \boldsymbol{Y'_i})$ its matching keypoint in the other image.

Desired properties of keypoints

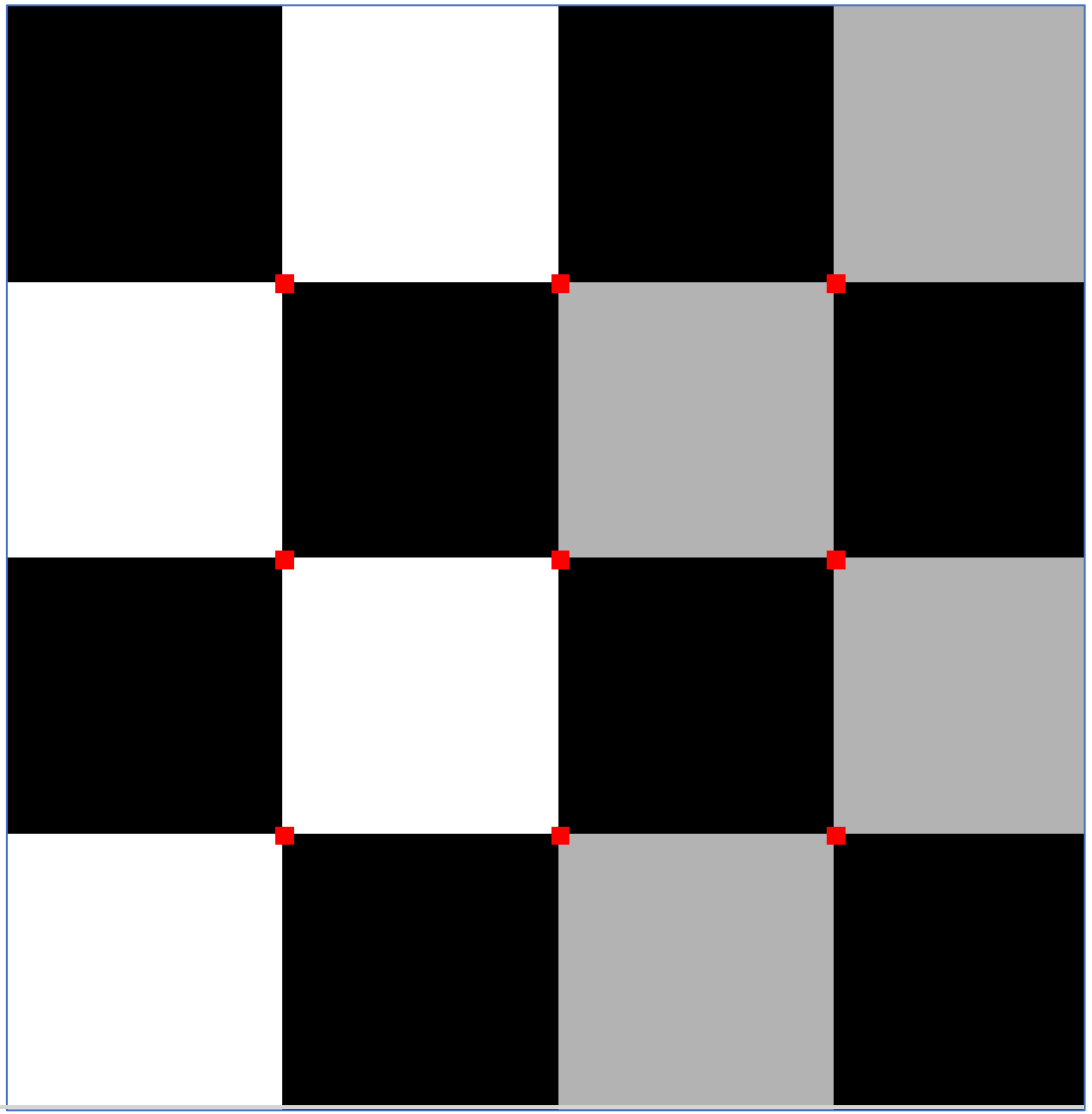
- The ideal keypoint is
  - Distinctive
    - Can differentiate between objects/object parts
  - Repetitive
    - Ideally
      - Invariant to image scaling
      - Invariant to rotation
      - Invariant to change in illumination
      - Invariant to 3D camera viewpoint

Algorithms for Keypoints Detection

- The are a number of keypoint detectors and descriptors
  - Harris-Stephens Corner
  - SIFT
  - SURF
  - ORB
  - and more

Harris-Stephens Corner

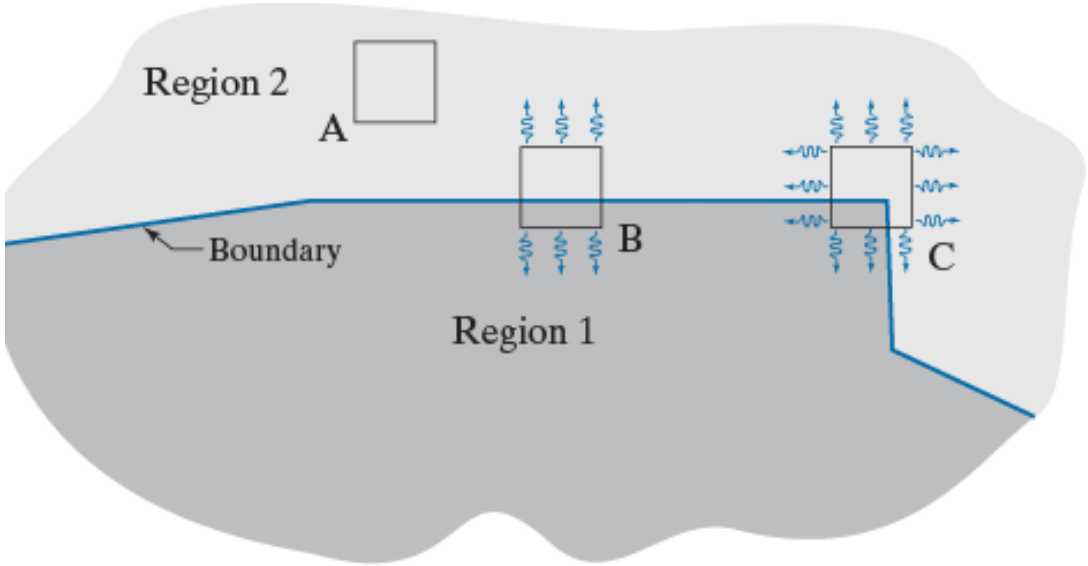
- Introduced by Chris Harris and Mike Stephens in 1988
- Corner – a rapid change of direction in a curve
  - We can think of it as the junction of two edges (a point whose local neighborhood stands in two dominant and different edge directions)
- Corner is invariant to rotation, translation and change in illumination



