Project Three Report : Matching Pipeline

Negar Nejatishahidin
Computer Science
George Mason University
Virginia, USA
nnejatis@gmu.edu

***Abstract*—This is a report for project three.**

I. P3.1 Scaling and Rotating Patches

In this assignment I first do the unrotate and then unscale and then traslate to the center.

Here is the each matrix:

$$traslate = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$unscale = \begin{bmatrix} \frac{1}{s} & 0 & 0 \\ 0 & \frac{1}{s} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$unrotate = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Here is the H matrix :

$$H = traslate * unscale * unrotate$$

Here is the figure with $base_center_x = 500, base_center_y = 640$:

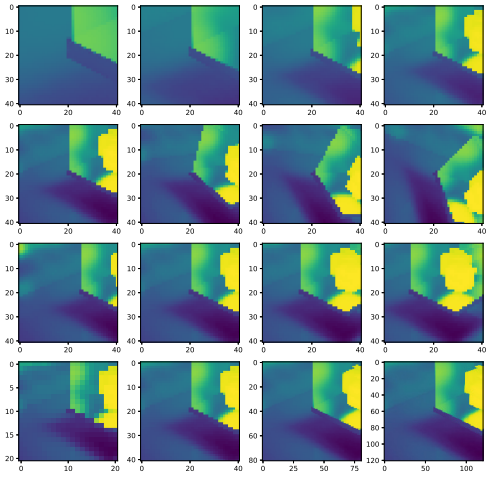


Fig. 1. Result with $base_center_x = 500, base_center_y = 640$.

II. P3.2 Computing Homographies from Matches

There was Homographies are shown in figure 5.

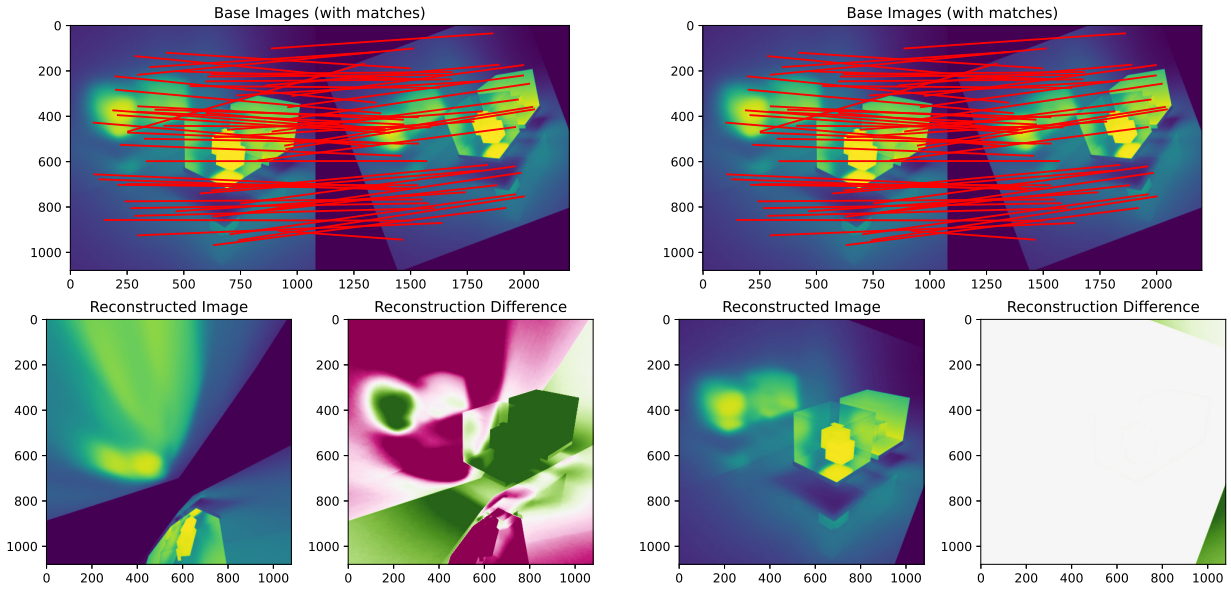
III. P3.2.2 Computing Homographies from Noisy Matches

In the first algorithm that we have used, there is no method to omit the noise or just ignore them, so the noise will have a huge effect on the homography that we have computed. In other words the method is not robust to the outliers. The result is shown in figure 2(a).

The second method is Ransac algorithem. In this pipline we choose 2 random points. Then we compute the distance of other points to the line that passes from the 2 random points. Set a threshold and continue to fined the best line that number on inliers are less than the threshold . In this method we can easily see that the it is robust to outliers. Therefore, the computed homography is much better than the previous method. The results are shown in 2(b). I put the results of this question to more easily see the difference.

