Objectives

In this chapter, you will able learn the following,

- Introduction to SIFT
- Motivation and background
- SIFT algorithm
- Applications

Prerequisites

For the better understanding of this chapter, following are some prerequisities but not mandatory.

- Basic ideas of computer vision
- Basic calculus

Introduction

Let's recall the example you have already seen in the first chapter about creating the panorama from two images. Since both images are on the same scale and same orientation, we could easily create a panorama.

However, in a real-world application, the problem may go much harder. What if those images are of different scaling and orientation or rotations? Identifying the matching keypoints will be much harder, right? The formulation of these problems can be shown in the images below.

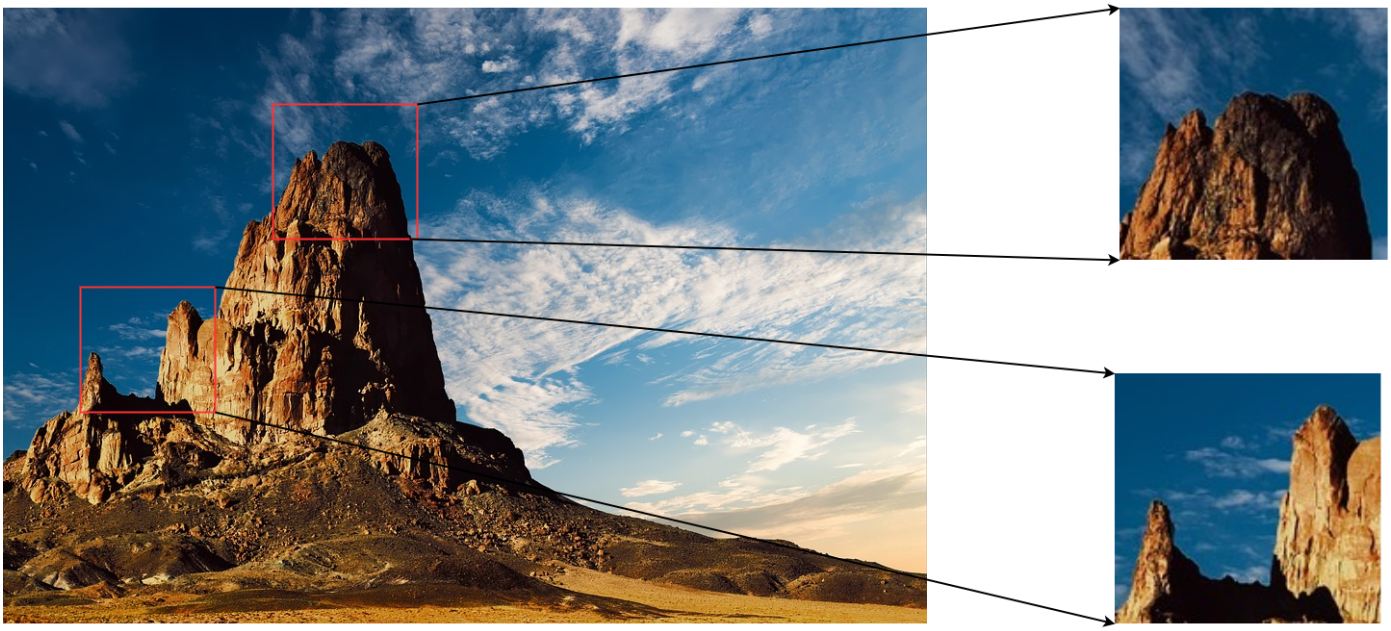


Fig. 1: Images with different scaling

In the above images, the features of right-side images are different from those of the left side due to scaling, although they are the same features. Similarly, the figure below represents the features matching problem due to rotation.

Fig. 2: Images with different rotation

Hence, we need a robust tool that is invariant to these scaling and rotations problems while matching the detected features of the images. Here comes a handy algorithm called SIFT (Scale Invariant Feature Transform), which is invariant to both the scaling and rotations.

By definition, Scale Invariant Feature Transform (SIFT) is a feature detection algorithm that falls under the computer vision to detect and describe local features in an image. We will explain this algorithm in detail in the following sections.