Harris-Stephens Corner

- Corners are detected by running a small window over an image
- The detector window is designed to compute intensity changes

FIGURE 11.45 Illustration of how the Harris-Stephens corner detector operates in the three types of sub-regions indicated by A (flat), B (edge), and C (corner). The wiggly arrows indicate graphically a directional response in the detector as it moves in the three areas shown.



- The HS corner detector attempts to differentiate between these three conditions

Harris-Stephens Corner

- How to differentiate between tree conditions?
  - shifting each window by a small amount in a given direction and measuring the amount of change that occurs in the pixel values
  - take the sum squared difference (SSD) of the pixel values before and after the shift and identifying pixel windows where the SSD is large for shifts.

$$C(x,y) = \sum_s \sum_t w(s,t)\left[f(s+x,t+y) - f(s,t)\right]^2$$

where $f(s,t)$ is a patch of an image centered in pixel (s,t), and $f(s+x,t+y)$ is the patch of the same size shifted by (x,y); $w(s,t)$ is a weighting function

Harris-Stephens Corner

$$C(x,y) = \sum_s \sum_t w(s,t)\left[f(s+x,t+y) - f(s,t)\right]^2$$

- The shifted patch can be approximated by the linear terms of a Taylor expansion

$$f(s+x,t+y) \approx f(s,t) + xf_x(s,t) + yf_y(s,t)$$

- We can then write

$$C(x,y) = \sum_s \sum_t w(s,t)\left[xf_x(s,t) + yf_y(s,t)\right]^2$$

- Maximum in $C(x,y)$ values indicates presence of corner

Harris-Stephens Corner

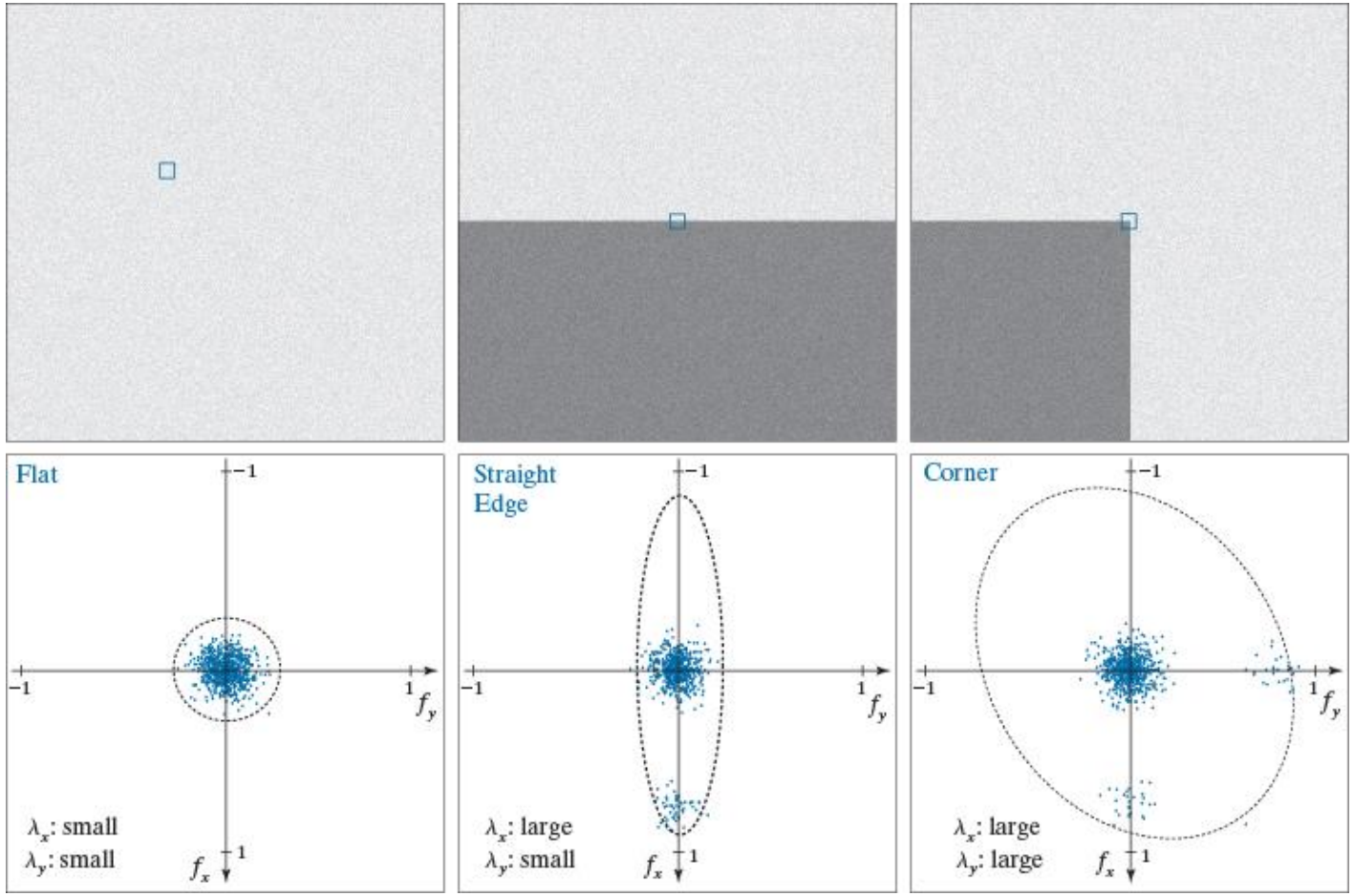
- The last equation can be written in matrix form as

$$C(x,y) = [x \;\; y]\,\mathbf{M}\begin{bmatrix} x \\ y \end{bmatrix}$$

$$\mathbf{M} = \sum_s \sum_t w(s,t)\,\mathbf{A} \qquad \mathbf{A} = \begin{bmatrix} f_x^2 & f_x f_y \\ f_x f_y & f_y^2 \end{bmatrix}$$

