P3 Matching Pipeline

Siddharth Patel

I. P3.1.1 Scaling and Rotating Features: Concepts

A. *If I have a feature located at (x_f, y_f) with orientation θ and radius ("scale") s, what is the transformation matrix H that simultaneously moves the feature to the origin, un-rotates it, and un-scales it (so that the feature becomes 1 pixel wide)?*

As defined above, we are coming up with a matrix that will revert the transformed image to its original state. First we take care of scaling the image or to say "unscaling" the image because we are reverting it to the original form by dividing the pixels by radius. In the x orientation, we use cos of theta while using sin in y orientation. This method un-scales, un-rotates the image. Finally, by doing 0 0 1 to the z coordinates, we are making each pixel wide by 1.

$$\begin{bmatrix} \frac{1}{s} \cdot -cos(\theta) & sin(\theta) & x' - x \\ -sin(\theta) & \frac{1}{s} \cdot -cos(\theta) & y' - y \\ 0 & 0 & 1 \end{bmatrix}$$

II. P3.1.2 Scaling and Rotating Features: Implementation

Using the solution from the previous question, I implemented the transformation matrix for the get_scaled_rotated_patch function. After that, I applied changes to the image one by one by searching through the new coordinates and applying patches to the image using the bi-cubic interpolation. Given the base_center_x = 800 and base_center_y = 600, the output will be as followed. Once I was able to compute the image using base x and y, I changed the base_center_x value to 500 followed by base_center_y value to 640. As a result I got the following image output which seems appropriate.

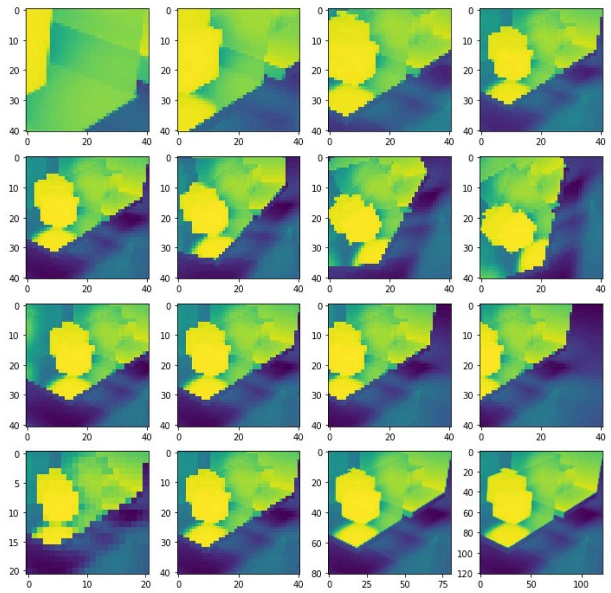
Fig. 1. 800 x 600

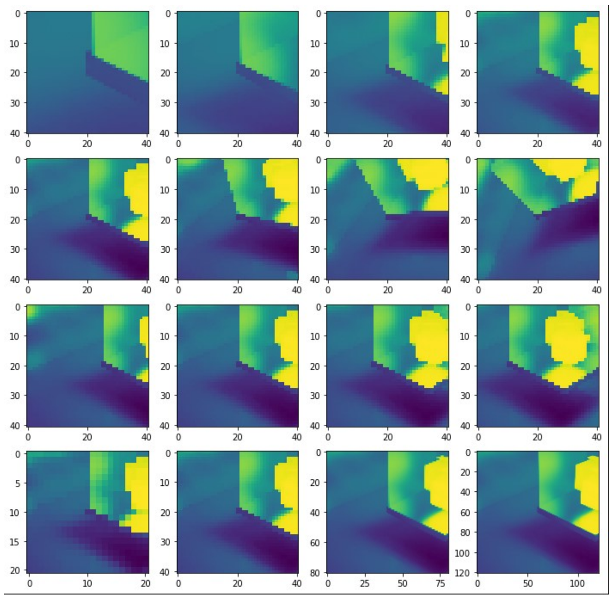
Fig. 2. Caption

III. P3.2.1 Computing Homographies from Perfect Matches

To understand the process of solving homography, I looked at the given example (you can view it below) which transforms an image by rotating, scaling, and/or translation, provided some matches, and uses those matches to compute the transformed image. Throughout this section, you also display the reconstructed image, followed by the difference between base and transformed image. Along with that, I also understood the process my looking at the given matrix.