Background and Motivation

SIFT algorithm was first published in a paper (https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf) by David Lowe in 1999 and patented by the University of British Columbia in Canada. This algorithm solves the drawbacks of the Harris corner detector. We are already familiar with the Harris corner detector algorithm. One problem with this algorithm is it is not scale-invariant. To better understanding, consider the following image.

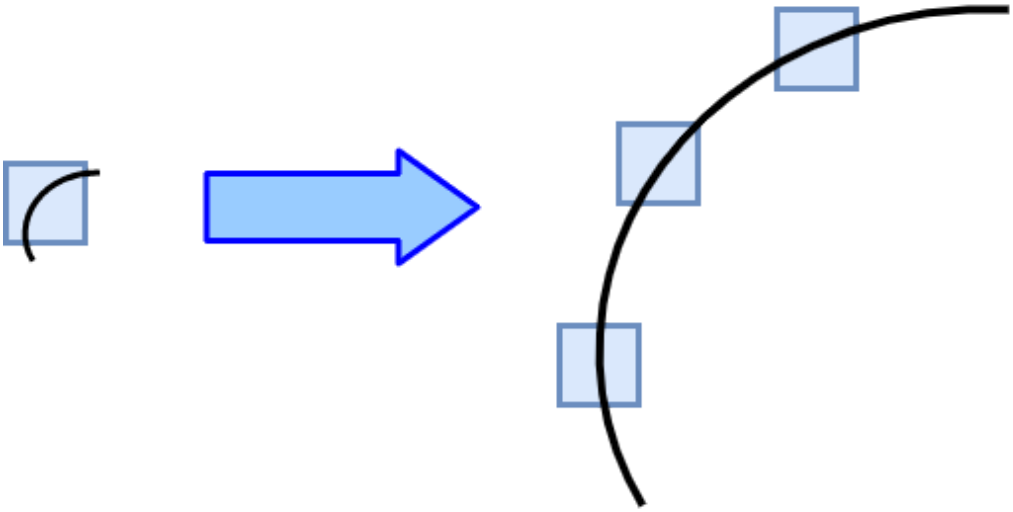


Fig. 3: Drawback of Harris corner detector

According to the Harris corner algorithm, the image before scaling treated as a corner. However, after scaling, the same image treated as an edge. Hence, we need a robust algorithm to address this problem of scaling, and the SIFT algorithm does the same.

Although Harris corner is rotation invariant, SIFT is invariant to both scaling and rotation. Hence, SIFT turns out to more robust feature descriptor algorithm by eliminating the Harris corner detector's weakness.

SIFT Algorithm

Let's talk about the core of this reading material, i.e., SIFT algorithm. The following two flowcharts gives the overview of the SIFT algorithm. The left algorithm is a little more in depth than the right side one. We will discuss each step in detail in the following section.