- M sometimes is called the Harris matrix
- The eigenvectors of M point in the direction of maximum increases in C, and their eigenvalues are proportional to the amount of increases

Harris-Stephens Corner

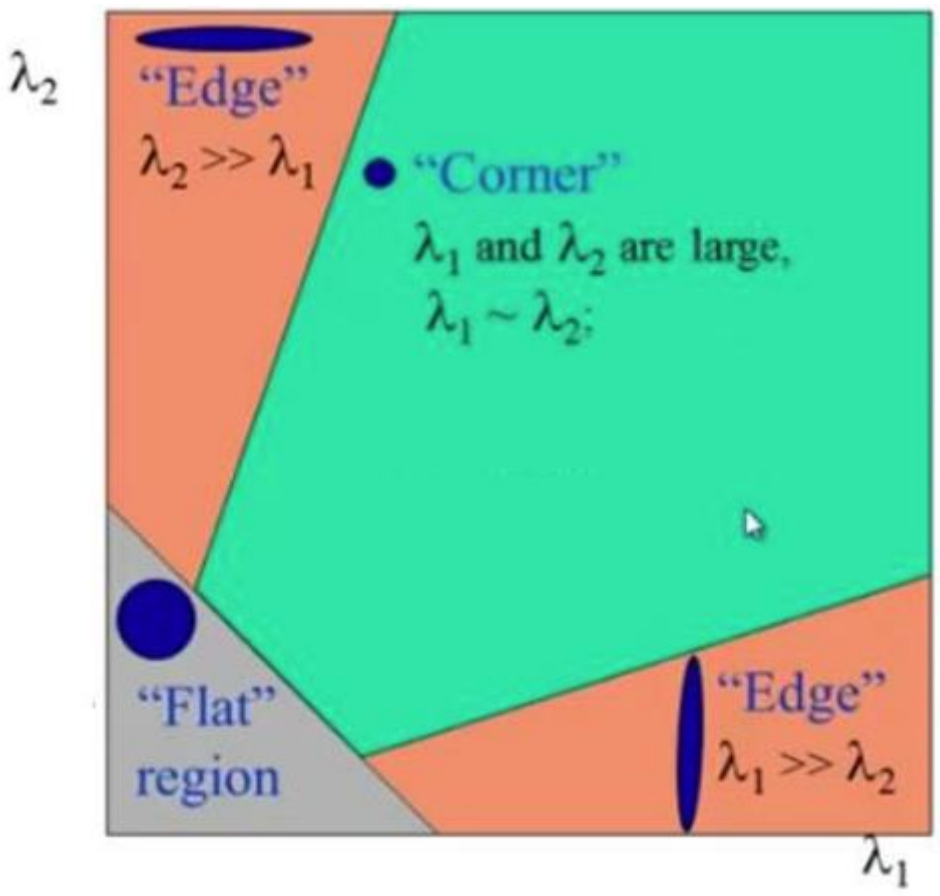


a b c
d e f

FIGURE 11.46 (a)–(c) Noisy images and image patches (small squares) encompassing image regions similar in content to those in Fig. 11.45. (d)–(f) Plots of value pairs (f_x, f_y) showing the characteristics of the eigenvalues of **M** that are useful for detecting the presence of a corner in an image patch.

Harris-Stephens Corner

1. Two small eigenvalues indicate “flat” region (nearly constant intensity)
2. One small and one large eigenvalue imply the presence of a vertical of horizonal edge
3. Two large eigenvalues imply the presence of a corner or (unfortunately) isolated points



Harris-Stephens Corner

- Eigenvalues are expensive to compute
- Instead, HS utilizes the following formula

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

$$\det M = \lambda_1 \lambda_2$$

$$trace\, M = \lambda_1 + \lambda_2$$

λ1 and λ2 are the eigenvalues of M

Harris-Stephens Corner

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

$$\det M = \lambda_1 \lambda_2$$

$$trace\, M = \lambda_1 + \lambda_2$$

- If $|R|$ is small => $\lambda 1$ and $\lambda 2$ are small => the region is flat.
- If $R<0$ => $\lambda 1 >> \lambda 2$ or vice versa => the region is an edge.
- If R is large => $\lambda 1$ and $\lambda 2$ are large and $\lambda 1 \sim \lambda 2$ => the region is a corner.
- k is a “sensitivity factor”; the smaller k is, the more likely the detector is to find corners (typically range from 0.04 - 0.06).
- Typically, R is used with a threshold T (a corner is detected if R > T) in that location

Harris-Stephens Corner – the pseudocode

1. Take the grayscale of the original image
2. Apply a Gaussian filter to smooth out any noise
3. Apply Sobel operator to find the x and y gradient values for every pixel in the grayscale image
4. For each pixel p in the grayscale image, consider a 3×3 window around it and compute the corner strength function. Call this its Harris value.
5. Find all pixels that exceed a certain threshold and are the local maxima within a certain window
6. For each pixel that meets the criteria in 5, compute a feature descriptor.

