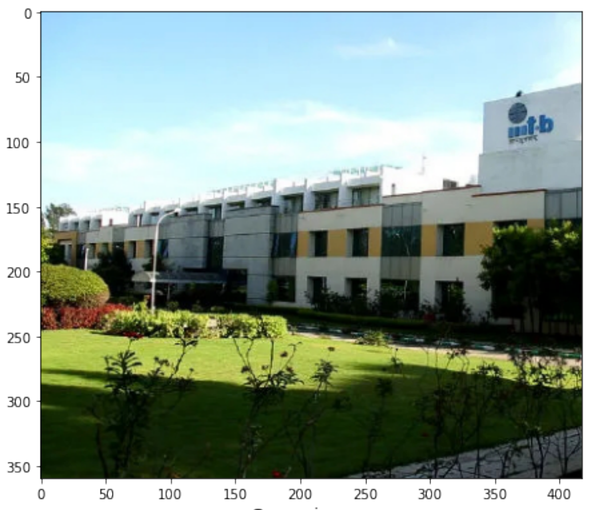
AI 825 / Visual Recognition 03-03-2022

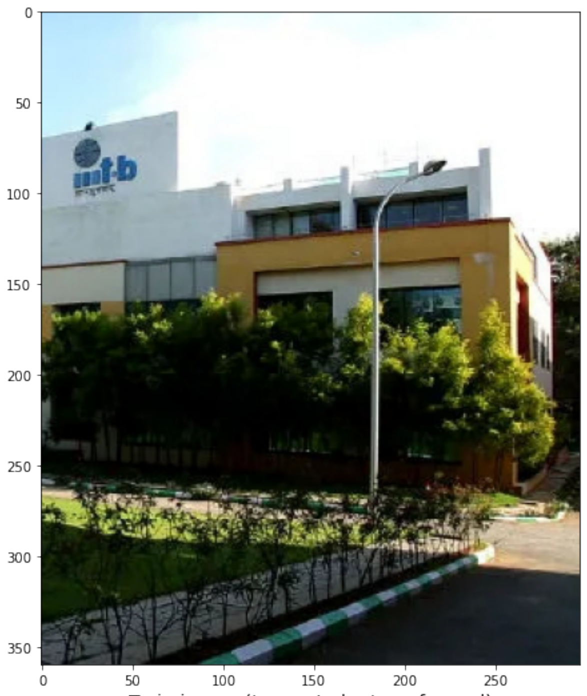
ASSIGNMENT 2
Question 1

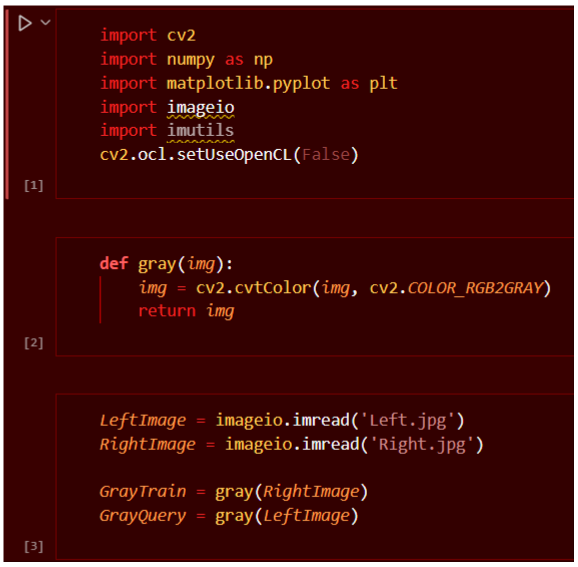
Instructor: Prof. Dinesh Babu Shaikh Fahed IMT2019079

1 Image Selection

I have taken two photographs of the Aryabhatta block in IIITB and made sure they both contain the overlapping part. The two images, Query and Train are as shown below. The code of required libraries and reading images, converting them to grayscale is as shown below.