$$\text{base matches} = \begin{bmatrix} 30 & 125 & 36 & 150 \\ 20 & 30 & 25 & 35 \\ 80 & 120 & 96 & 144 \\ 220 & 90 & 264 & 108 \end{bmatrix}$$

$$\text{tr matches} = \begin{bmatrix} 255 & 13 & 232 & 64 \\ 161 & 117 & 137 & 164 \\ 56 & 123 & 31 & 173 \\ 168 & 88 & 145 & 140 \\ 135 & 160 & 105 & 210 \\ 122 & 10 & 95 & 55 \end{bmatrix}$$

$$\text{ro matches} = \begin{bmatrix} 35 & 130 & 135 & 192 \\ 35 & 40 & 35 & 105 \\ 145 & 135 & 243 & 110 \\ 158 & 135 & 255 & 78 \\ 60 & 125 & 140 & 152 \\ 85 & 179 & 220 & 183 \\ 170 & 92 & 215 & 22 \\ 97 & 99 & 153 & 100 \end{bmatrix}$$

$$\text{sa matches} = \begin{bmatrix} 210 & 113 & 295 & 175 \\ 55 & 115 & 75 & 178 \\ 155 & 128 & 205 & 200 \\ 22 & 128 & 30 & 205 \\ 18 & 23 & 26 & 47 \\ 187 & 78 & 241 & 122 \end{bmatrix}$$

$$\text{ha matches} = \begin{bmatrix} 163 & 88 & 129 & 65 \\ 51 & 123 & 38 & 99 \\ 250 & 8 & 176 & 2 \\ 156 & 112 & 119 & 82 \\ 228 & 77 & 173 & 57 \\ 147 & 218 & 128 & 192 \\ 275 & 80 & 208 & 60 \\ 24 & 36 & 19 & 24 \end{bmatrix}$$

$$\text{hb matches} = \begin{bmatrix} 230 & 74 & 198 & 48 \\ 213 & 113 & 192 & 77 \\ 158 & 127 & 151 & 98 \\ 54 & 71 & 54 & 52 \\ 60 & 210 & 75 & 200 \\ 50 & 20 & 50 & 10 \\ 150 & 25 & 120 & 20 \end{bmatrix}$$

The five more transformed image along with its matrix which uses similar concept as explained above of rotating, translating, and/or scaling are listed below.

Note: all the matrix and images are chronologically aligned.

IV. P3.2.2 Computing Homographies from Noisy Matches

We basically use the RANSAC solution to determine which features are inliers and outliers in our image. Just as we did in the above computation, I am using the RANSAC solution rather than the solve_homography function to compute the homographies. Once again, we are using the visualizing function from the above section to display the image. Now the one thing that sets the RANSAC solution from the solve homography one apart is that RANSAC accounts for outliers or bad feature matches, but the solve homography from the above equation does not do so. As you can see in the output images attached, the base image comparison which uses the solve homography function has lots of noise in the image. Now checkout the RANSAC solution. You can see in the reconstruction difference that there is no noise