Harris-Stephens Corner – OpenCV implementation

dst=cv2.cornerHarris(src, blockSize, ksize, k[, dst[, borderType]])

Parameters

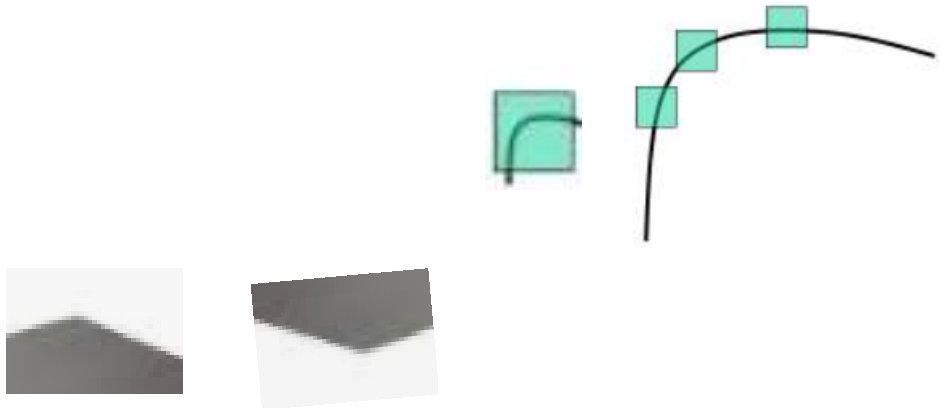
src	Input single-channel 8-bit or floating-point image.
dst	Image to store the Harris detector responses. It has the type CV_32FC1 and the same size as src .
blockSize	Neighborhood size (see the details on **cornerEigenValsAndVecs**).
ksize	Aperture parameter for the Sobel operator.
k	Harris detector free parameter. See the formula above.
borderType	Pixel extrapolation method. See **BorderTypes**. **BORDER_WRAP** is not supported.

Harris-Stephens Corner – OpenCV implementation

```
import cv2
import numpy as np


filename = "Pisa/IMG_7275.JPG"

img = cv2.imread(filename)

gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
gray = np.float32(gray)
dst = cv2.cornerHarris(gray,2,3,0.04)


# Threshold for an optimal value, it may vary depending
on the image.
img[dst>0.01*dst.max()]=[0,0,255]
cv2.imwrite("HarrisCorners.tif", img)
```

Harris-Stephens Corner – some properties

- Repetitive? **yes**
- Distinctive ? **yes**
- Invariant to image scaling? **no**
- Invariant to rotation? **yes**
- Invariant to change in illumination ? **Yes, up to some point**
- Invariant to 3D camera viewpoint ? **Yes, up to some point**



SIFT(Scale Invariant Feature Transform)

- Developed by Lowe, David G. "Distinctive image features from scale-invariant keypoints." International journal of computer vision 60.2 (2004): 91-110.
- It is called a transform because it transforms image data into scale-invariant coordinates relative to local image features
- The algorithm was patented. The patent expired in 2020.

SIFT – the main steps

1. **Scale-space peak selection:** Potential location for keypoints.
2. **Keypoint Localization:** Filter false keypoints.
3. **Orientation Assignment:** Assigning orientation to keypoints.
4. **Keypoint descriptor:** Describing the keypoints as a high dimensional vector.
5. **Keypoint Matching -** The matching is performed using nearest neighbors.

What is scale-space?

- Scale-space is a collection of images having different scales, generated from a single image
- To create a set of images of different scales, we take the original image and create its blur versions
- Then, reduce the scale by half. For each new image, we will create blur versions, and so on.