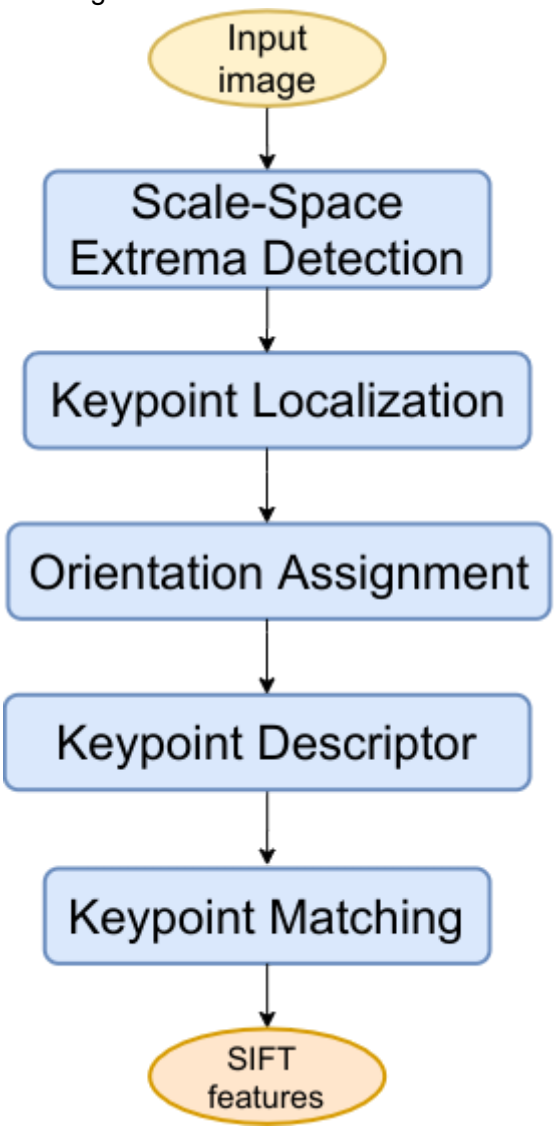


Fig. 4: Flowchart of SIFT procedure

1. Scale-Space Extrema Detection
    - From Harris corner detection limitation, we can't use the same window to detect keypoints of images with different scales. Hence, we need a space-scale filter.
    - Laplacian of Gaussian (LoG) operator acts as a blob detector with different σ, where σ is a scaling parameter.
    - Changing in σ gives different sizes of blob detector. So, we can find potential keypoints at (x, y) with σ.
    - But, LoG is computationally very expensive. Hence, SIFT uses Difference of Gaussian (DoG) approximation of LoG to reduce this complexity.
    - The DoG can be obtained by differencing two different Gaussian blurrings of an image, as shown in the figure below.

