CMSC 426
Project 2 - Panorama Stitching

-Rachith Prakash

Overview:

The project involves understanding the key concepts in creating a panoramic view. Panoramic stitching involves stitching of images that are with little(depends) variation with respect to each other in terms of illumination, orientation, translation, etc. With these variations in images, how to produce a single image with all these images stitched to each other so that it represents the view of the combined images.

Implementation:

1. **Feature detection/Corner point detection:**

    a. Corners or key points or interest points are crucial for this process. These points are those whose intensity varies almost along all directions as we move away from it. Hence, these points sort of define an image.
    b. Used the inbuilt *detectHarrisFeatures,* another version of *cornermetric* function. Difference being, we can specify a threshold for corner detection in *detectHarrisFeatures* i.e. *detectHarrisFeatures (Image, 'MinQuality', <threshold>)* identifies a point as a corner only if its strength is greater than the threshold provided, whereas, the *cornermetric* uses a very low threshold and hence gives a point in the sky as a corner which is not a good feature point in terms of uniqueness. Hence, by using *detectHarrisFeatures,* we filter out weak corners/feature points.
    c. Threshold for my project - 0.000001

2. **Adaptive Non-Maximal Suppression (ANMS):**

    a. Corners obtained from the previous step can be very close to each other (geometrically/spatially). For e.g. in a 5x5 window of an image, we can have 20 pixels identified as corners. This might look like a good thing as we got more corners meaning we got more feature points. But, since these are in close proximity, the value each point adds is very less due to several reasons. First, our main is to identify as many distinguishable feature points across the whole image as possible so that we can get a sense of how the whole image is related to another. Second, it will be difficult to get to identify these corner points while feature mapping as these would lie very close to each other and there would be a probability of wrong mapping leading to bad Homography matrices.
    b. ANMS algorithm reduces the density of corner points across the whole image i.e. it locally minimizes the number of feature/corner points. The below image illustrates this. The left half of the image shows the corner points as detected by *detectHarrisFeatures.* The right half of the image shows the ANMS output image. Clearly, the number of corners has reduced, but, more importantly, the corners are now widely spaced with respect to each other. This would enable feature extraction and feature mapping to work better.
    c. How do we do this? Initially, *imregionalmax* with 8 connected neighbors to filter out weak corner points is used. Next, a corner point is taken and if any other corner point is stronger than this (intensity wise), distance of all such corner points from it is calculated and the nearest one is kept i.e. each corner point is associated with its nearest stronger neighbor corner point. This

is done for all corner points. These distances are sorted in decreasing order and 'Nbest' number of them are selected. This means we are eliminating corners nearby a corner which has more intensity than the rest. I have used 'Nbest' of 400.