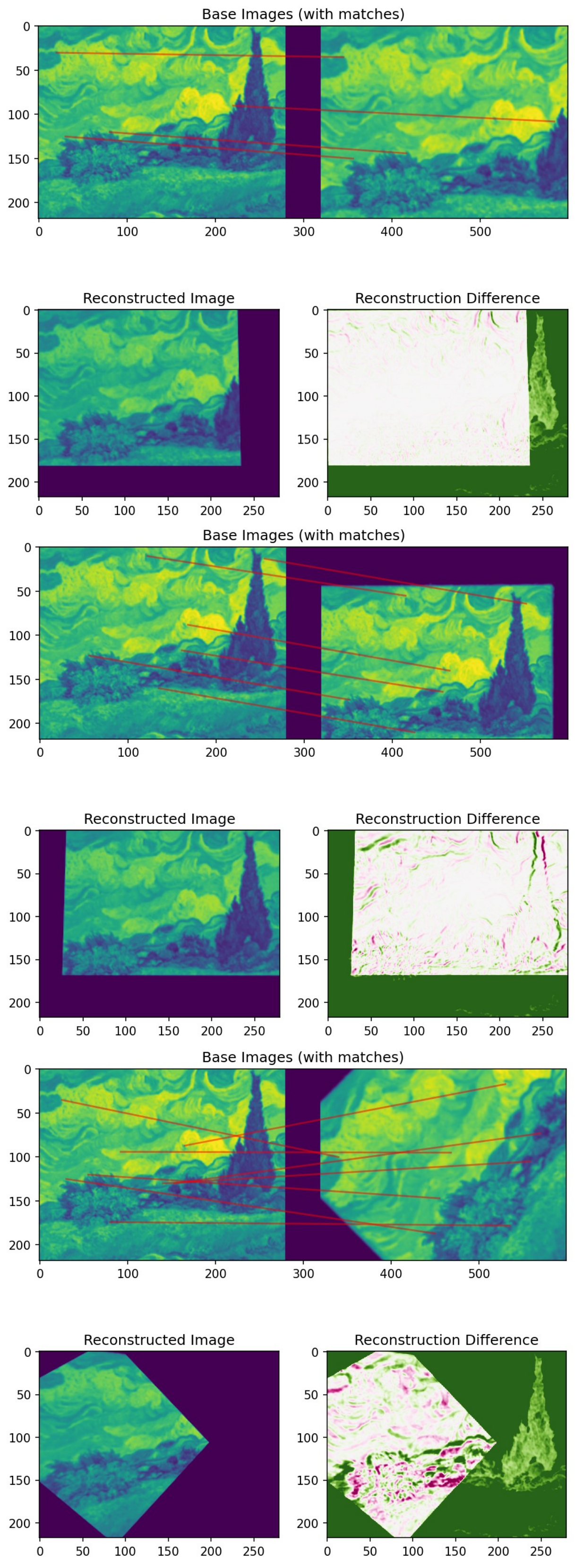

Fig. 3. 3.2.1 Base Image and Difference

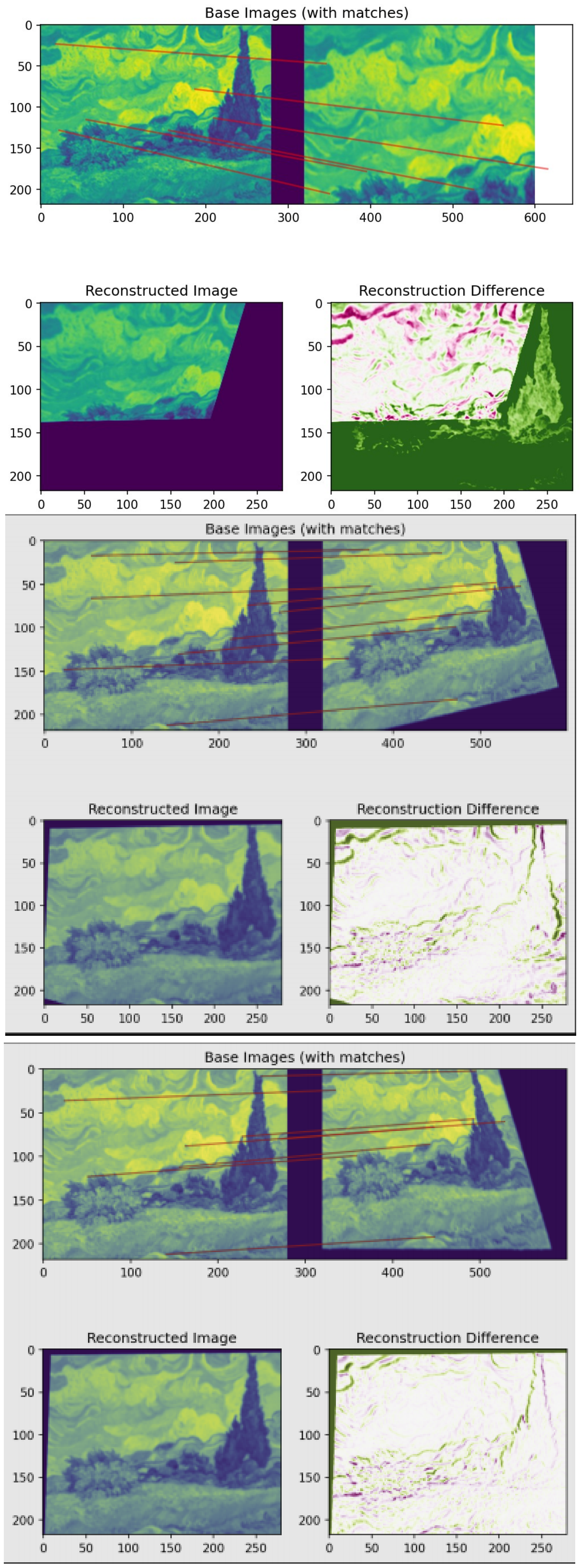

Fig. 4. 3.2.1 Reconstructed Images

throughout the image, indicating that we have taken care of the outliers in the image.