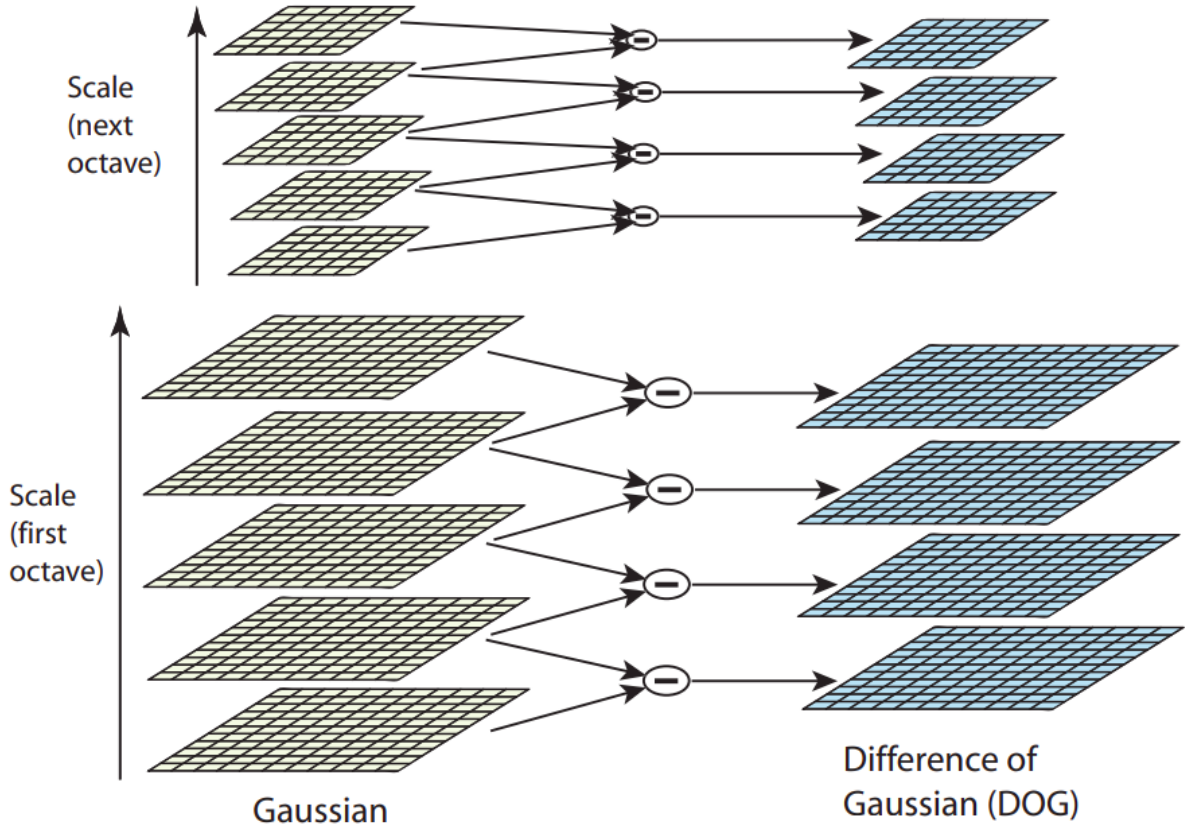


Fig. 5: Showing Difference of Gaussian (DoG)

- Once the DoG is found, local extrema of an image can be found by comparing 8 neighbors of its own and 9 neighbors of previous and next scale images.
- Finally, potential key points are the local extrema of an image, which is the best representation in that scale.

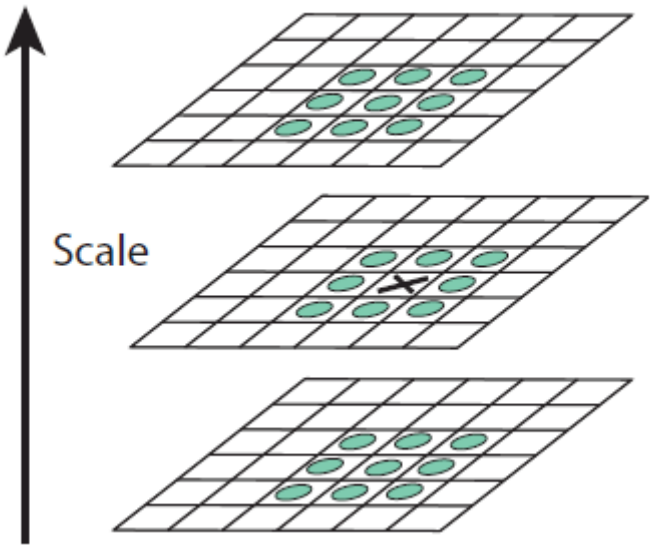


Fig. 6: Comparing single pixel to neighboring pixels of different scales

2. Keypoint Localization

    - The potential key points with intensity higher than the threshold value and robust to various noises and illumination are selected for further processing.
    - The key points near to the edges are also removed due to ambiguity in the image matching problem.
    - The algorithm here uses a Talor's series expansion of scale-space for finding the accurate location of potential keypoints.
3. Orientation Assignment