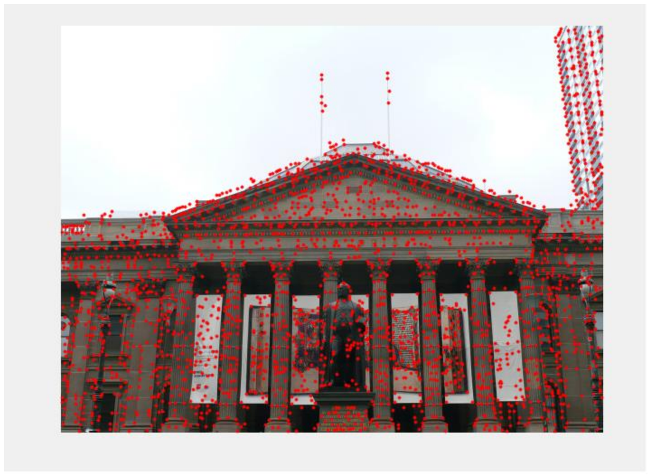
Figure 1: Before ANMS

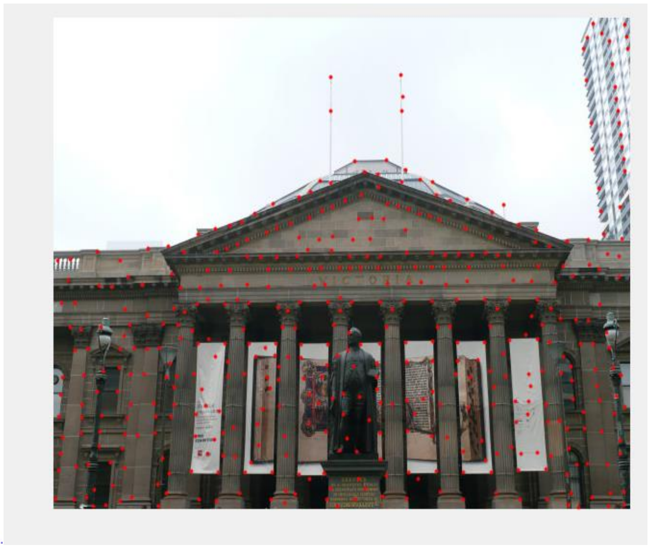
Figure 2: After ANMS

3. Feature Descriptor:

a. Once a set of corners that are reliable are obtained it can be used to describe/identify an image. We describe a feature vector for each corner. A corner would have 3 components to it. Its location coordinates (X, Y) and intensity value. Using these values, it will be difficult to explain a corner. Why? A position and intensity do uniquely represent a corner w.r.t to its image coordinate system, but when we try to identify similar corner in another image, how would these values help when the coordinate system of the other image along with the intensity value of every pixel could be changed. These changes are due to the change in camera angle and changes in illumination.
b. Thus, we need more robust representation of a corner to be successful in identifying it, even when there are changes to an image like change in illumination, image being rotated, zoom out, zoom in, etc. In computer vision, we rely on neighboring pixels for almost anything like filtering, edge detection, corner detection, derivatives, Laplacian of Gaussian, Difference of Gaussian, and in many other operations. Hence, here we rely on neighboring pixels of a corner to describe it.
c. Algorithm used here will take a 41x41 window around a corner, smoothen it using a 5x5 gaussian filter with std. deviation 1. This would weigh the corner image higher when compared to its neighbor. Next, we subsample the weighted window to get an 8x8 window. Then, this matrix is reshaped to a vector of 64 length (pixels). Sub sampling is done to reduce the size of the feature. Had we 41x41 window, we would have 1681 length array which would put lots of computational load and would take more time to produce output. This 1x64 or 64x1 vector is a

feature representation of a corner or also known as feature descriptor. Thus, there would be ‘n’ feature descriptors for an image with ‘n’ corners. This ‘n’ is user configurable.

4. **Feature matching:**
    a. Once a set of features are obtained for each corner, corners can now be matched with themselves across images with different artifacts.
    b. The closeness/similarity of one corner point with itself across images is estimated through least square method also sum of square differences (SSD) in this case. The value of each vector (feature) component is subtracted component wise (and squared) for every corner point w.r.t to every other corner point. These values are then sorted in ascending order. These points must be distinguishable w.r.t to each other i.e. if two features (ones with lowest SSD’s) appear close to each other then the uniqueness of that corner point comes into question. Hence, these points are rejected.
    c. Ratio of 0.25 is used as a criterion to remove features/corner points that are not strongly matched.
    d. Feature points which are distinguishable are considered as matched. Hence, the location of these points in their respective images is captured.
    e. *showMatchedFeatures* is used to visualize the features matched from one image to another.

