Assignment-2

Q2a. Image Stitching:

Ans: **At first, I took three photos from my hostel balcony at same time and same camera configuration. I made sure those three images will be pairwise overlapping.**

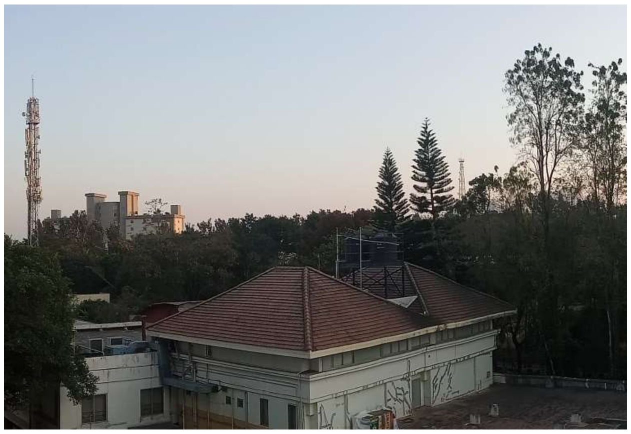


Image 1

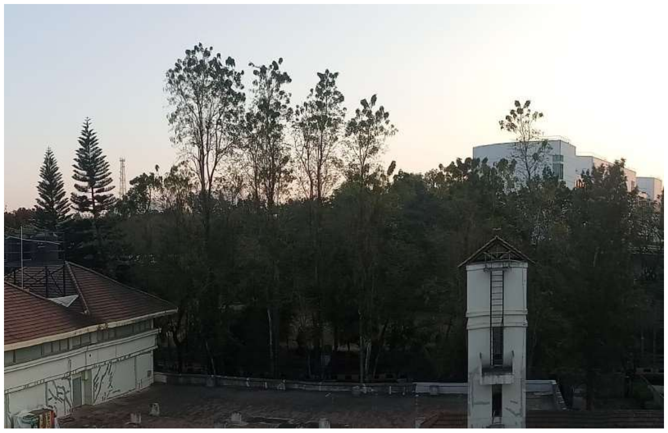


Image 2

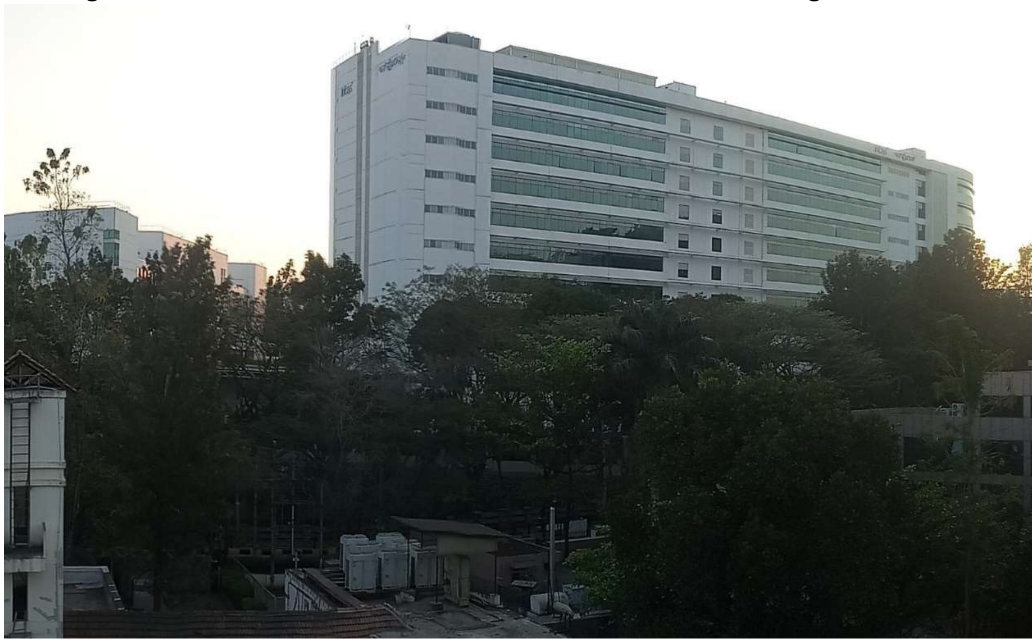


Image 3

Goal: **Combine those three images into one single image and form a panorama.**

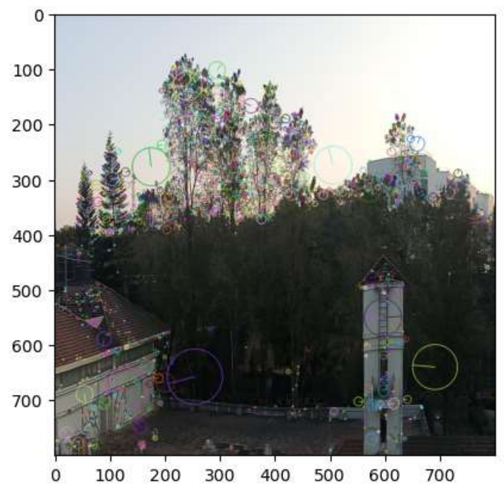
Approach: **To achieve the above goal following steps are implemented:**

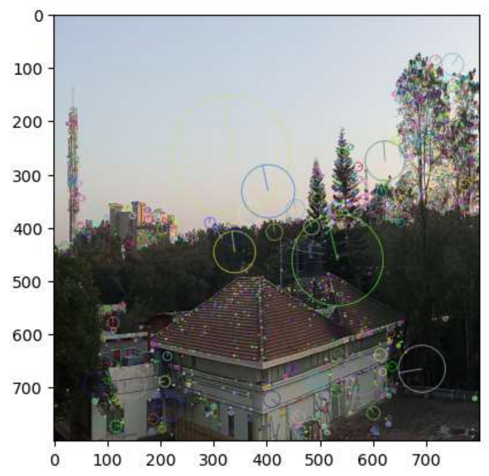
Here I took first two images and stitched them together and got a resultant image. Then took that resultant image as first image and image 3 as 2nd image and stitched them together and got the final image.

- **Pre-Processing**
- **Keypoint detection**
- **Local invariant descriptors (SIFT, SURF)**

- **Feature matching (Brute force matcher and Flann based matcher)**
- **Homography estimation using RANSAC**
- **Perspective warping**
- **Post Processing**
- **<u>Pre-Processing:</u>** At first, I loaded Image 2 as img1 and Image 1 as img2 and resized both of them as 800*800

- **<u>Keypoint Detection & Computing Local Invariant Descriptors:</u>** Corner detectors like Harris etc are rotation-invariant but not scale invariant. So, here we used Scale Invariant Feature Transform (SIFT), which extract keypoints and compute its descriptors. It is both scale and rotation invariant. Here we used OpenCV implementation of SIFT algorithm I.e. sift.detectAndCompute(). Using that we got 2415 keypoints and their respective descriptors (in 2415*128 NumPy array) in img1 and 2332 keypoints and their respective descriptors (in 2332*128 NumPy array) in img2.