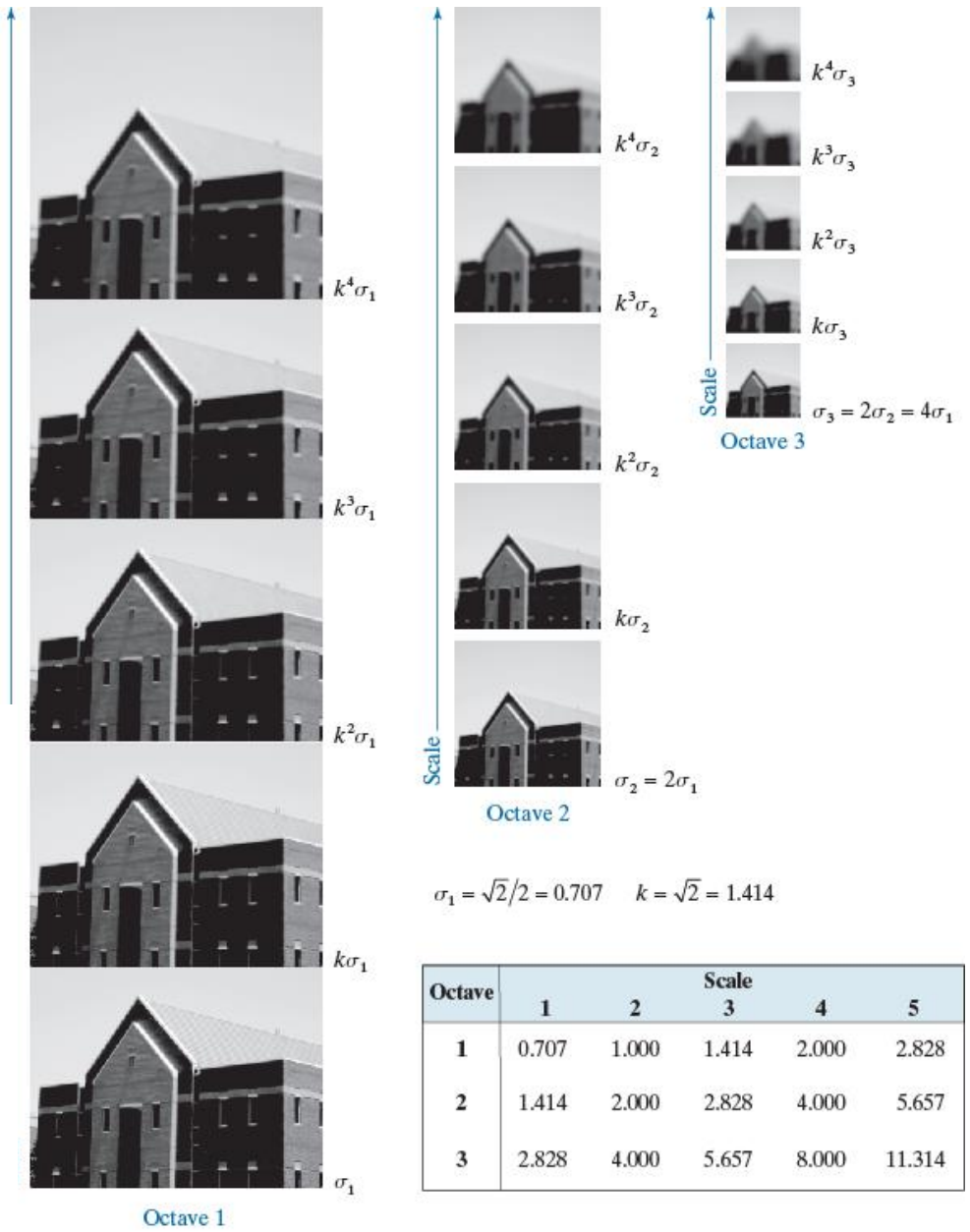


FIGURE 11.57 Illustration using images of the first three octaves of scale space in SIFT. The entries in the table are values of standard deviation used at each scale of each octave. For example the standard deviation used in scale 2 of octave 1 is $k\sigma_1$, which is equal to 1.0. (The images of octave 1 are shown slightly overlapped to fit in the figure space.)

$\sigma_1 = \sqrt{2}/2 = 0.707 \quad k = \sqrt{2} = 1.414$

Octave	Scale 1	2	3	4	5
1	0.707	1.000	1.414	2.000	2.828
2	1.414	2.000	2.828	4.000	5.657
3	2.828	4.000	5.657	8.000	11.314

What is scale-space?

- Scale-space is a multi-scale representation of an image
- The input image is successfully convolved with Gaussian kernels with std σ, $k\sigma, k^2\sigma, k^3\sigma, \ldots$
- The images are subdivided into octaves, each octave corresponds to a doubling of σ
- Each octave's image size is half the previous one.

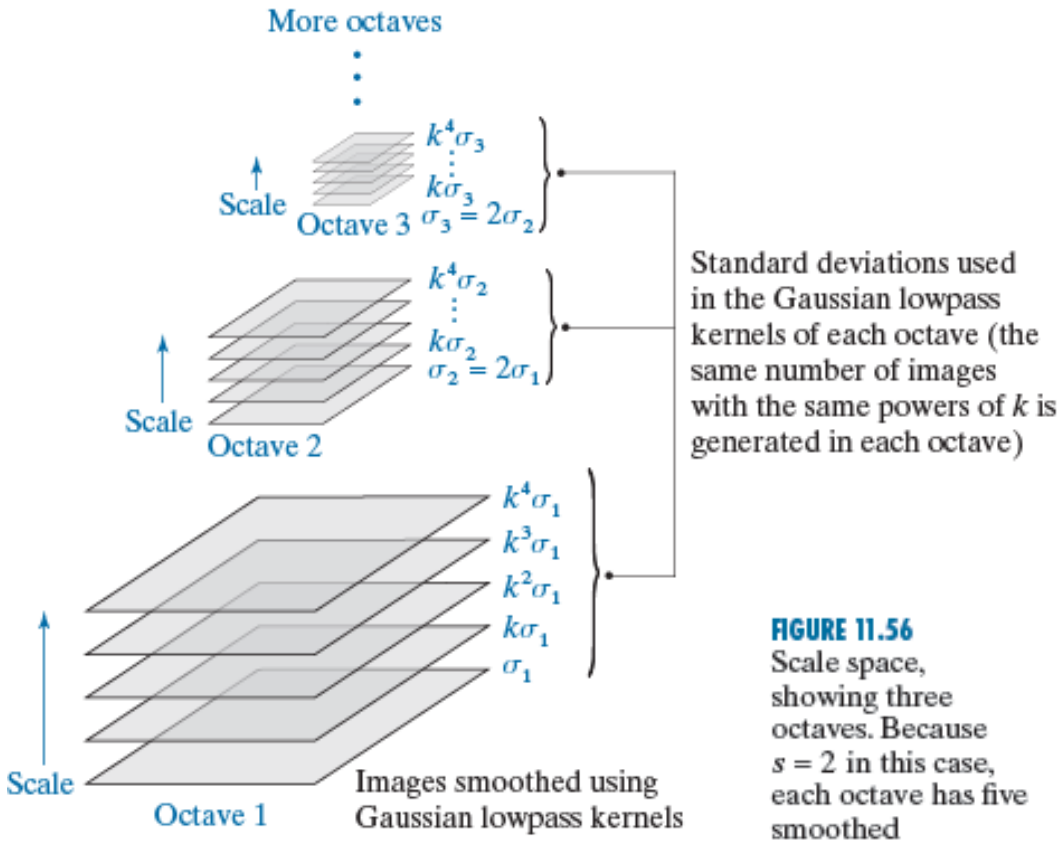


FIGURE 11.56 Scale space, showing three octaves. Because $s = 2$ in this case, each octave has five smoothed images. A Gaussian kernel was used for smoothing, so the space parameter is σ.