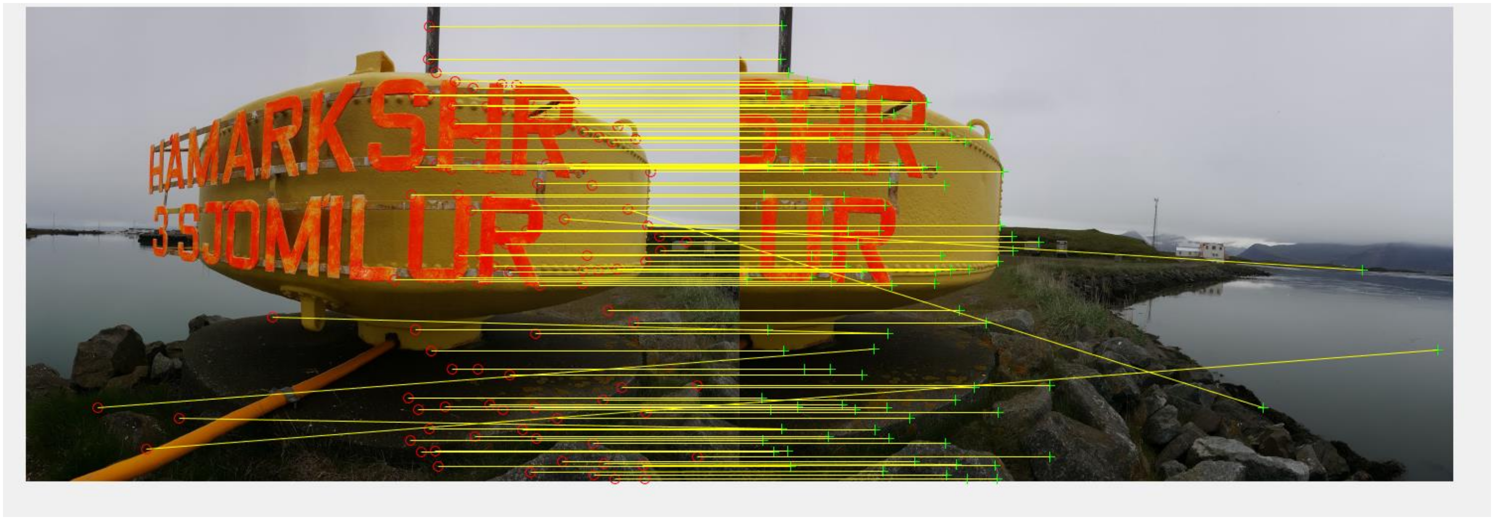

Figure 3: Feature Matching before RANSAC. In relation with Figure 5.

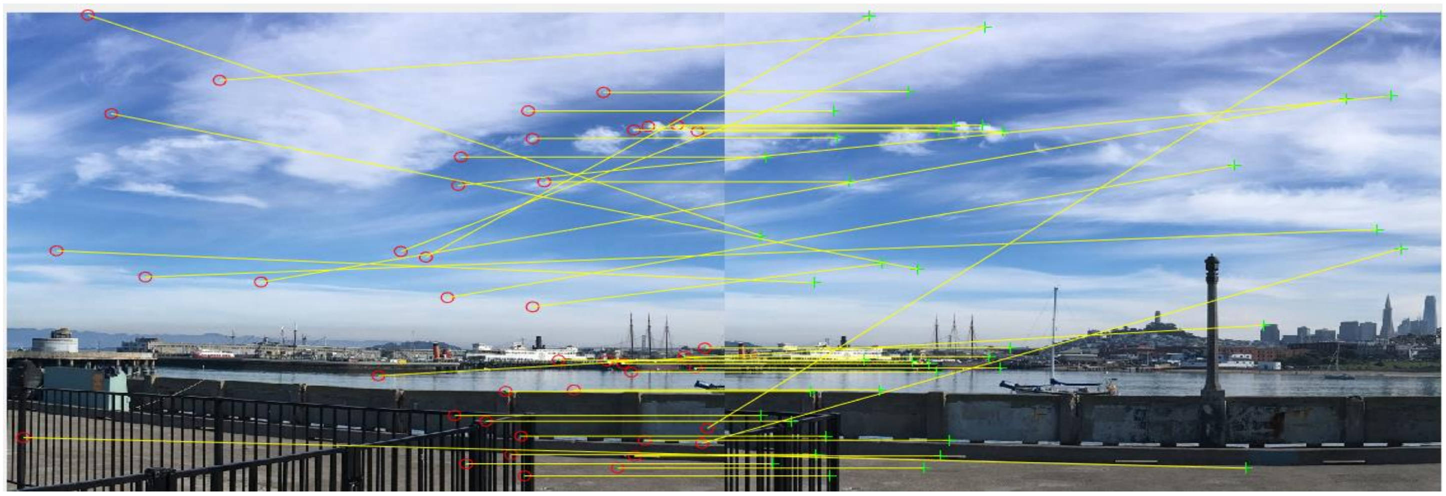

Figure 4: Feature Matching before RANSAC. In relation with Figure 6.

5. **RANSAC:**
    a. There are still possibilities of outliers or wrong matches to happen. Hence, RANSAC algorithm is used to filter out these outliers.
    b. How? By calculating the Homography matrix and finding the largest number of inliers. Homography matrix is used to projective transform one image to another. Then, the difference in the position of the actual corner and the projected corner is taken to identify an inlier. A threshold of about 5-10 is applied to capture inliers.
    c. If the inliers are very less, either the Homography matrix is wrong or the features are bad. Hence, we try to get the best Homography matrix i.e. has more inliers.
    d. Once all inliers are found, I'm using *fitgeotrans([DestPoints,] [SourcePoints],'projective')* for calculating the Homography matrix. This function is used for purposes of future calculations where *imwarp* is involved and this function outputs the matrix in desired format.