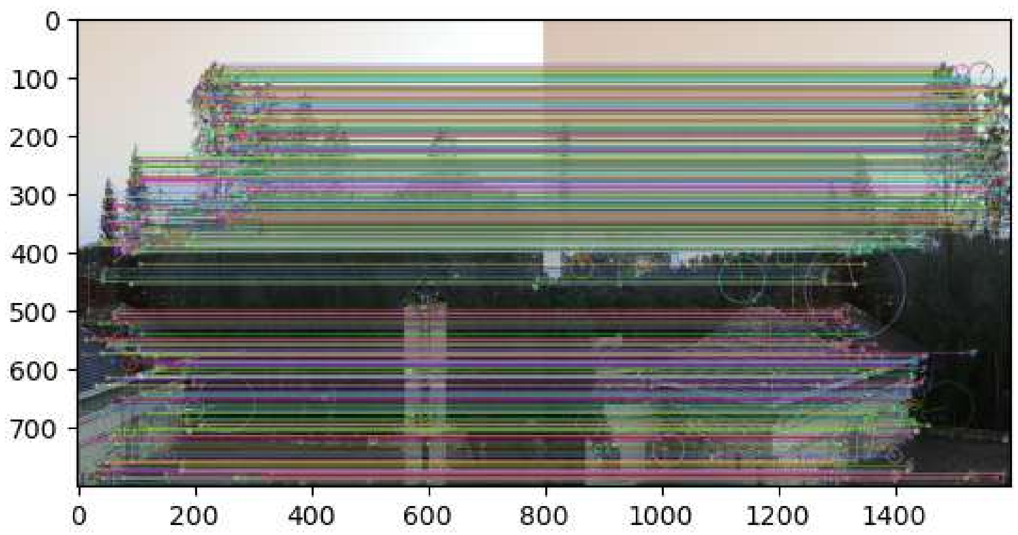




Keypoints and their Orientations in img2 & img1 respectively

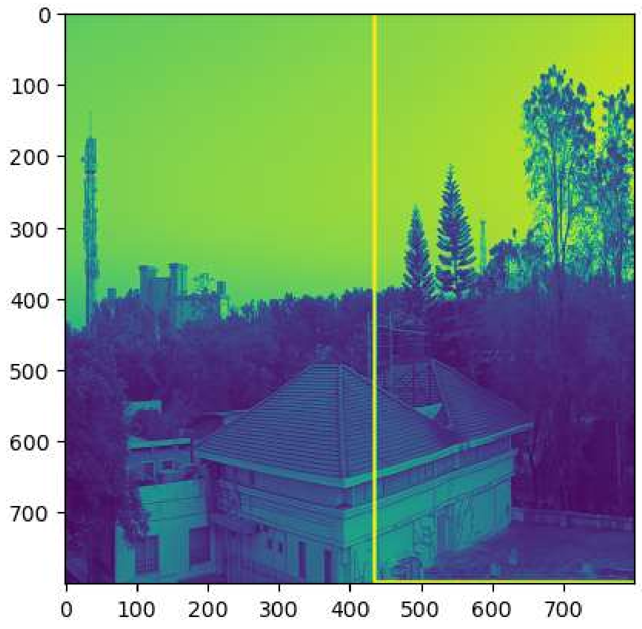
- **<u>Feature matching</u>:**
- **<u>Brute Force Matcher</u>: Here we used OpenCV's brute force matcher to match the descriptors in both the images and got a tuple named 'matches', whose each element is a tuple of two points from two images.**
- **<u>Flann Based Matcher</u>: We also performed matching using OpenCV's cv2.FlannBasedMatcher , taking FLANN_INDEX_KDTREE =1**

After matching we performed a ratio test taking the thresold as 0.3 and found a list named 'good' of length 648. Then we drew lines between corresponding points in the two images using cv2.drawMatchesKnn() and got the below image.

