BSC~Microsoft Research Centre

Computer Vision Features Extraction

Tassadaq Hussain

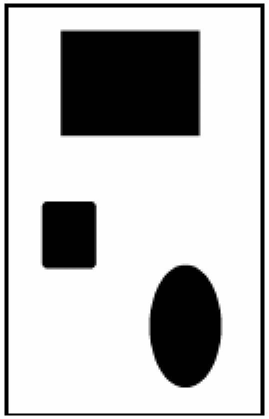
Associate Prof: Riphah Int'l University
Microsoft Barcelona Supercomputing Center Barcelona, Spain
UCERD Pvt Ltd Islamabad

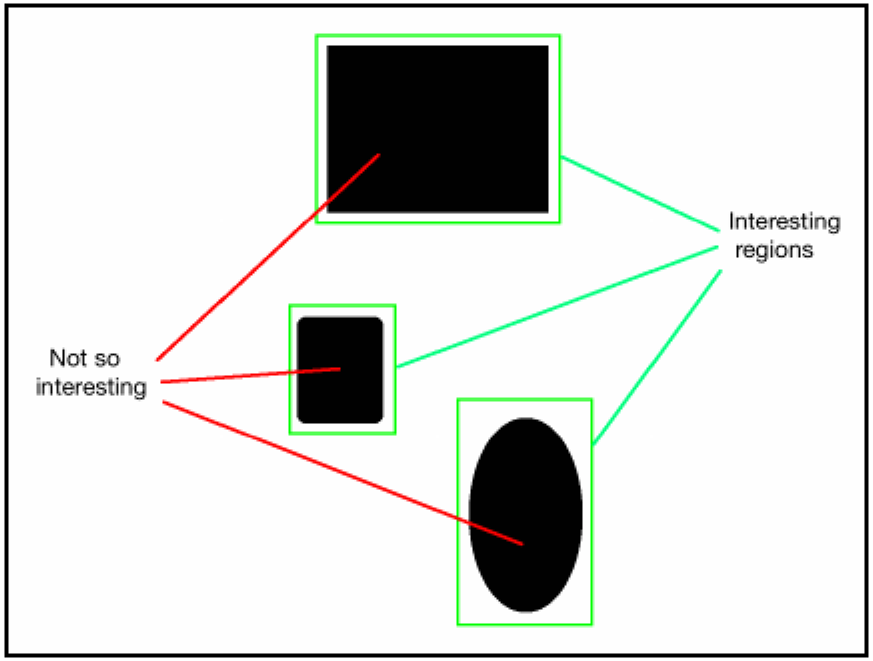
UCERD
Gathering Intellectuals
www.ucerd.com

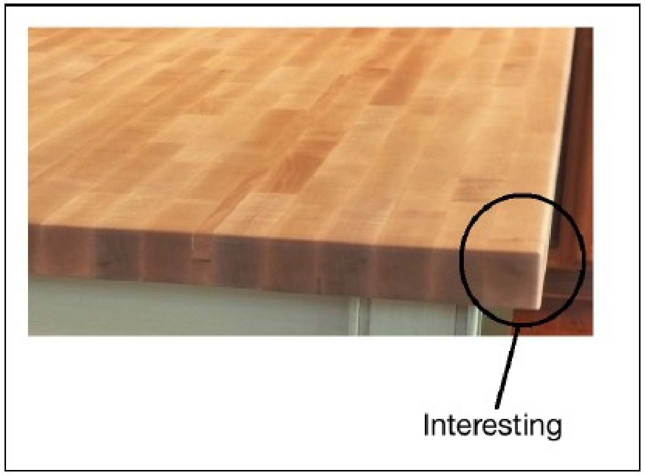
Key Points

Image content analysis refers to the process of understanding the content of an image so that we can take some action based on that. Let's take a step back and talk about how humans do it. Our brain is an extremely powerful machine that can do complicated things very quickly. When we look at something, our brain automatically creates a footprint based on the interesting aspects of that image.

What are keypoints?