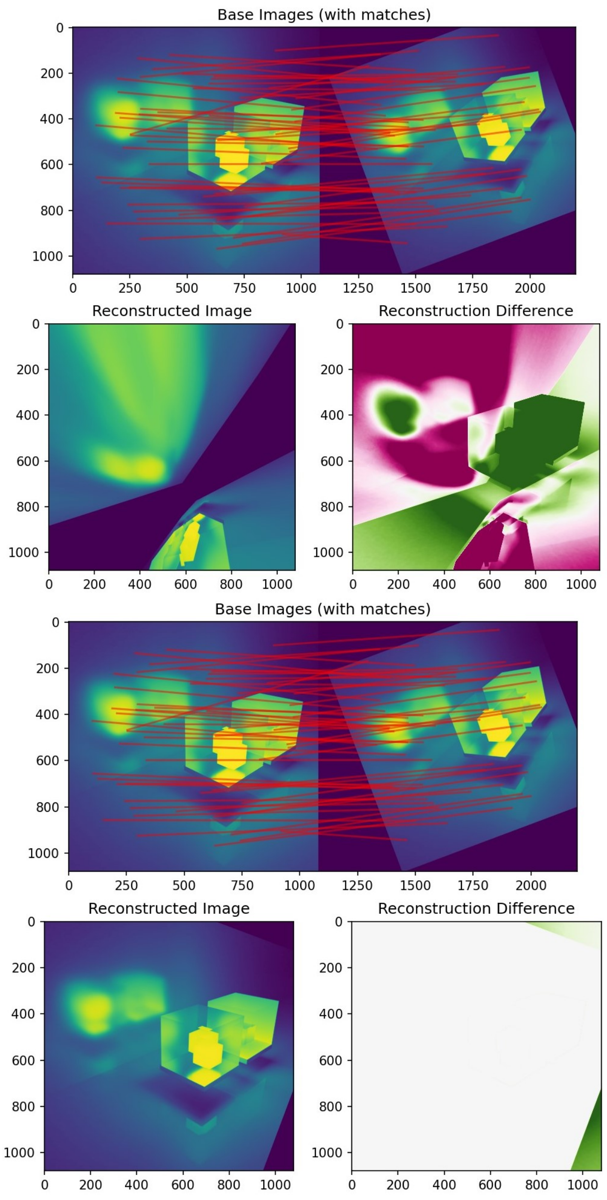

Fig. 5. 3.2.2 Base and Noise Reduced Images

V. P3.3 Feature Matching Pipeline

Using the code from previous project, project 2, I implemented the compute feature matches function which returns feature matches given two images. Using the sunflower base image, we are matching the two images to see the reconstructed image difference. The output of the whole computation will look like this:

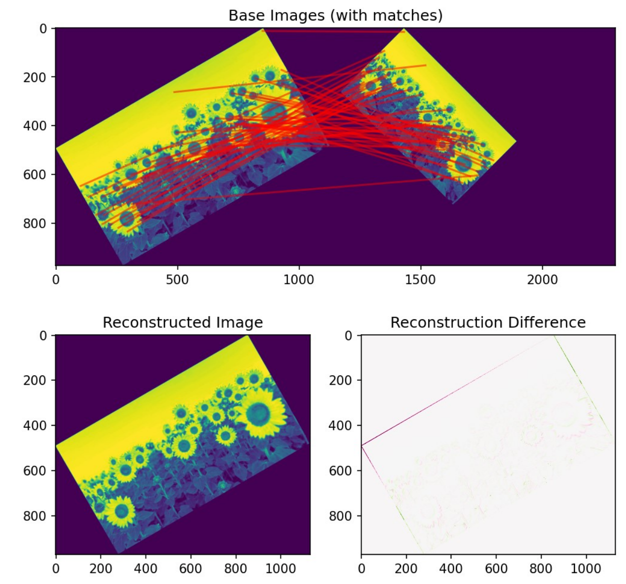

Fig. 6. Feature Matching Pipeline

VI. P3.4 Feature Matching with OpenCV

Lets talk about the performance of OpenCV compared to our code. Clearly, OpenCV solution is extremely faster than our solution and the reason for this difference might be due to the optimization of OpenCV. Depending on the background processes that were running on my PC and comparing OpenCV solution to the code that I implemented, I would say that OpenCV solution is 2 - 7 minutes faster that my solution however the speed may vary based on the number of background processes that are using my CPU. OpenCV gives you your output in about 1-2 minutes.

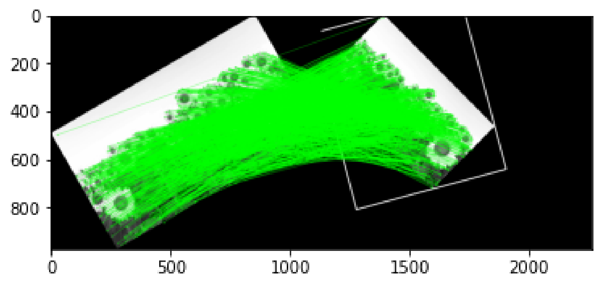

Fig. 7. Sunflower matching using OpenCV