Query image


Train image (Image to be transformed)



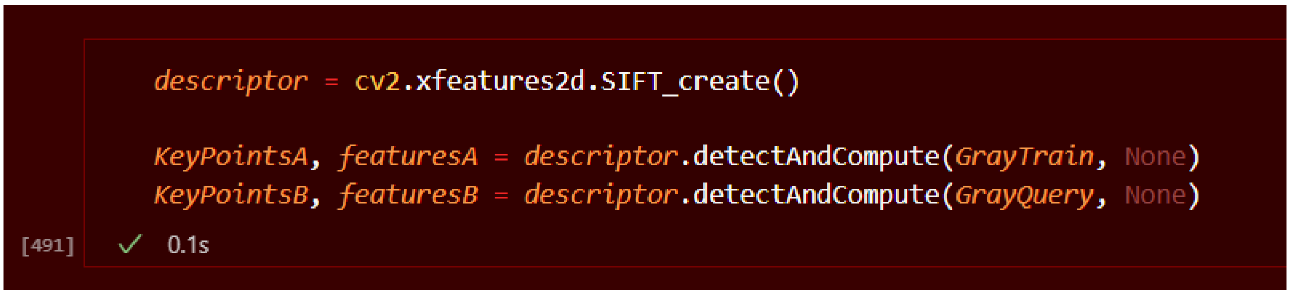
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import imageio
import imutils
cv2.ocl.setUseOpenCL(False)
```
[1]

```
def gray(img):
    img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    return img
```
[2]

```
LeftImage = imageio.imread('Left.jpg')
RightImage = imageio.imread('Right.jpg')

GrayTrain = gray(RightImage)
GrayQuery = gray(LeftImage)
```
[3]

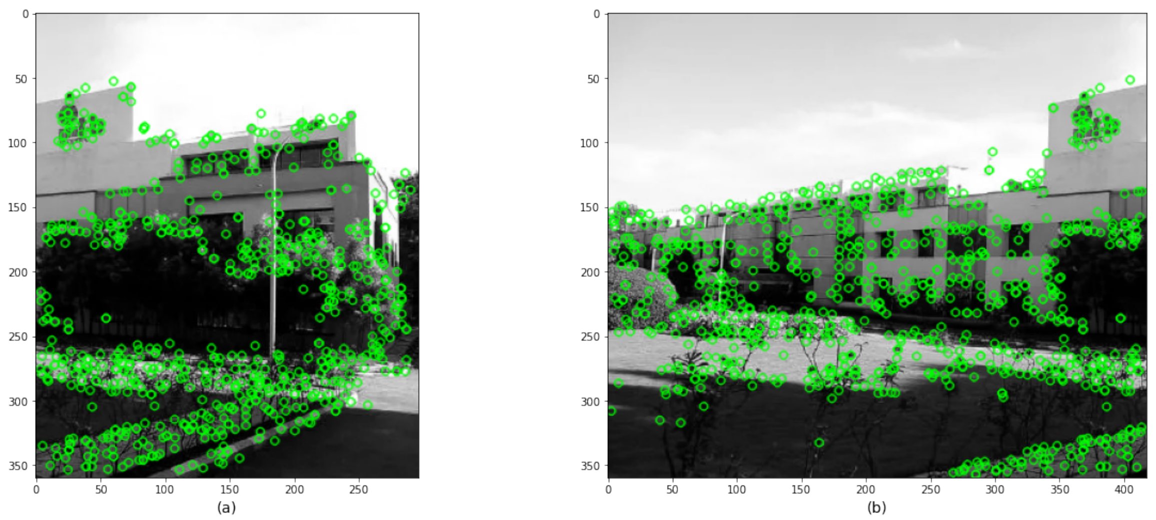
2 Keypoints Detection

Using the following code, I have detected the keypoints in both the images and depicted them over the grayscaled images of input files.



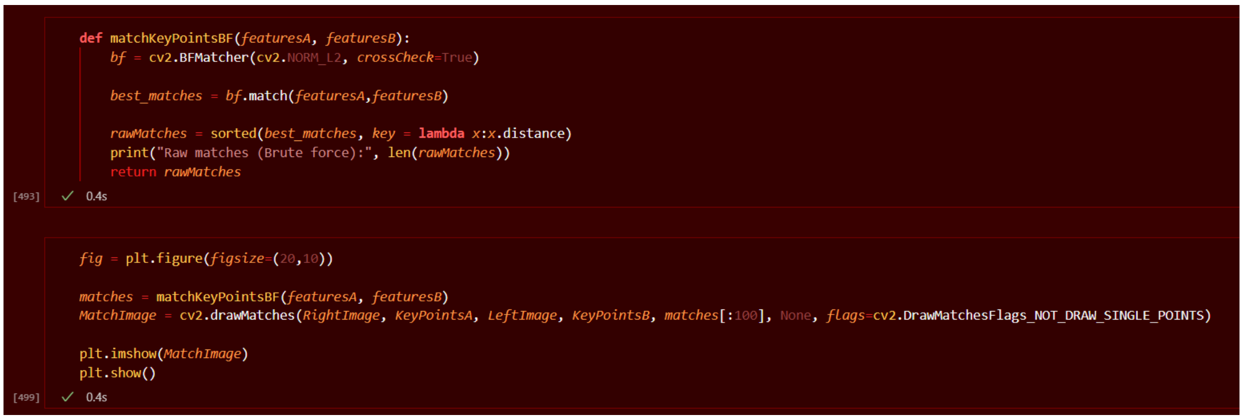
```
descriptor = cv2.xfeatures2d.SIFT_create()