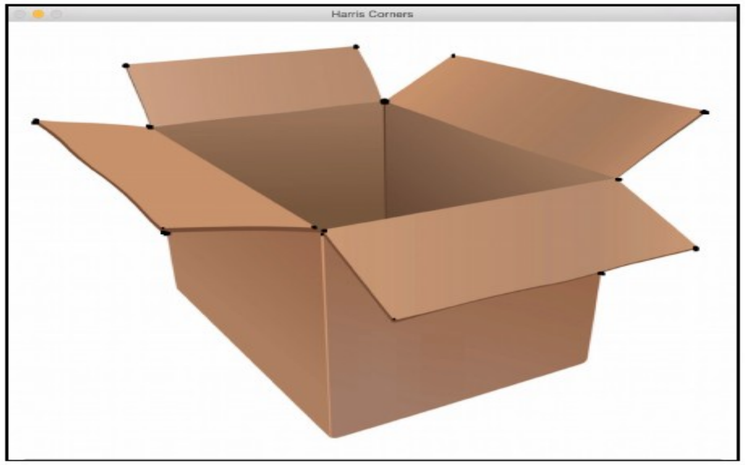


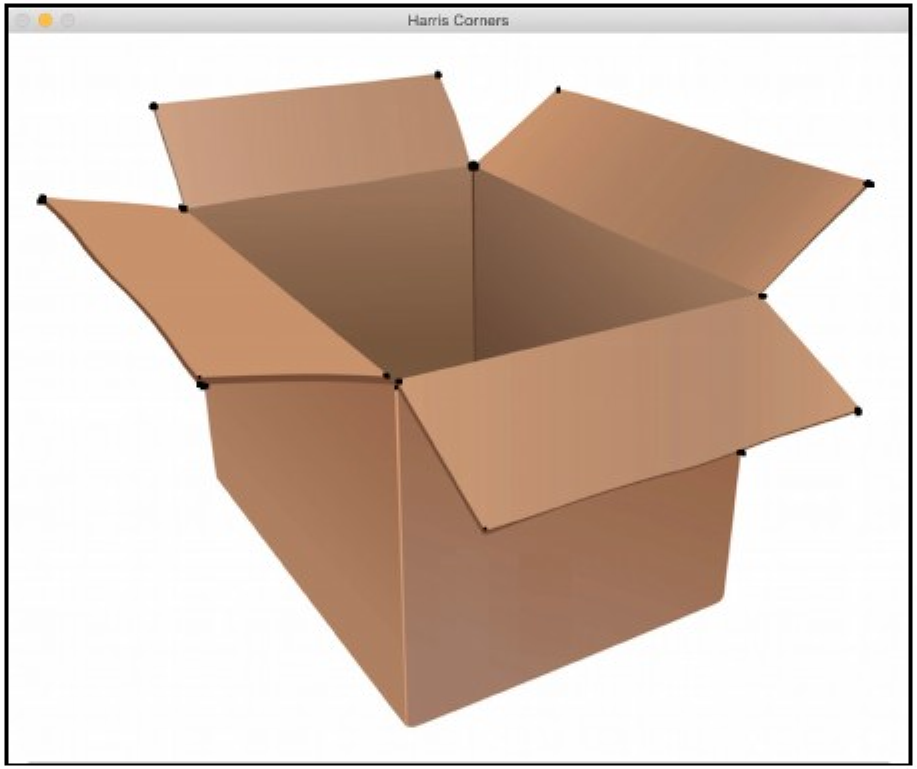
Detecting the corners

In computer vision, there is a popular corner detection technique called the Harris Corner Detector.

We basically construct a 2x2 matrix based on partial derivatives of the grayscale image, and then analyze the eigenvalues obtained. Eigenvalues are a special set of scalars associated with a linear system of equations that provide segmented information about the image by a cluster of pixels that belong together. In this case, we use them to detect the corners. This is actually an oversimplification of the actual algorithm, but it covers the gist.

```
import cv2
import numpy as np
img = cv2.imread('./images/box.png')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
gray = np.float32(gray)
# To detect only sharp corners
dst = cv2.cornerHarris(gray, blockSize=4, ksize=5, k=0.04)
# Result is dilated for marking the corners
dst = cv2.dilate(dst, None)
# Threshold for an optimal value, it may vary depending on the image
img[dst > 0.01*dst.max()] = [0,0,0]
cv2.imshow('Harris Corners(only sharp)',img)
# to detect soft corners
dst = cv2.cornerHarris(gray, blockSize=14, ksize=5, k=0.04)
dst = cv2.dilate(dst, None)
img[dst > 0.01*dst.max()] = [0,0,0]
cv2.imshow('Harris Corners(also soft)',img)
cv2.waitKey()
```