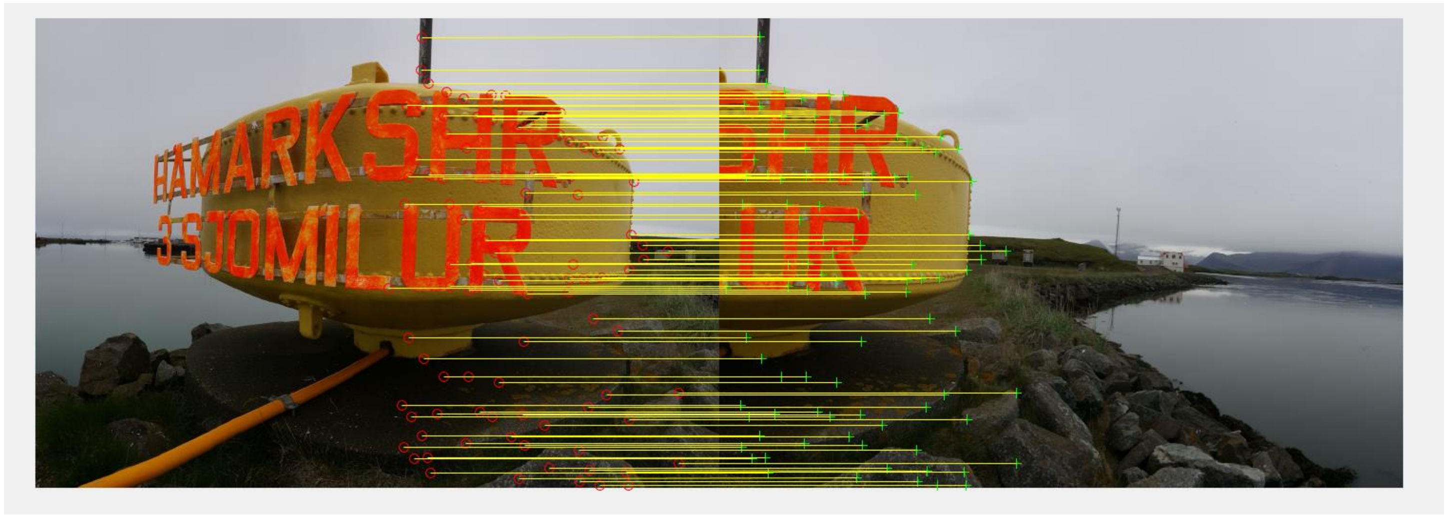

Figure 5: Feature Matching after RANSAC. Outliers are removed.

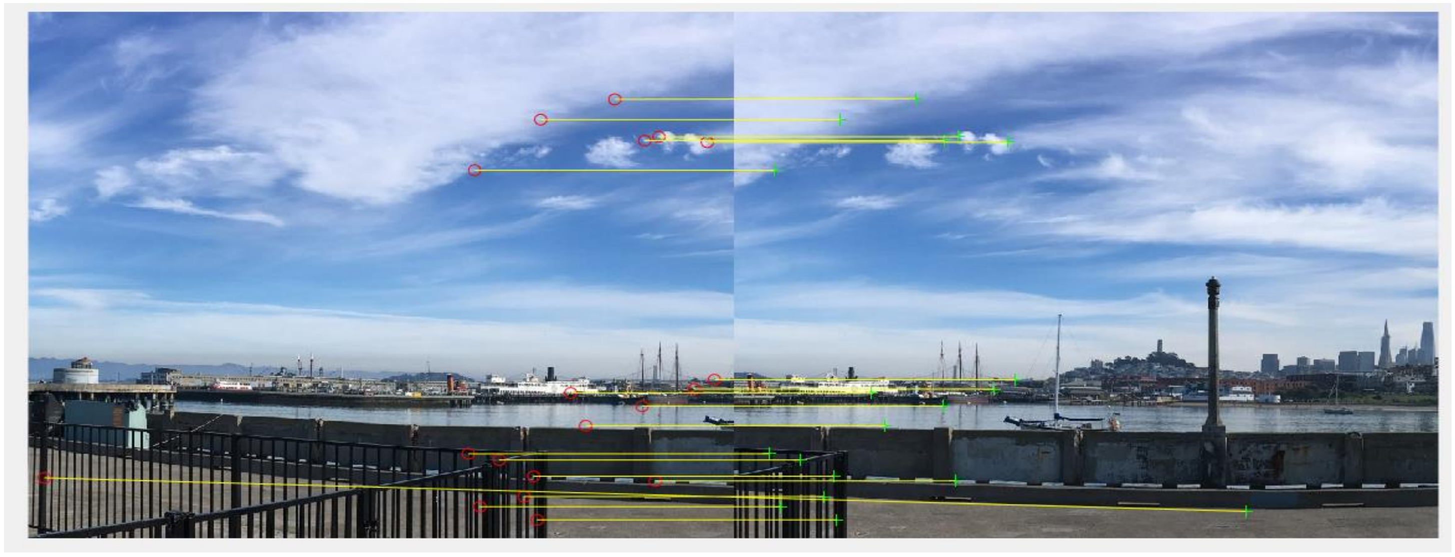

Figure 6: Feature Matching after RANSAC. Outliers are removed.