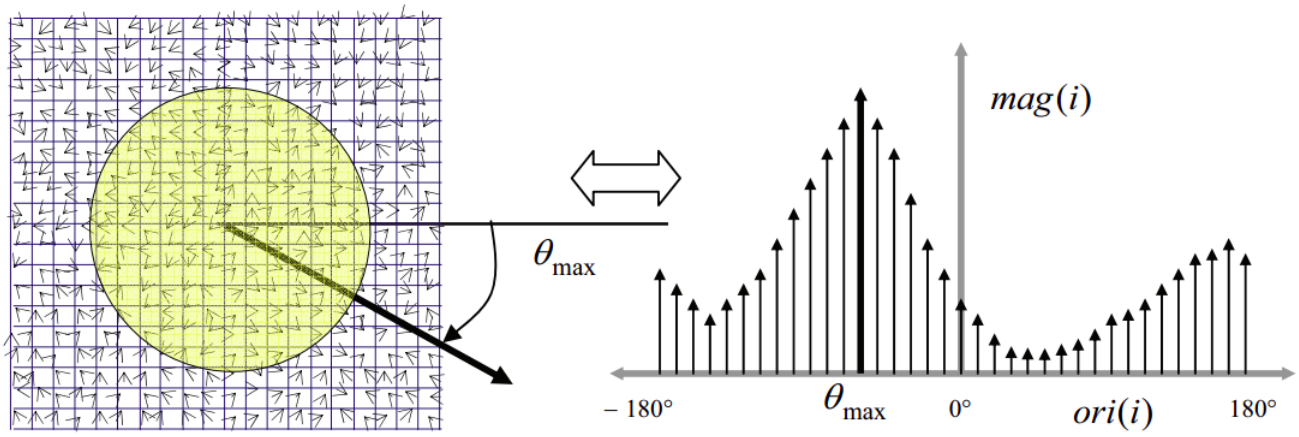


Fig. 7: Orientation of pixel and respective histogram

- Up to now, images are robust to the scaling but not to the rotation. The orientation assignment assigns an orientation to each of the interest points that combine with scale to produce scale and rotation invariant coordinate system for the descriptor vector.
- In this approach, each keypoints are assigned with some orientation, and a histogram of orientation is created.
- This assignment creates a keypoints with same location and scale but different directions.

4. Keypoint Descriptor

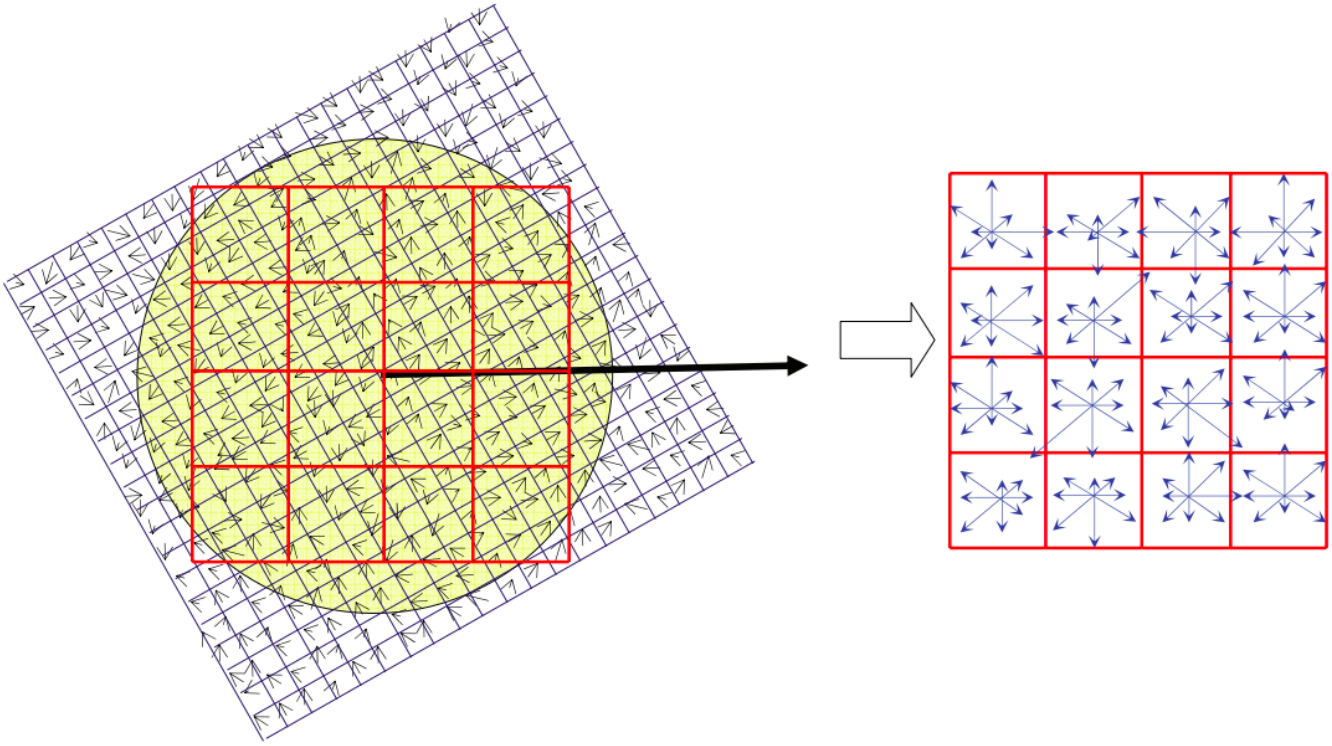


Fig. 8: Each keypoint descriptor vector

- The keypoint descriptors are the vector achieved from orientation histogram with the size of (128 * number of keypoints).
- Then, different normalization techniques are used to achieve invariance against illumination changes.