Harris Corners

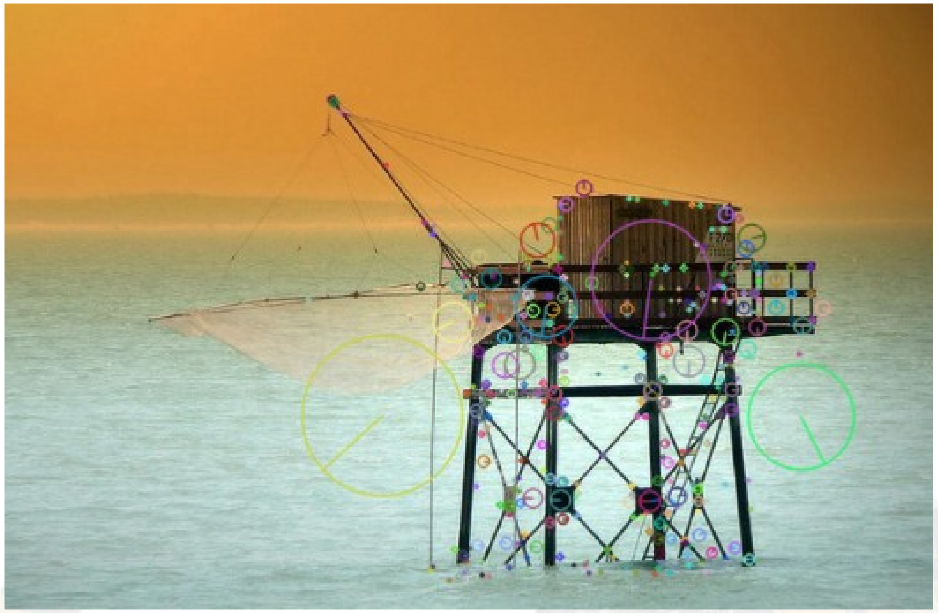
Good features to track

```
import cv2
import numpy as np
img = cv2.imread('images/box.png')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
corners = cv2.goodFeaturesToTrack(gray, maxCorners=7, qualityLevel=0.05,
minDistance=25)
corners = np.float32(corners)
for item in corners:
x, y = item[0]
cv2.circle(img, (x,y), 5, 255, -1)
cv2.imshow("Top 'k' features", img)
cv2.waitKey()
```

Scale-invariant feature transform (SIFT)

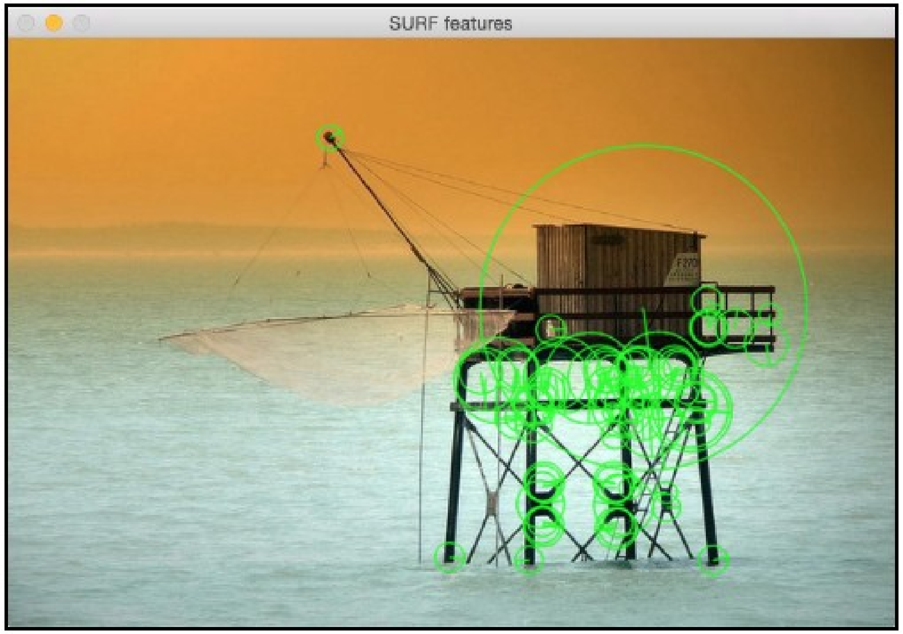
SIFT builds a pyramid by downsampling an image and taking the difference of Gaussian.

```
import cv2
import numpy as np
input_image = cv2.imread('images/fishing_house.jpg')
gray_image = cv2.cvtColor(input_image, cv2.COLOR_BGR2GRAY)
# For version opencv < 3.0.0, use cv2.SIFT()
sift = cv2.xfeatures2d.SIFT_create()
keypoints = sift.detect(gray_image, None)
cv2.drawKeypoints(input_image, keypoints, input_image, \
flags = cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
cv2.imshow('SIFT features', input_image)
cv2.waitKey()
```