6. **Stitching and blending:**
   a. Once RANSAC is done, fine feature matches of images are obtained as shown above.
   b. Stitching:
      i. From RANSAC, Homography matrix of previous image is calculated w.r.t to the next image.
      ii. These, Homography matrices are then multiplied in the form $H(i) = H(i) * H(i-1)$, i equal to 1to n, where n represents the number of Images.
      iii. This gives all frames w.r.t frame of first image. But, we need these matrices w.r.t center image so that we can transform any image to this center image and fit in the panorama.
      iv. H inverse of 4 w.r.t 1 is calculated to give H 1 w.r.t 4.
      v. Now, we have all images' Homography matrix w.r.t center image. Thus, these images can be easily warped with center image.
      vi. Using *outputLimits* to calculate the range of maximum and minimum x axis and y axis value for the panorama. Once, we have this, we can calculate the width and height of the panorama. Also, initializing an empty image with the dimensions of the height and width so that all stitches can fit in.
      vii. Applied the Homography matrix to each image, to stitch it with the center one.
   c. Blending:
      i. Using an inbuilt function blender for this purpose.

OUTPUT IMAGES AND OBSERVATIONS:

- The distortions at the end of the images is because I didn't use cylindrical projections for the panorama stitching.

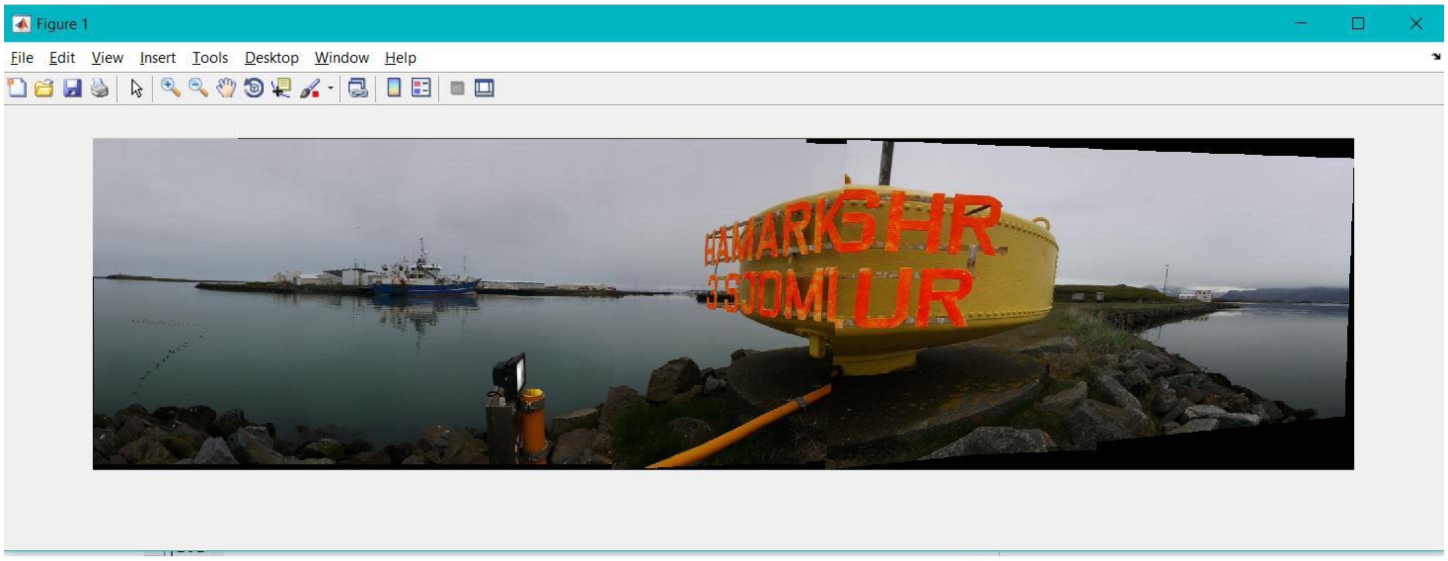

Figure 7: Test 1 Panorama Output. r(2)/r(1)=0.25. Nbest = 450

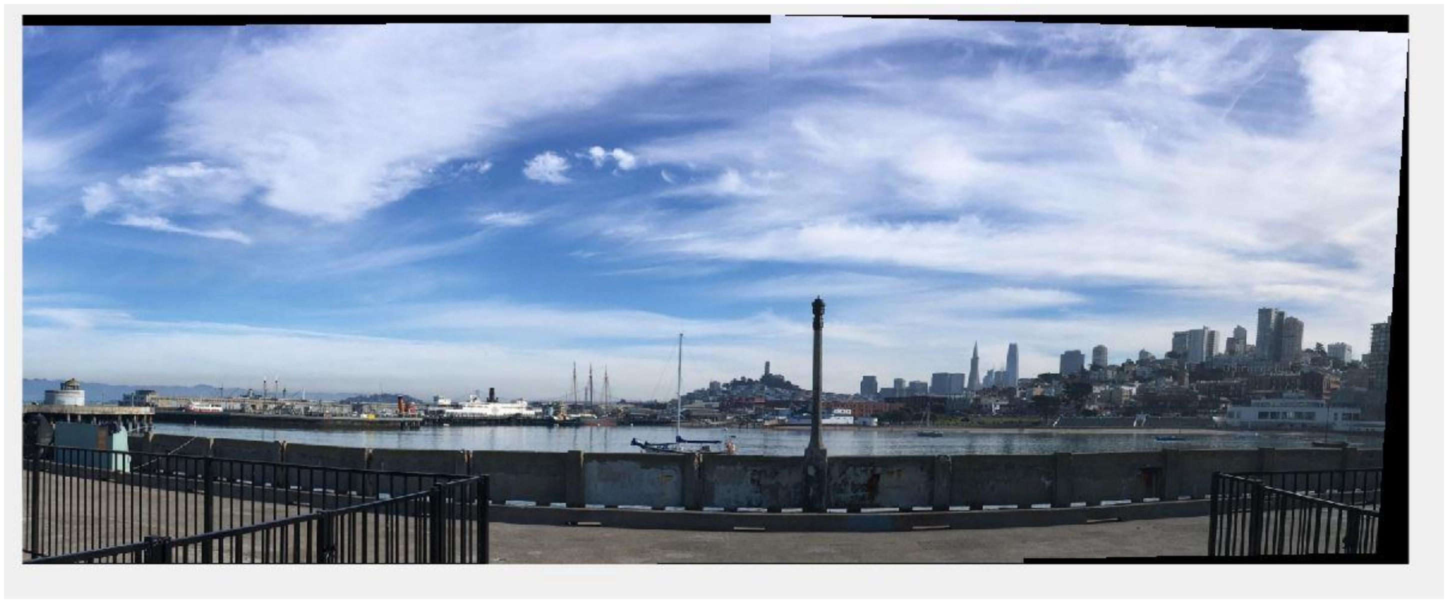

Figure 8: Test 2 Panorama Output. r(2)/r(1)=0.25. Nbest = 450