I have attached the Ransac code below.

```
def solve_homography_ransac(matches, rounds):
    in_max = []
    thresh = 10
    computedH = None
    for i in range(rounds):
        inliers = []
        #take 4 random points
        random_num = np.random.randint(low=0,
                                high=len(matches),size=4)

        randomFour = matches[random_num]
        #calculate homography with those random point

        h = homography(randomFour)

        for i in range(len(matches)):

            match =  matches[i]
            p1 = np.transpose(np.matrix([match[0],
                                match[1], 1]))

            #our estimation with the homography
            estimatep2 = np.dot(h, p1)
            estimatep2 = (1/estimatep2.item(2))*estimatep2

            #Actual
            p2 = np.transpose(np.matrix([match[2],
                                match[3], 1]))

            # Error is the difference
            error = p2 - estimatep2

            dist = np.linalg.norm(error)
            if dist < 10:
                inliers.append(matches[i])

        if len(inliers) > len(in_max):
```



(a) Homography with all noisy matches.

(b) Homography using Ransac.

Fig. 2. Computing homographies.

```
            in_max = inliers
            computedH = h

    return computedH
```

The H matrix is as follows.

$$H = \begin{bmatrix} 8.00000000e-01 & 3.00000000e-01 & -1.37303007e-09 \\ -3.00000000e-01 & 8.00000000e-01 & 2.20000000e+02 \\ 1.71741988e-15 & 3.76878252e-15 & 1.00000000e+00 \end{bmatrix}$$

IV. 3.3 Feature Matching Pipeline

The reason that I did not get the good result was because the program needs a better computational power. I only run the program for 4 or 5 sigma and it takes around 30 minutes, but If I could do it for 40 or more sigma I would defiantly get a better result. The result has shown at figure 3.

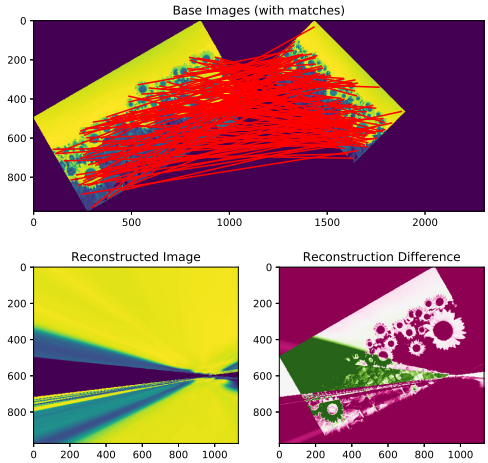


Fig. 3. Feature Matching result.

V. 3.4 Feature Matching with OpenCV

I have used the openCV pip-line to do the feature matching in this part. I am going to compare them from 4 different perspective :

- Number of features: Open CV has much more features than what I have in both the number and the accuracy of the features.
- Accuracy and matches: OpenCV by far perform better in both terms of accuracy of feature detection and accuracy of the matches.
- How much faster: OpenCV by far is a faster algorithm it takes few second to run, but mine takes around 30 minutes or even more for around 6 or 7 sigma.