SIFT- Finding the keypoints

- Gaussian filtered images are used to find the locations of keypoints
- The difference of Gaussian (DoG) is obtained as the difference between two adjacent images in octave
- DoG is a feature enhancement algorithm that involves the subtraction of one blurred version of an original image from another, less blurred version of the original image

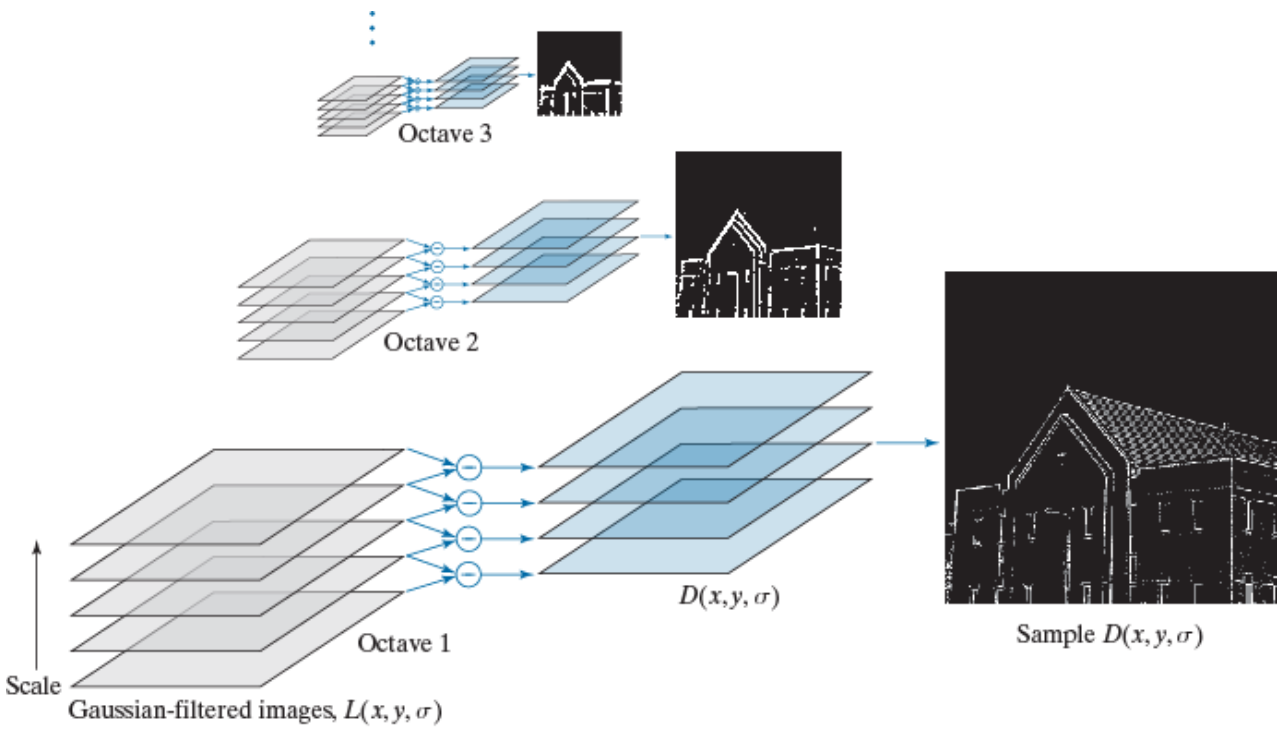


FIGURE 11.58 How Eq. (11-69) is implemented in scale space. There are $s+3$ $L(x, y, \sigma)$ images and $s+2$ corresponding $D(x, y, \sigma)$ images in each octave.

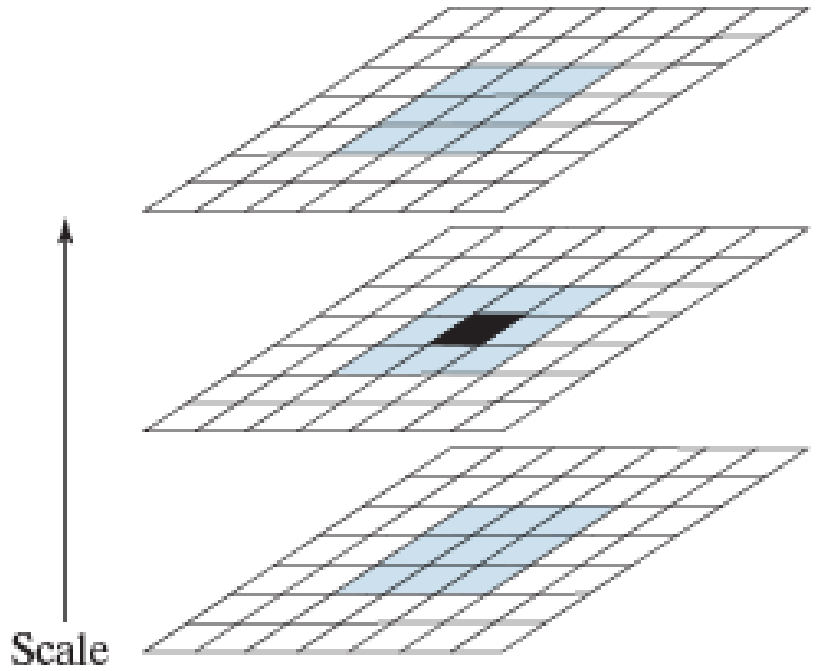
SIFT – Finding the keypoints

- The next step is to find the important keypoints from the image that can be used for feature matching
- **The idea is to find the local maxima and minima for the images.**
  - Find the local maxima and minima
  - Remove low contrast keypoints

SIFT- Finding the keypoints

- Pixel in an image is compared with its 8 neighbors as well as 9 pixels in the next scale and 9 pixels in previous scales.
- If it is a local extrema, it is a potential keypoint.

FIGURE 11.59 Extrema (maxima or minima) of the $D(x, y, \sigma)$ images in an octave are detected by comparing a pixel (shown in black) to its 26 neighbors (shown shaded) in 3×3 regions at the current and adjacent scale images.



Corresponding sections of three contiguous $D(x, y, \sigma)$ images

SIFT- Keypoint selection

- Eliminate the keypoints that have low contrast or edge keypoints
- Low contrast keypoints - a second-order Taylor expansion is computed for each keypoint. If its magnitude value is less than 0.03, the keypoint is rejected.
- Edge keypoints - second-order Hessian matrix is used to identify such keypoints.

SIFT- Eliminating false keypoints

FIGURE 11.60
SIFT keypoints detected in the building image. The points were enlarged slightly to make them easier to see.