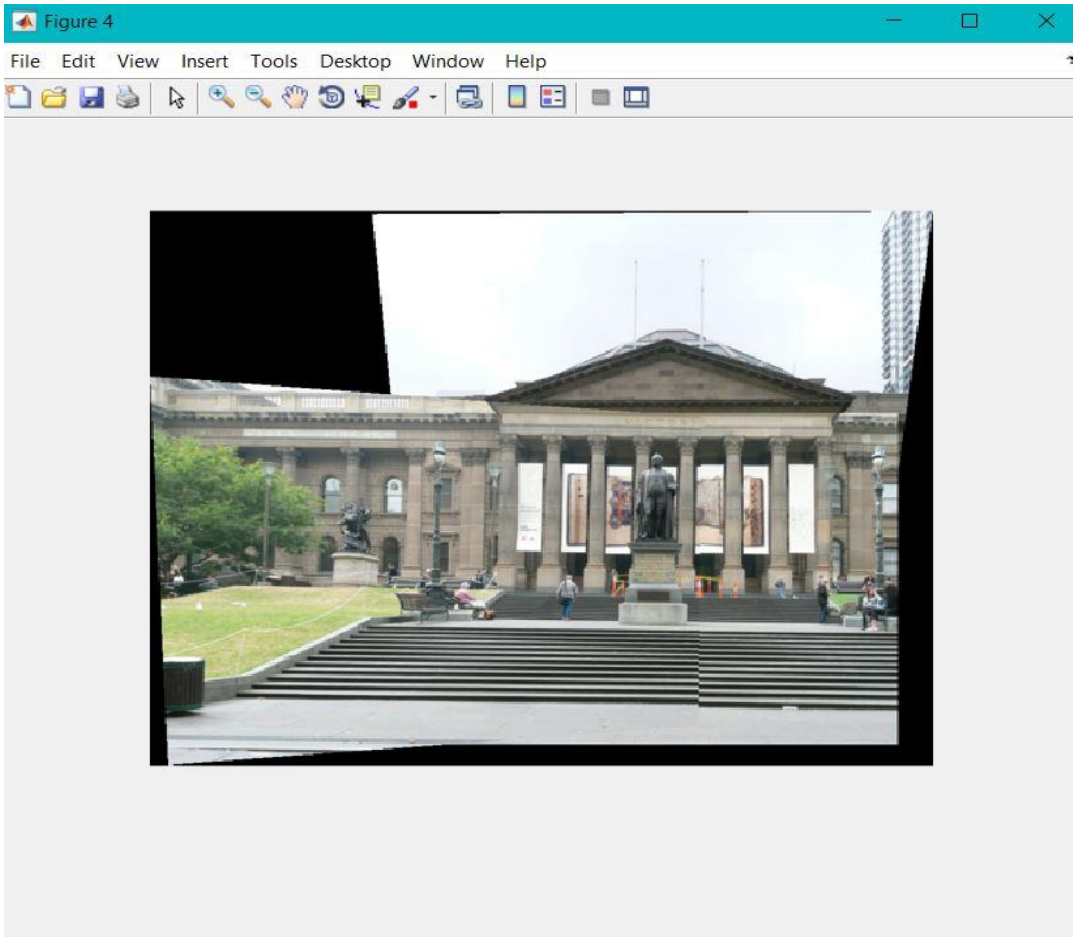

Figure 9: Training Set 1 Output Image. r(2)/r(1)=0.25. Nbest = 450.

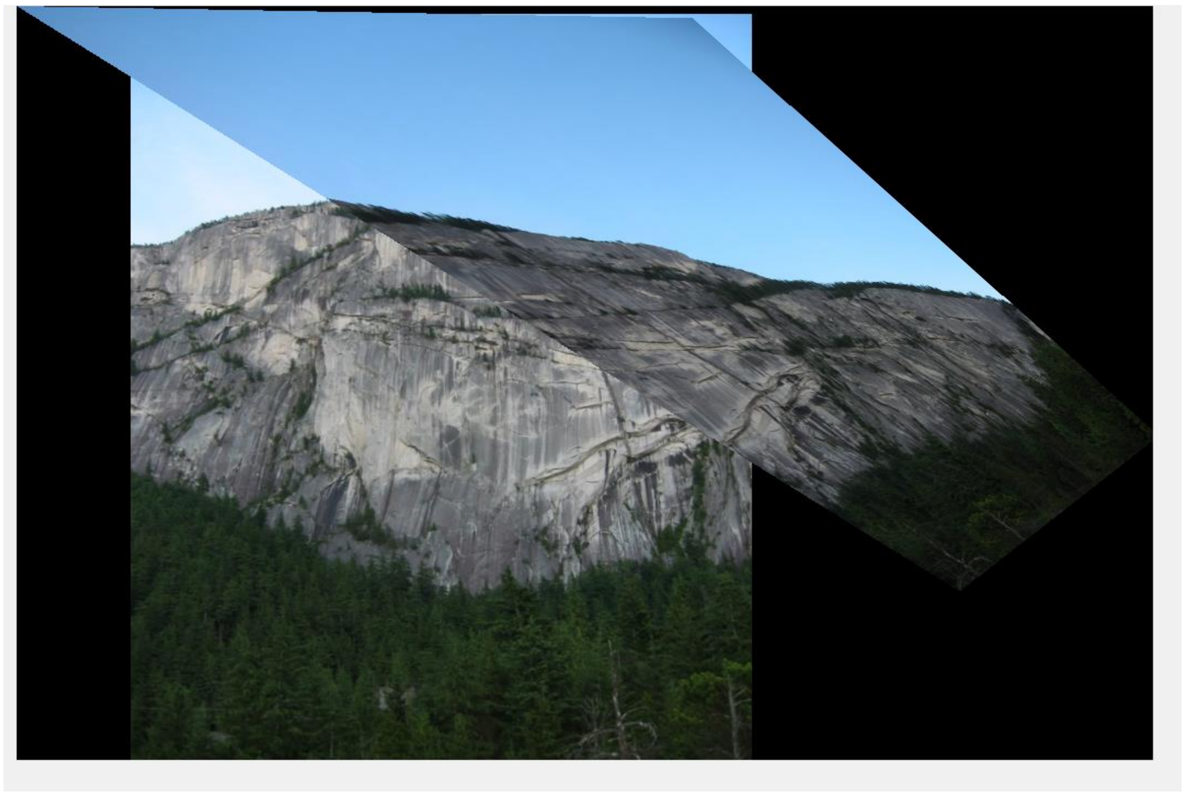

Figure 10: Training set 2 output Image. r(2)/r(1)=0.2. Nbest =450.

- In the training set 2, we had the above picture of a hill. The output of this is bad because I couldn't get features matched. As long as there are wrong feature matches, the Homography matrix will give wrong results as seen in the above image.
- The similar condition applies to input set 3 where the images' feature mapping is not good and hence stitching and alignment could not be done accurately.
- Also, if the ratio r(2)/r(1), i.e. used during matching is kept really small, then the number of matches that happen decreases leading to very few points being matched.