KeyPointsA, featuresA = descriptor.detectAndCompute(GrayTrain, None)
KeyPointsB, featuresB = descriptor.detectAndCompute(GrayQuery, None)
```
```
[491]  ✓ 0.1s
```

Then, using the ***cv2.drawKeypoints()*** function, we can portray the keypoints in the image as shown below.


(a) (b)

3 Matchmaking

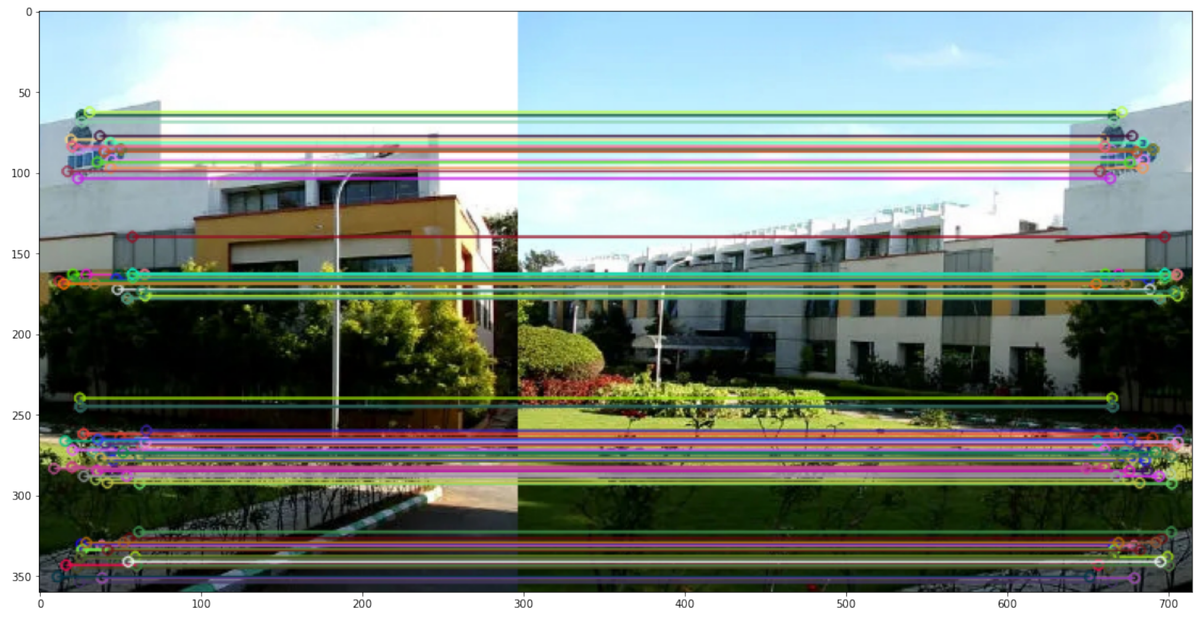
Matchmaking is the process of joining the overlapping keypoints which can be achieved by the assistance of following part of the code.