Speeded-up robust features (SURF)

SURF uses a simple box filter to approximate the Gaussian. The good thing is that this is really easy to compute and it's reasonably fast.



UCERD
Gathering Intellectuals
www.ucerd.com

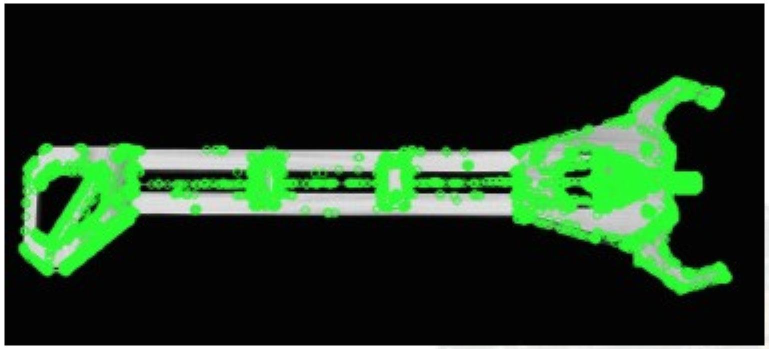
BSC~Microsoft Research Centre

```
import cv2
import numpy as np
input_image = cv2.imread('images/fishing_house.jpg')
gray_image = cv2.cvtColor(input_image, cv2.COLOR_BGR2GRAY)
# For version opencv < 3.0.0, use cv2.SURF()
surf = cv2.xfeatures2d.SURF_create()
# This threshold controls the number of keypoints
surf.setHessianThreshold(15000)
keypoints, descriptors = surf.detectAndCompute(gray_image, None)
cv2.drawKeypoints(input_image, keypoints, input_image, color=(0,255,0),\
flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
cv2.imshow('SURF features', input_image)
cv2.waitKey()
```

Features from Accelerated Segment Test (FAST)

Even though SURF is faster than SIFT, it's just not fast enough for a real-time system, especially when there are resource constraints. When you are building a real-time application on a mobile device, you won't have the luxury of using SURF to do computations in real time.

We need something that's really fast and computationally inexpensive. Hence, Rosten and Drummond came up with FAST. As the name indicates, it's really fast!



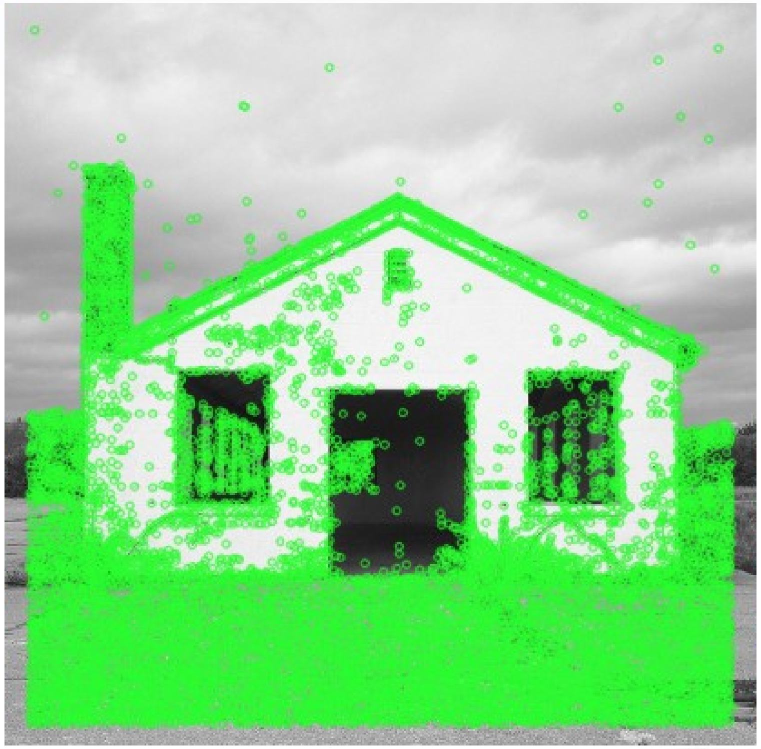
UCERD
Gathering Intellectuals
www.ucerd.com