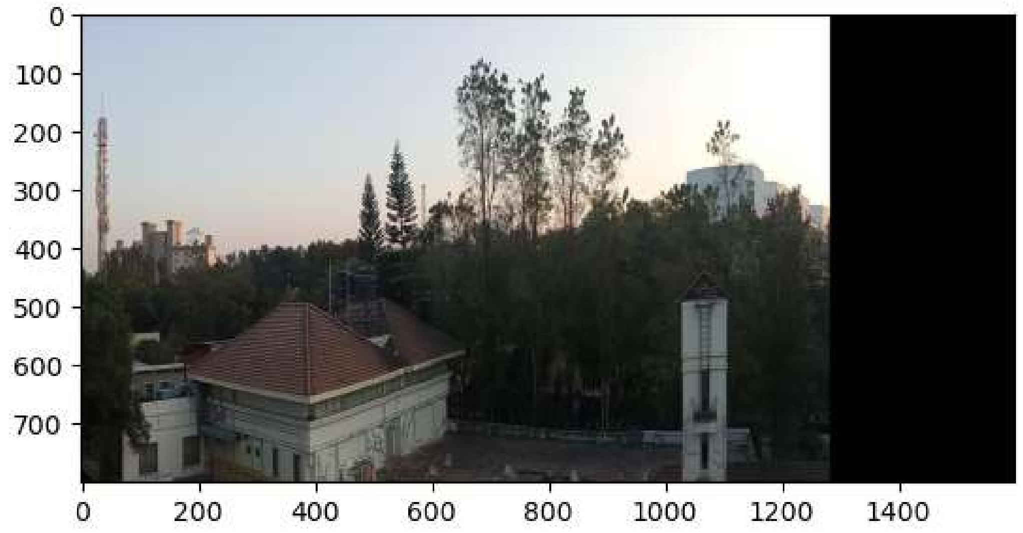


- **Homography Estimation Using RASNAC:**

Now, we need to take these points and find the transformation matrix that will stitch the 2 images together based on their matching points. Such a transformation is called the Homography matrix. Briefly, the homography is a 3x3 matrix that can be used in many applications such as camera pose estimation, perspective correction, and image stitching. The Homography is a 2D transformation. It maps points from one plane (image) to another. Here, we are going to use RANSAC to estimate the Homography matrix. It turns out that the Homography is very sensitive to the quality of data we pass to it. Hence, it is important to have an algorithm (RANSAC) that can filter outliers points from inliers. We computed Homography matrix using cv2.findHomography(). After that we also showed the overlapping regions between two images using cv2.perspectiveTransform() and cv2.polylines() functions.

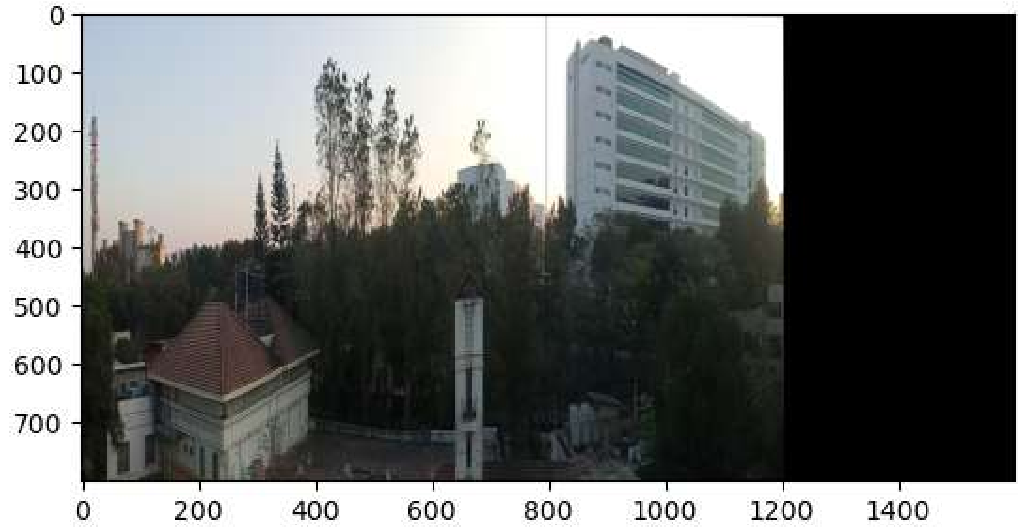


- <u>Perspective warping</u>: Now we need to warp one of the images to a common plane by applying a perspective transformation. Basically, a perspective transform may combine one or more operations like rotation, scale, translation, or shear. We did this by using OpenCV's cv2.warpPerspective() function, which takes Homography matrix M and right image as input and warps the two images based on homography.