```
def matchKeyPointsBF(featuresA, featuresB):
    bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)

    best_matches = bf.match(featuresA,featuresB)

    rawMatches = sorted(best_matches, key = lambda x:x.distance)
    print("Raw matches (Brute force):", len(rawMatches))
    return rawMatches
```
```
[493]  ✓ 0.4s
```

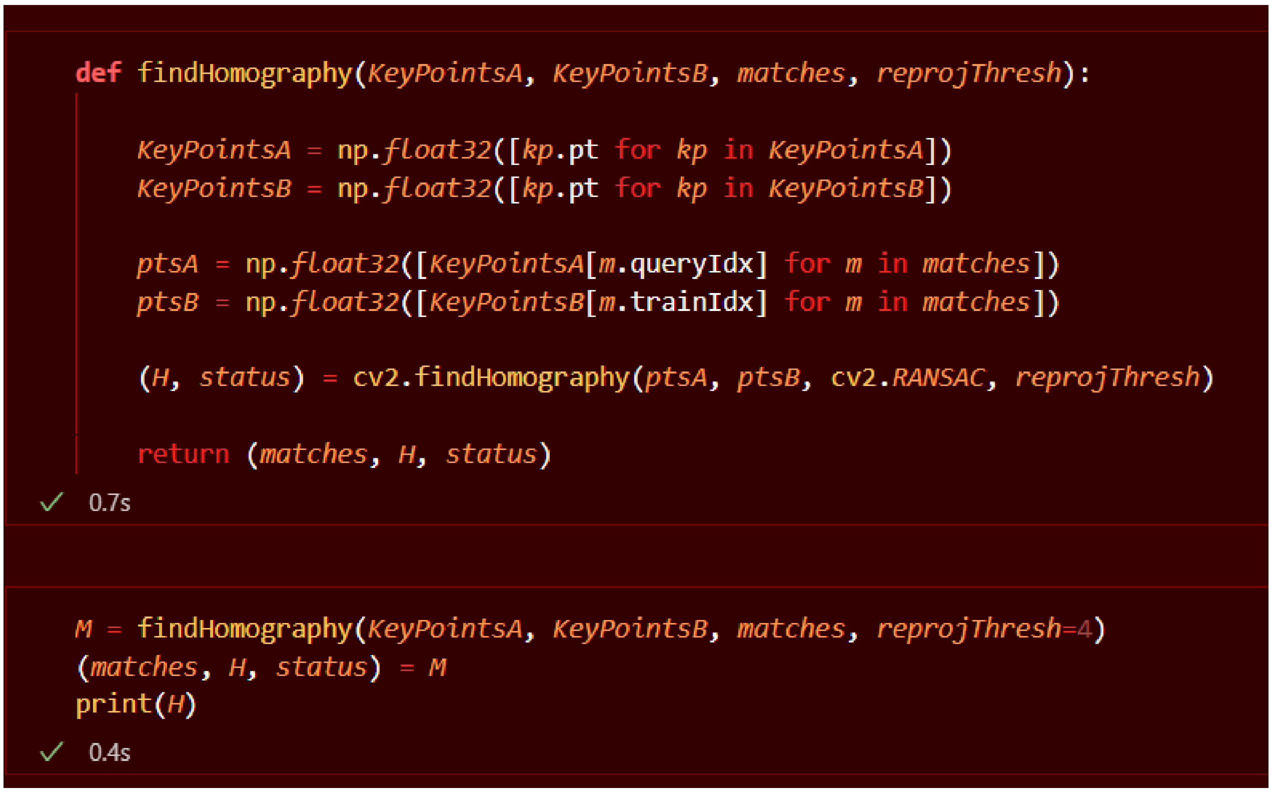
```
fig = plt.figure(figsize=(20,10))

matches = matchKeyPointsBF(featuresA, featuresB)
MatchImage = cv2.drawMatches(RightImage, KeyPointsA, LeftImage, KeyPointsB, matches[:100], None, flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)