SIFT- Orientation Assignment

- We know the scale of a keypoint
- We will now assign an orientation to each of these keypoints so that they are invariant to rotation
- Select a neighborhood around a point, for each pixel calculate the gradient and orientation
- Build an orientation histogram
- The highest peak in the histogram is the orientation of the keypoint
  - The highest pick is taken and any peak above 80% of it is also considered to calculate the orientation.
  - It creates keypoints with same location and scale, but different directions

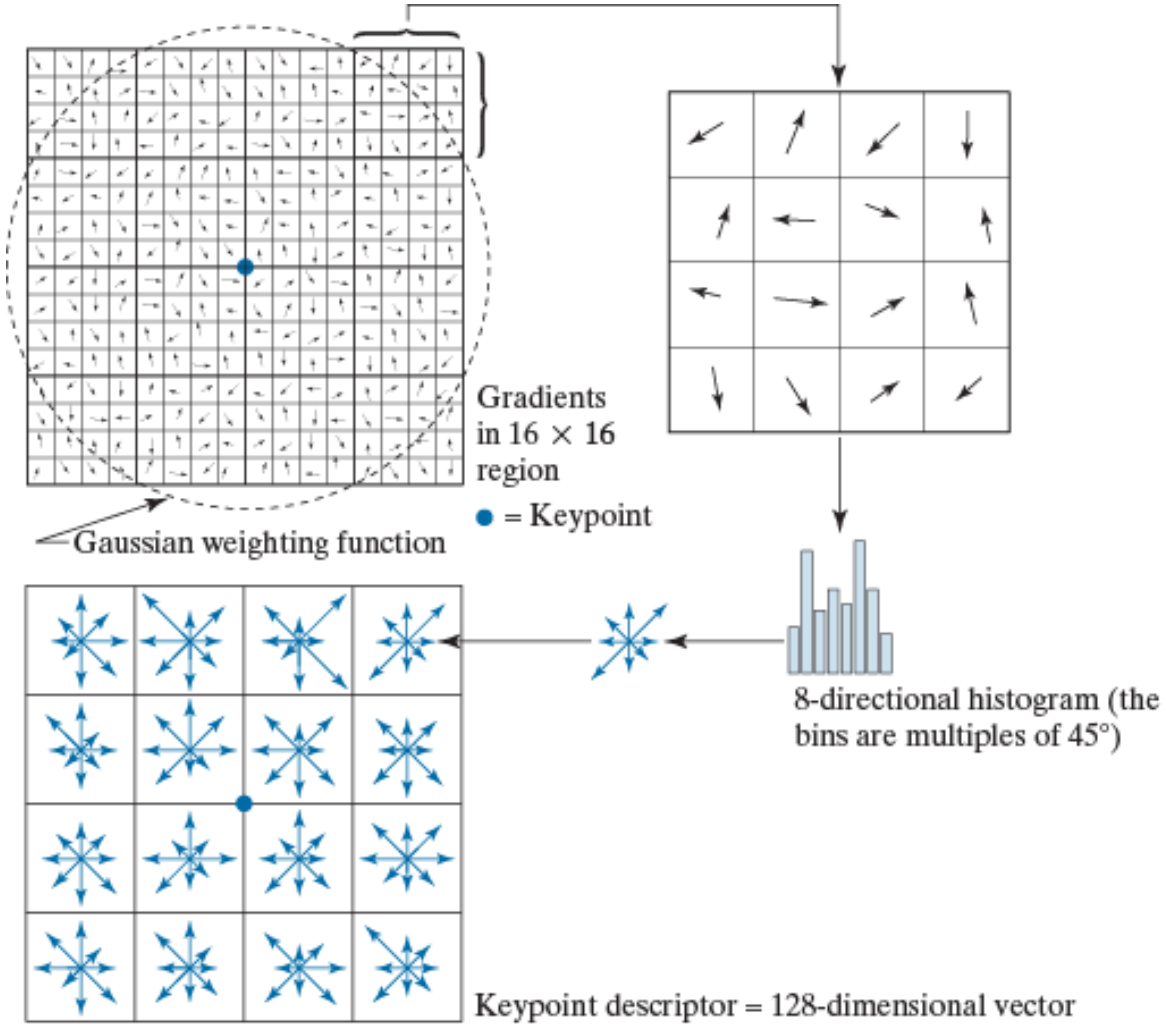
SIFT- Orientation Assignment

FIGURE 11.61
The keypoints from Fig. 11.60 superimposed on the original image. The arrows indicate keypoint orientations.

SIFT- Keypoint Descriptor

- A16x16 window around the keypoint is taken. It is divided into 16 sub-blocks of 4x4 size.
- For each sub-block, 8 bin orientation histogram is created
- To achieve rotation independence, the keypoint's rotation is subtracted from each orientation
- The size of the final vector is 4 x 4 x 8 = 128
- The vector is normalized to achive illumination independence

FIGURE 11.62 Approach used to compute a keypoint descriptor.