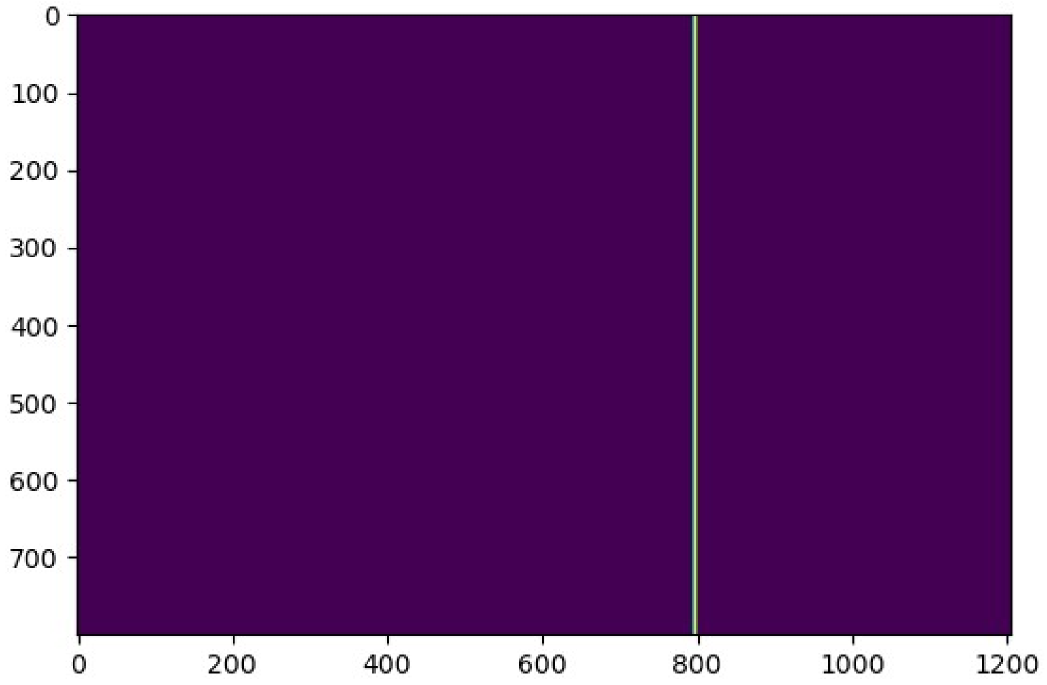
Resultant Image

Then we took the resultant image as left image and image 3 as right image and performed the above steps again and got the the final result or our panorama image.

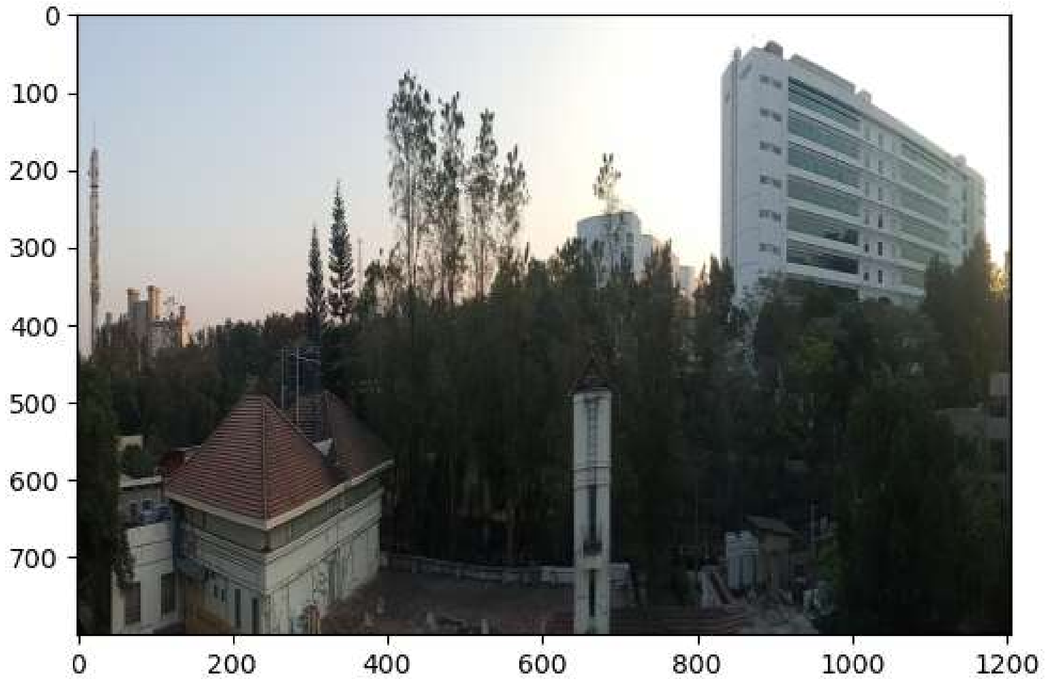


<u>**Panorama**</u>

- <u>Post Processing</u>:
- After getting the initial resultant image or the final panorama image we trimmed the image's black borders by defining a trim function.
- And near the width 800 you can spot a very thin black line for removing that we defined a mask and used inpainting method i.e. OpenCV's cv2.inpaint(img1,mask2,3,cv2.INPAINT_TELEA) method.



Mask



<u>Final Panorama Image</u>

- **<u>Before trying image stitching on images taken by me, I tried it on images taken from internet. I will attach that notebook as well. That notebook is self-explanatory.</u>**