plt.imshow(MatchImage)
plt.show()
```
```
[499]  ✓ 0.4s
```

The function **textitmatchKeyPointsBF(featuresA, featuresB)** takes the features of both the images and builds the matches. After that, with the help of a simple function ***cv2.drawMatches()***, we can show all the matches found on the collage of both the input images. The output would look something like this



4 Get Homography

Now that we have made the matches, we need to get the Homography from the keypoints and matches. For that, we have a code as shown:



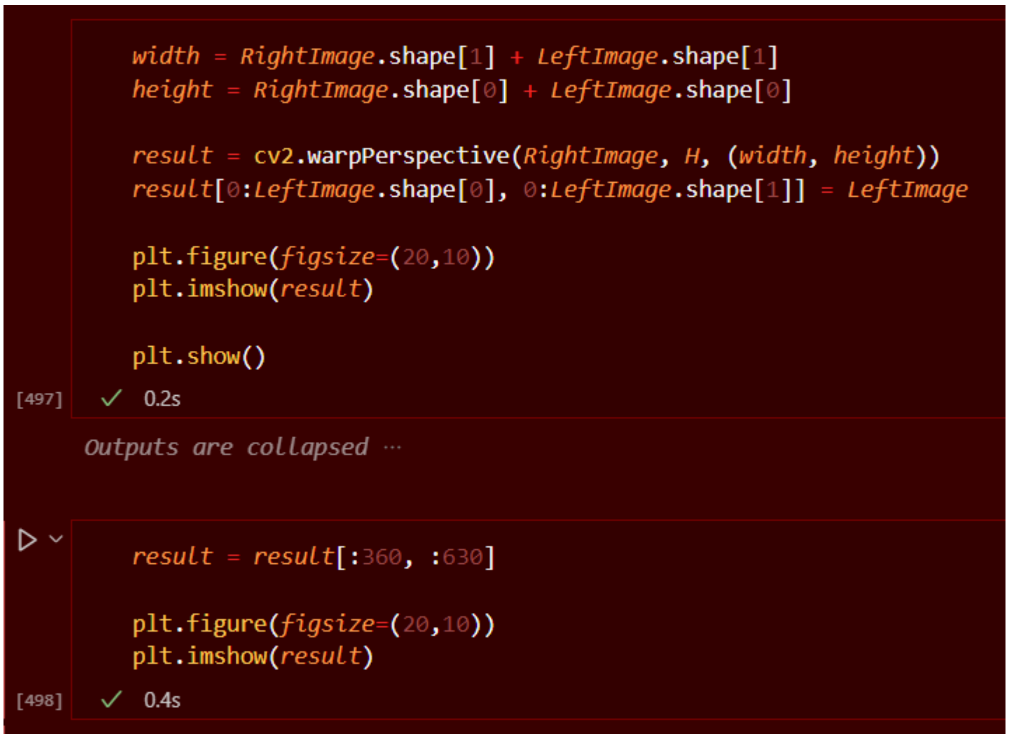
```
def findHomography(KeyPointsA, KeyPointsB, matches, reprojThresh):

    KeyPointsA = np.float32([kp.pt for kp in KeyPointsA])
    KeyPointsB = np.float32([kp.pt for kp in KeyPointsB])

    ptsA = np.float32([KeyPointsA[m.queryIdx] for m in matches])
    ptsB = np.float32([KeyPointsB[m.trainIdx] for m in matches])

    (H, status) = cv2.findHomography(ptsA, ptsB, cv2.RANSAC, reprojThresh)

    return (matches, H, status)
✓ 0.7s
```

```
M = findHomography(KeyPointsA, KeyPointsB, matches, reprojThresh=4)
(matches, H, status) = M
print(H)
✓ 0.4s
```

Using this, we get Homography which is later used in the ***cv2.warpPerspective(RightImage, H, (width, height))*** function to get the resultant image.

5 Resulting Panaroma

Now, to merge the images in a common perspective, we can simply define the height and width of the merged image and print the final panaromic view as shown below.