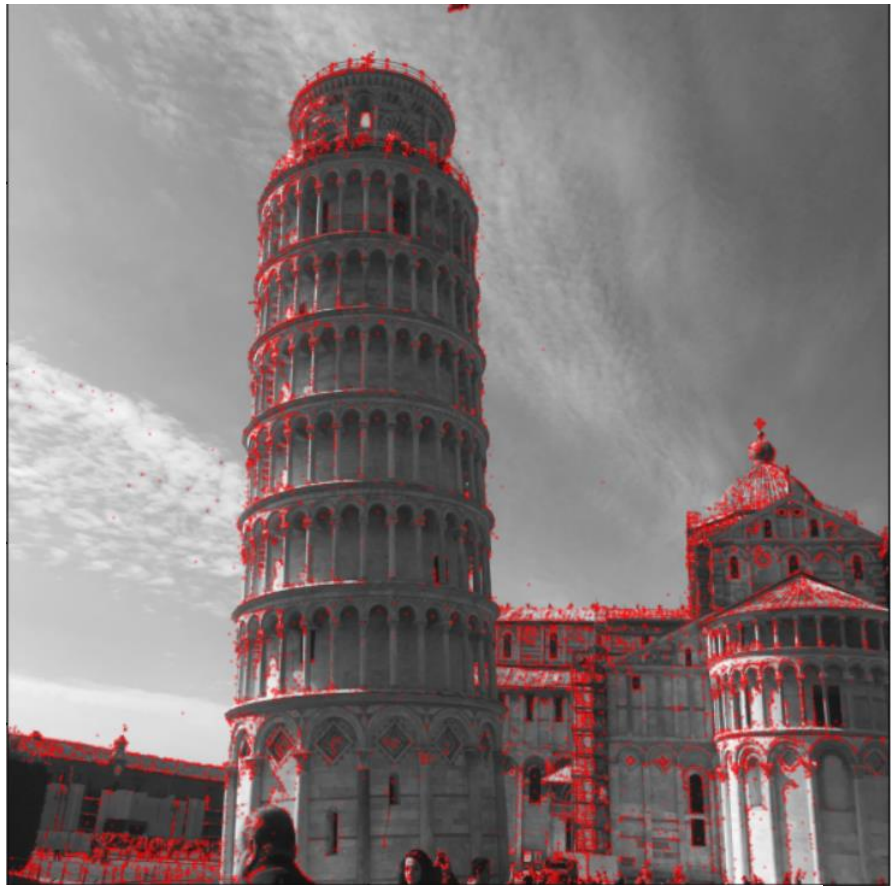
SIFT – OpenCV implementation

cv2.SIFT_create([, nfeatures[, nOctaveLayers[, contrastThreshold[, edgeThreshold[, sigma]]]]])

- **nfeatures** The number of best features to retain. The features are ranked by their scores (measured in SIFT algorithm as the local contrast)
- **nOctaveLayers** The number of layers in each octave. 3 is the value used in D. Lowe paper. The number of octaves is computed automatically from the image resolution.
- **contrastThreshold** The contrast threshold used to filter out weak features in semi-uniform (low-contrast) regions. The larger the threshold, the less features are produced by the detector. Note: The contrast threshold will be divided by nOctaveLayers when the filtering is applied. When nOctaveLayers is set to default and if you want to use the value used in D. Lowe paper, 0.03, set this argument to 0.09.
- **edgeThreshold** The threshold used to filter out edge-like features. Note that the its meaning is different from the contrastThreshold, i.e. the larger the edgeThreshold, the less features are filtered out (more features are retained).

SIFT – OpenCV implementation

```
import cv2
import matplotlib.pyplot as plt

filename = "Pisa/IMG_7276.JPG"
image = cv2.imread(filename)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

sift = cv2.SIFT_create()
(kps, descs) = sift.detectAndCompute(gray, None)

# draw kepoints in red on the original image
img_1 = cv2.drawKeypoints(gray,kps,image, color = [255,0,0])
```



- Each keypoint is a special structure which has many attributes like its (x,y) coordinates, size of the meaningful neighborhood, angle which specifies its orientation, response that specifies strength of keypoints etc.
- descs is a numpy array of shape (Number of Keypoints)×128.

SIFT – OpenCV implementation

```
img_1 = cv2.drawKeypoints(gray, kps, image,
                          flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS
```

OpenCV also provides **cv.drawKeyPoints()** function which draws the small circles on the locations of keypoints. If you pass a flag, **cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS** to it, it will draw a circle with size of keypoint and it will even show its orientation

Keypoints Matching

- Basic brute-force matcher
- Brute-Force matcher is simple. It takes the descriptor of one feature in first set and is matched with all other features in second set using some distance calculation. And the closest one is returned.

SIFT – OpenCV implementation

```
img1 = cv2.imread('Pisa/IMG_7326.JPG')
img2 = cv2.imread('Pisa/IMG_7328.JPG')

img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

#sift
sift = cv2.SIFT_create()

keypoints_1, descriptors_1 = sift.detectAndCompute(img1,None)
keypoints_2, descriptors_2 = sift.detectAndCompute(img2,None)

#feature matching
bf = cv2.BFMatcher(cv2.NORM_L1, crossCheck=True)

matches = bf.match(descriptors_1,descriptors_2)
matches = sorted(matches, key = lambda x:x.distance)
#draw best 50 matches
img3 = cv2.drawMatches(img1, keypoints_1, img2, keypoints_2,
                       matches[:50], img2, flags=2)
plt.imshow(img3),plt.show()
```

SIFT - Keypoint Matching

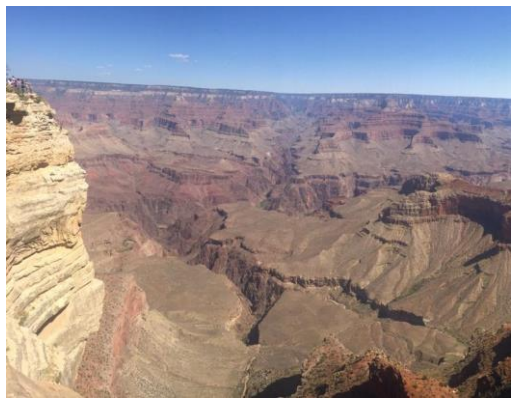
Figure 12. The training images for two objects are shown on the left. These can be recognized in a cluttered image with extensive occlusion, shown in the middle. The results of recognition are shown on the right. A parallelogram is drawn around each recognized object showing the boundaries of the original training image under the affine transformation solved for during recognition. Smaller squares indicate the keypoints that were used for recognition.

SIFT - Keypoints Matching

Figure 13. This example shows location recognition within a complex scene. The training images for locations are shown at the upper left and the 640 × 315 pixel test image taken from a different viewpoint is on the upper right. The recognized regions are shown on the lower image, with keypoints shown as squares and an outer parallelogram showing the boundaries of the training images under the affine transform used for recognition.

Summary

- The are a number of keypoints detectors and descriptors
- We have seen Harris-Stephens Corner and SIFT
- Assignment 3 – use kepoints detection to create panorama from two images



+

=