BSC~Microsoft Research Centre

```
import cv2
import numpy as np
input_image = cv2.imread('images/tool.png')
gray_image = cv2.cvtColor(input_image, cv2.COLOR_BGR2GRAY)
# Version under opencv 3.0.0 cv2.FastFeatureDetector()
fast = cv2.FastFeatureDetector_create()
# Detect keypoints
keypoints = fast.detect(gray_image, None)
print("Number of keypoints with non max suppression:", len(keypoints))
# Draw keypoints on top of the input image
img_keypoints_with_nonmax=input_image.copy()
cv2.drawKeypoints(input_image, keypoints, img_keypoints_with_nonmax,
color=(0,255,0), \ flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
cv2.imshow('FAST keypoints - with non max suppression',
img_keypoints_with_nonmax)
# Disable nonmaxSuppression
fast.setNonmaxSuppression(False)
# Detect keypoints again
keypoints = fast.detect(gray_image, None)
print("Total Keypoints without nonmaxSuppression:", len(keypoints))
# Draw keypoints on top of the input image
img_keypoints_without_nonmax=input_image.copy()
cv2.drawKeypoints(input_image, keypoints, img_keypoints_without_nonmax,
color=(0,255,0), \ flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
cv2.imshow('FAST keypoints - without non max suppression',
img_keypoints_without_nonmax)
cv2.waitKey()
```

Binary Robust Independent Elementary Features (BRIEF)

Even though we have FAST to quickly detect the keypoints, we still have to use SIFT or SURF to compute the descriptors. We need a way to quickly compute the descriptors as well. This is where BRIEF comes into the picture. BRIEF is a method for extracting feature descriptors. It cannot detect the keypoints by itself, so we need to use it in conjunction with a keypoint detector. The good thing about BRIEF is that it's compact and fast.



UCERD
Gathering
Intellectuals
www.ucerd.com

BSC~Microsoft Research Centre

```
import cv2
import numpy as np
input_image = cv2.imread('images/house.jpg')
gray_image = cv2.cvtColor(input_image, cv2.COLOR_BGR2GRAY)
# Initiate FAST detector
fast = cv2.FastFeatureDetector_create()
# Initiate BRIEF extractor, before opencv 3.0.0 use
cv2.DescriptorExtractor_create("BRIEF")
brief = cv2.xfeatures2d.BriefDescriptorExtractor_create()
# find the keypoints with STAR
keypoints = fast.detect(gray_image, None)
# compute the descriptors with BRIEF
keypoints, descriptors = brief.compute(gray_image, keypoints)
cv2.drawKeypoints(input_image, keypoints, input_image, color=(0,255,0))
cv2.imshow('BRIEF keypoints', input_image)
cv2.waitKey()
```

Oriented FAST and Rotated BRIEF (ORB)

So, now we have arrived at the best combination out of all the combinations that we have discussed so far. This algorithm came out of the OpenCV Labs. It's fast, robust, and open source! The SIFT and SURF algorithms are both patented and you can't use them for commercial purposes; this is why ORB is good in many ways.

```
import cv2
import numpy as np
input_image = cv2.imread('images/fishing_house.jpg')
gray_image = cv2.cvtColor(input_image, cv2.COLOR_BGR2GRAY)
# Initiate ORB object, before opencv 3.0.0 use cv2.ORB()
orb = cv2.ORB_create()
# find the keypoints with ORB
keypoints = orb.detect(gray_image, None)
# compute the descriptors with ORB
keypoints, descriptors = orb.compute(gray_image, keypoints)
# draw only the location of the keypoints without size or orientation
cv2.drawKeypoints(input_image, keypoints, input_image, color=(0,255,0))
cv2.imshow('ORB keypoints', input_image)
cv2.waitKey()
```