5. Keypoint Matching
    - Identifying the nearest neighbors helps is a keypoint matching of the two images.
    - There are different approaches used by the SIFT algorithm for matching the features. Some of them are,
        - SIFT Correspondences Search
        - Mismatches Discarding

Well, we descibed all the steps of SIFT algorithm. However, it is an iterative approach. The process will repeated until keypoint descritors are of all scale and for all octaves. Otherwise, the process will redirected to DoG generation again.

Applications

Since, SIFT finds the destinctive keypoints that are invariant to scale, rotation, location, and robust to changes in illuminations and various affine transformations, applications of the SIFT algorithm in the computer vision field are limitless. Some of them where SIFT algorithm used, are,

- Object Recognition
- Robotic Mapping
- Image stitching
- 3D Modeling
- Gesture Recognition
- Video Tracking
- Individual Identification of moving objects

For all applications mentioned above, first, SIFT features are obtained from an input image by applying the SIFT algorithm. These features can be used for various matching purposes. For eg., in image stitching applications, the features of two images can compared and merged accordingly. In the same way, these features are helpful for object recognition, 3D modeling, video tracking, etc.

SIFT Implementation

Let's implement the SIFT algorithm to see how it detects the key points of an image from the `opencv` library.

SIFT algorithm was first published in a paper (https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf) by David Lowe in 1999 and patented by the University of British Columbia (UBC) in Canada.

Since the UBC patents SIFT, `opencv` library doesn't provide this function by `cv2.SIFT()` . Hence, we should have to install `opencv-contrib` library explicitly as following,

In [ ]:

```
# Install opencv-contrib library
!pip install opencv-contrib-python==3.4.2.17
```

```
Collecting opencv-contrib-python==3.4.2.17
  Downloading https://files.pythonhosted.org/packages/61/29/fc60b2de17
13aa92946992544329f20ccb5e4ba26290f403e04b7da44105/opencv_contrib_pyth
on-3.4.2.17-cp36-cp36m-manylinux1_x86_64.whl (https://files.pythonhost
ed.org/packages/61/29/fc60b2de1713aa92946992544329f20ccb5e4ba26290f403
e04b7da44105/opencv_contrib_python-3.4.2.17-cp36-cp36m-manylinux1_x86_
64.whl) (30.6MB)
     |████████████████████████████████| 30.6MB 1.2MB/s
Requirement already satisfied: numpy>=1.11.3 in /usr/local/lib/python
3.6/dist-packages (from opencv-contrib-python==3.4.2.17) (1.18.5)
Installing collected packages: opencv-contrib-python
  Found existing installation: opencv-contrib-python 4.1.2.30
    Uninstalling opencv-contrib-python-4.1.2.30:
      Successfully uninstalled opencv-contrib-python-4.1.2.30
Successfully installed opencv-contrib-python-3.4.2.17
```