```
width = RightImage.shape[1] + LeftImage.shape[1]
height = RightImage.shape[0] + LeftImage.shape[0]

result = cv2.warpPerspective(RightImage, H, (width, height))
result[0:LeftImage.shape[0], 0:LeftImage.shape[1]] = LeftImage

plt.figure(figsize=(20,10))
plt.imshow(result)

plt.show()
```

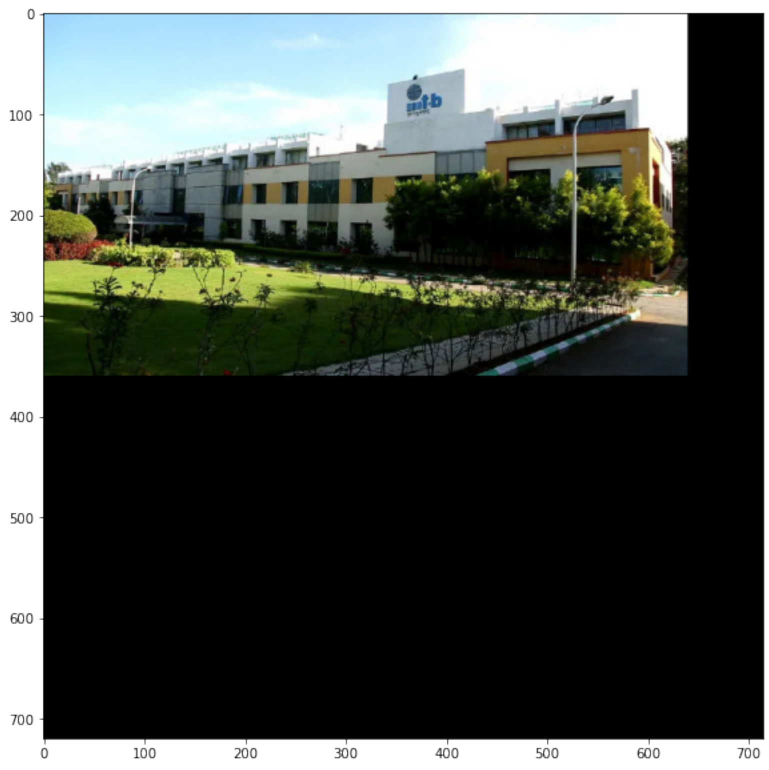
[497] ✓ 0.2s

Outputs are collapsed ···

```
result = result[:360, :630]

plt.figure(figsize=(20,10))
plt.imshow(result)
```

[498] ✓ 0.4s

The output would look as shown below, but I have cropped and resized it for a better look in the next page.



.
.

iiit-b
0
50
100
150
200
250
300
350
0
100
200
300
400
500
600

Question1:
Explain how SURF is different from SIFT.

Answer1:
SIFT and SURF are good in illumination changes images.
SIFT and SURF, both are robust method to find feature detection and matching.
Both are invariant to rotation, scale, blur, Illumination, warping and noise area.
In illumination, both have same effect to find detect feature points.

SURF is not as good as SIFT in different scale images.
SURF is 3 times faster than SIFT because using of integral image and box filter.
SIFT uses the original filter and SURF has approximated filter.
SIFT uses Greyscale images as algorithm inputs whereas SURF takes intensity images.
Base of the descriptor for SIFT is Gradients, and SURF uses Haar-wavelet filter responsse.
Dimension of the descriptor is 128-d and 64-d in in SIFT and SURF respectivvely.
SIFT is more reliable although SURF is faster.
SIFT has good result than SURF in scale area.
SURF has good result in rotation, blur, warping, RGB noise and time consuming than SIFT.

Question2:
Main Principles of FLANN Matching and RANSAC

Answer2:
RANSAC
The abbrevation for Random Sample Conseus is an iterative appraoch to estimate a mathematical model from a dataset that consists of outliers. It is a sturdy method to estimate the fundamental matrix in stereo vision, to find the similarities among units of factors for characteristic-based totally object detection, and listing down sequential video frames for video stabilization.

FLANN Matching
This method ventures the excessive-dimensional features to a minimal-dimensional space after which generates the compact binary codes. The substantial benefit for these compact codes is that from the produced codes, rapid image searches can be carried out through the binary pattern matching which dramatically declines the computational cost and thereby improves the performance of the hunt.