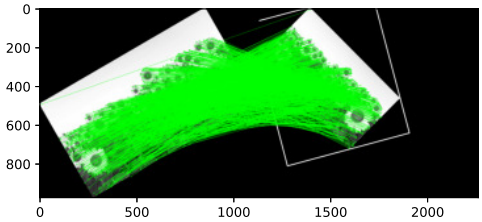
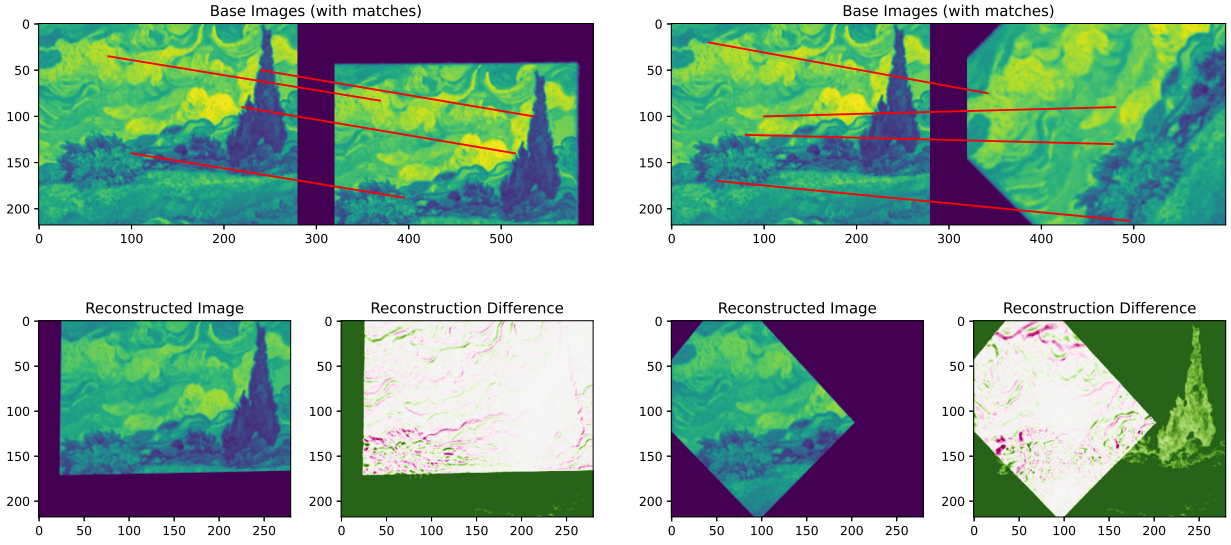
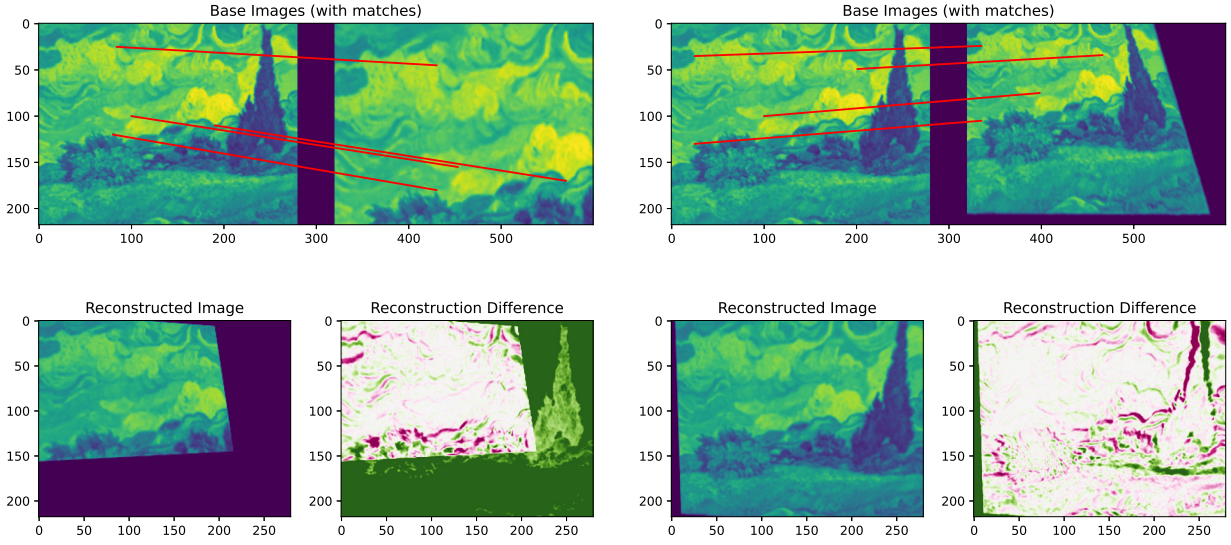


Fig. 4. Feature Matching of OpenCV.



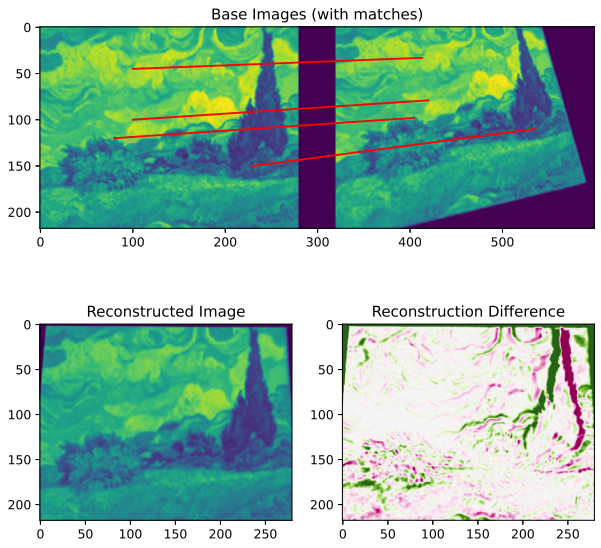
(a) Translation

(b) Rotation 2



(c) Scale

(d) Homography A



(e) Homography B

Fig. 5. 5 different transformations.