In [ ]:

```
### DOWNLOAD IMAGES FOR IMPLEMENTATION
!wget -O dog.jpg https://imgur.com/aeZVeM9.jpg
!wget -O book.jpg https://imgur.com/kztoYod.jpg
!wget -O books.jpg https://imgur.com/COaQRaM.jpg
# !wget -O box_in_scene.png https://imgur.com/KdBpgCZ.png
# !wget -O box.png https://imgur.com/mmsJrSg.png
```

```
--2020-07-03 06:46:23--  https://imgur.com/aeZVeM9.jpg (https://imgu
r.com/aeZVeM9.jpg)
Resolving imgur.com (imgur.com)... 151.101.40.193
Connecting to imgur.com (imgur.com)|151.101.40.193|:443... connecte
d.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://i.imgur.com/aeZVeM9.jpg (https://i.imgur.com/aeZVe
M9.jpg) [following]
--2020-07-03 06:46:23--  https://i.imgur.com/aeZVeM9.jpg (https://i.
imgur.com/aeZVeM9.jpg)
Resolving i.imgur.com (i.imgur.com)... 151.101.52.193
Connecting to i.imgur.com (i.imgur.com)|151.101.52.193|:443... conne
cted.
HTTP request sent, awaiting response... 200 OK
Length: 136844 (134K) [image/jpeg]
Saving to: ‘dog.jpg’

dog.jpg             100%[===================>] 133.64K  --.-KB/s
 in 0.02s

2020-07-03 06:46:23 (6.25 MB/s) - ‘dog.jpg’ saved [136844/136844]

--2020-07-03 06:46:26--  https://imgur.com/kztoYod.jpg (https://imgu
r.com/kztoYod.jpg)
Resolving imgur.com (imgur.com)... 151.101.40.193
Connecting to imgur.com (imgur.com)|151.101.40.193|:443... connecte
d.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://i.imgur.com/kztoYod.jpg (https://i.imgur.com/kztoY
od.jpg) [following]
--2020-07-03 06:46:26--  https://i.imgur.com/kztoYod.jpg (https://i.
imgur.com/kztoYod.jpg)
Resolving i.imgur.com (i.imgur.com)... 151.101.52.193
Connecting to i.imgur.com (i.imgur.com)|151.101.52.193|:443... conne
cted.
HTTP request sent, awaiting response... 200 OK
Length: 4540954 (4.3M) [image/jpeg]
Saving to: ‘book.jpg’

book.jpg            100%[===================>]   4.33M  --.-KB/s
 in 0.06s

2020-07-03 06:46:26 (74.4 MB/s) - ‘book.jpg’ saved [4540954/4540954]

--2020-07-03 06:46:28--  https://imgur.com/COaQRaM.jpg (https://imgu
r.com/COaQRaM.jpg)
Resolving imgur.com (imgur.com)... 151.101.40.193
Connecting to imgur.com (imgur.com)|151.101.40.193|:443... connecte
d.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://i.imgur.com/COaQRaM.jpg (https://i.imgur.com/COaQR
```

```
aM.jpg) [following]
--2020-07-03 06:46:28--  https://i.imgur.com/COaQRaM.jpg (https://i.
imgur.com/COaQRaM.jpg)
Resolving i.imgur.com (i.imgur.com)... 151.101.52.193
Connecting to i.imgur.com (i.imgur.com)|151.101.52.193|:443... conne
cted.
HTTP request sent, awaiting response... 200 OK
Length: 627718 (613K) [image/jpeg]
Saving to: ‘books.jpg’

books.jpg           100%[===================>] 613.01K  --.-KB/s
 in 0.06s

2020-07-03 06:46:28 (10.8 MB/s) - ‘books.jpg’ saved [627718/627718]
```

In [ ]:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [20, 10]

def get_keypoints(input_image):
    # Initiate SIFT detector
    sift = cv2.xfeatures2d.SIFT_create()
    keypoints, descriptor = sift.detectAndCompute(input_image, None)
    return keypoints, descriptor
```

In [ ]:

```
input_image = cv2.imread('dog.jpg', 0)
keypoints, _ = get_keypoints(input_image)
output_image = cv2.drawKeypoints(input_image,
                         keypoints,
                         input_image)

plt.axis('off')
plt.imshow(output_image)
```

Out[5]:

```
<matplotlib.image.AxesImage at 0x7f8887044be0>
```

Cool, right? `opencv` library also provides another function `cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS` which draws the circle around the key points. For this, flag should be passed to this function and it will draw the circle with the size of keypoint. Along with that, it will gives the orientation of that keypoint as well.

In [ ]:

```
keypoints_flag = cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS
out_image = cv2.drawKeypoints(input_image,
                    keypoints,
                    input_image,
                    flags=keypoints_flag)
plt.axis('off')
plt.imshow(out_image)
```

Out[6]:

```
<matplotlib.image.AxesImage at 0x7f8887044f60>
```

Brute force matching Implementation

After extracing feature descriptors, the final step is to match the correspondeces between the images features using imahe matching algorithms. We have already discussed Brute Force Matching algorithm in previous chapters. So, let's implement it now.

We will use SIFT Descriptor as feature descriptors in this implementation.

In [ ]:

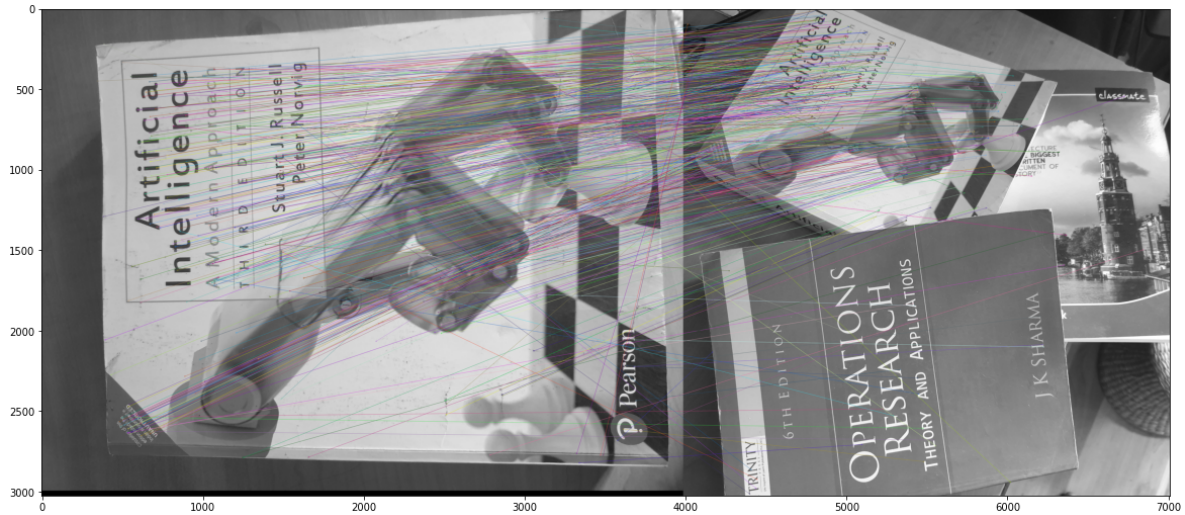
```
def get_matches(des1, des2):
    # Brute Force Matcher with default params
    bf = cv2.BFMatcher()
    matches = bf.knnMatch(des1, des2, k=2)

    # Apply ratio test
    good_matches = []
    for m, n in matches:
        if m.distance < 0.75 * n.distance:
            good_matches.append([m])
    return good_matches

def match_images(query_img_path, train_img_path):

    query_img = cv2.imread(query_img_path, 0) # queryImage
    train_img = cv2.imread(train_img_path, 0) # trainImage

    # find the keypoints and descriptors with SIFT
    kp1, des1 = get_keypoints(query_img)
    kp2, des2 = get_keypoints(train_img)

    good_matches = get_matches(des1, des2)

    # cv2.drawMatchesKnn expects list of lists as matches.
    target_img = cv2.drawMatchesKnn(query_img, kp1, train_img, kp2, good_matches, N
    return target_img
```

In [ ]:

```
matched_image = match_images('book.jpg', 'books.jpg')
plt.imshow(matched_image)
```

Out[51]:

```
<matplotlib.image.AxesImage at 0x7f942909c7b8>
```



References

- Books
  - Richard Szeliski (2010), Computer Vision: Algorithms and Applications (http://szeliski.org/Book/drafts/SzeliskiBook_20100903_draft.pdf), 2010 Draft, Springer
    - Check pages 191 - 196 to understand the foundations and mathematics of feature detection and matching in detail.

- University lectures
  - L. W. Kheng, CS4243 (Computer Vision and Pattern Recognition) (https://www.comp.nus.edu.sg/~cs4243/lecture/feature.pdf) - National University of Singapore
    - Check pages 18 - 32 for mathmatical understanding of SIFT.
  - Dr. Mubarak S., CAP 5414 Computer Vision (https://www.youtube.com/watch?v=L77m5xuDSKw) - University of Central Florida
- Tutorial
  - OpenCV Brute Force Matching Tutorial (https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html)