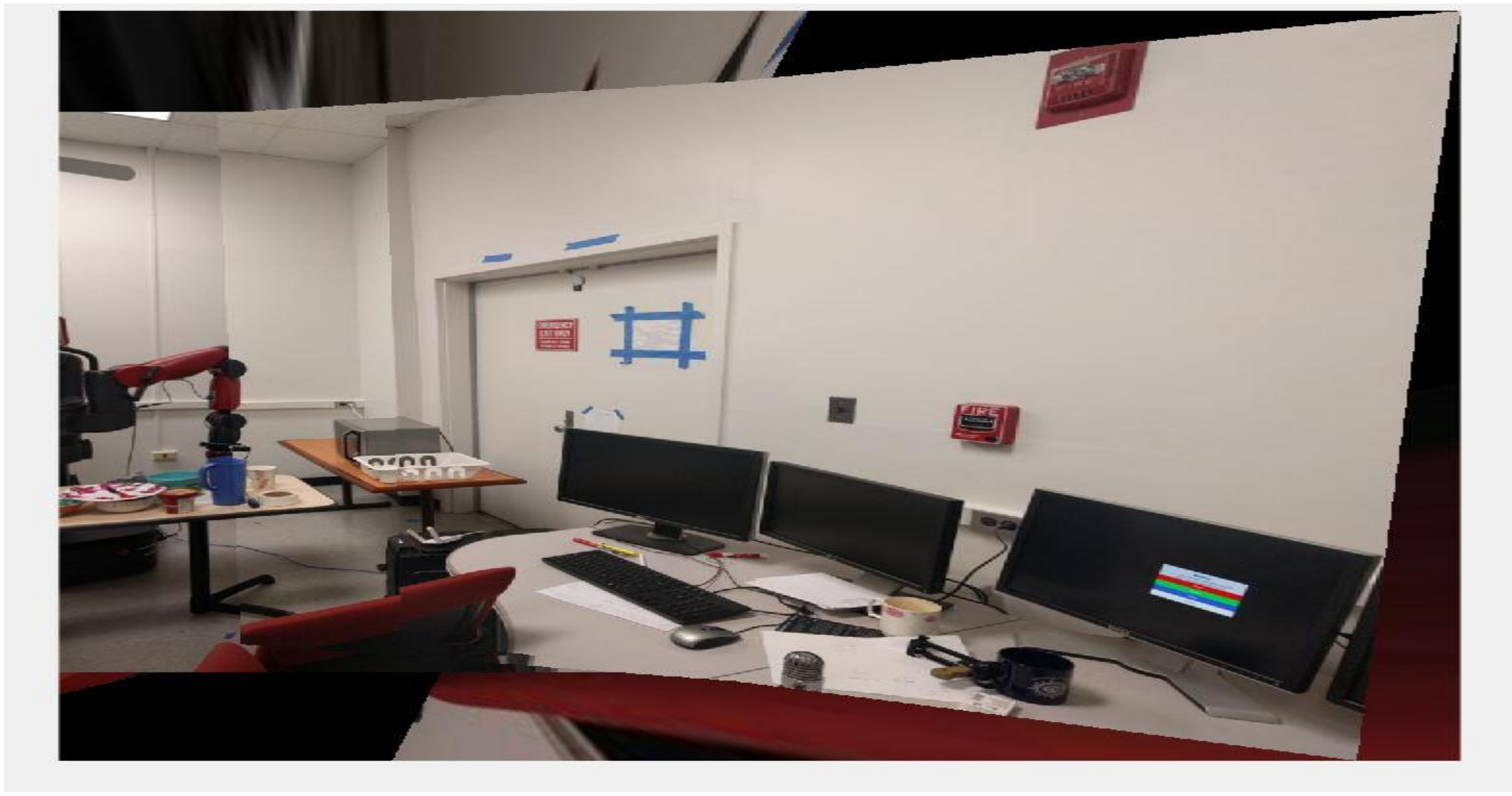

Figure 11: Last 4 images from set 3 of training images.

- The best output for set 3 of training images was as shown above. Here, though there were few cases where, features were mapped/matched correctly, there were lots of mismatches leading to wrong Homography matrices. Especially in the first 4 images. Hence, I took the last 4 images to stitch as they had good feature matching. If too much restriction is put on thresholds like r(2)/r(1) or RANSAC threshold, Homography matrices cannot be computed as there wouldn't be 4 points available many times.

REFERENCES:

[Link 1]