Transformation Matching Pipeline

Nooreldean Koteb - Team Member: Abdallah Soliman

I. INTRODUCTION

This project was centered around homographies and feature matching. For the first part patches of features from an image are found. For the second part a Homography matrix is calculated using the A-matrix that is computed using the given matches. For the third part matches in images are calculated and marked.

II. SCALING & ROTATING PATCHES

A. Features – Concepts

The cosine theta rotates an image while the opposite signed theta rotates an image back to the original. The first two diagonal values are used for scaling, and the last values of the first two rows in the matrix transform the image. This matrix was creates by combining those three properties. **QUESTION** – If I have a feature located at (x_f, y_f) with orientation θ and radius ("scale") s, what is the transformation matrix $\boldsymbol{H}$ that simultaneously moves the feature to the origin, un-rotates it, and un-scales it (so that the feature becomes 1 pixel wide)?

$$\boldsymbol{H} = \begin{bmatrix} \frac{\cos(-\theta)}{s} & \frac{-\sin(-\theta)}{1} & -x_f \\ \frac{\sin(-\theta)}{1} & \frac{\cos(-\theta)}{s} & -y_f \\ 0 & 0 & 1 \end{bmatrix}$$

B. Features – Implementation

Matches was used to compute the A-matrix. Numpy was then utilized to compute the Homography H-matrix. The figure below was used to compute the A-matrix.

The full matrix form looks like:

$$\underset{2n\times 9}{\mathbf{A}} \quad \underset{9}{\mathbf{h}} = \underset{2n}{\mathbf{0}}$$

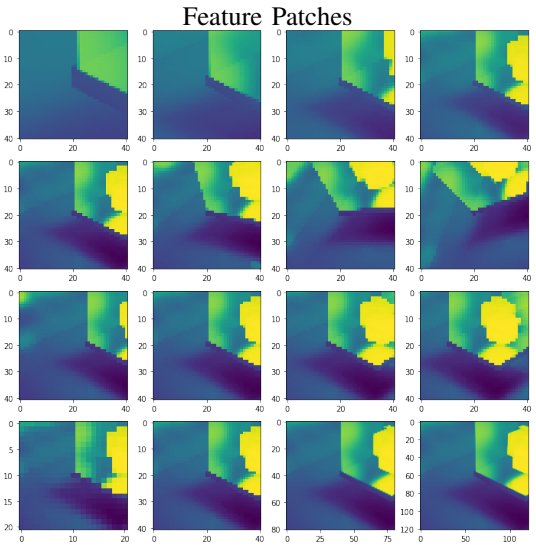
$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1x_1 & -x'_1y_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1x_1 & -y'_1y_1 & -y'_1 \\ & & & & \vdots & & & & \\ x_n & y_n & 1 & 0 & 0 & 0 & -x'_nx_n & -x'_ny_n & -x'_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -y'_nx_n & -y'_ny_n & -y'_n \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

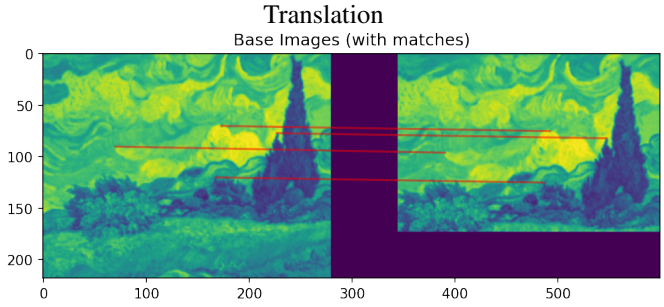
III. COMPUTING HOMOGRAPHIES FROM MATCHES

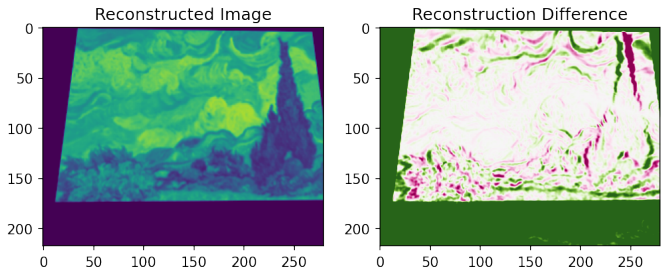
Homographies were computed from perfect matches and noisy matches.

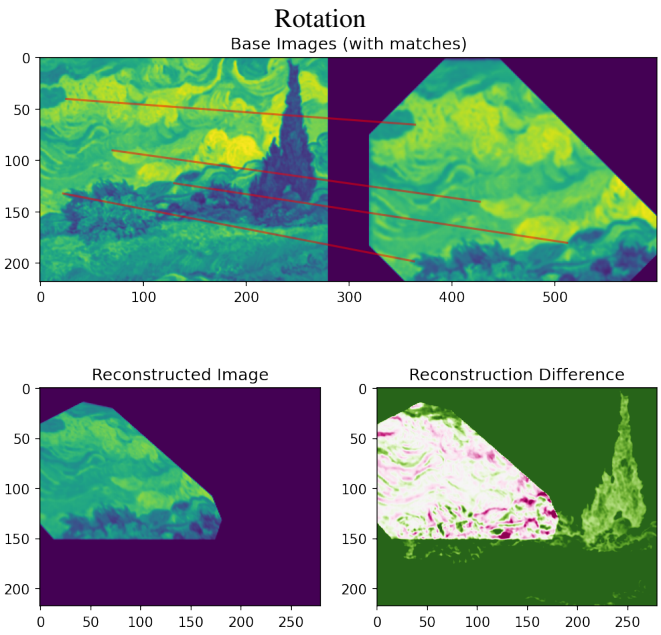
A. Perfect Matches

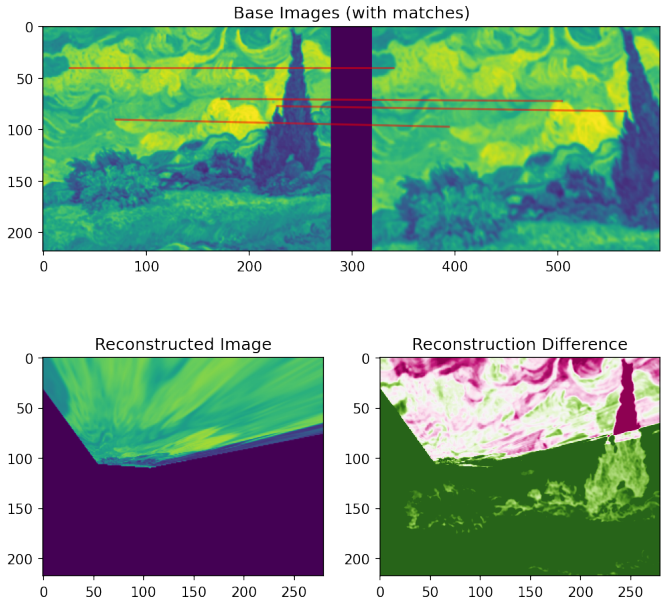
The perfect matches were computed by finding the H known and inserting it into the transformed image function.

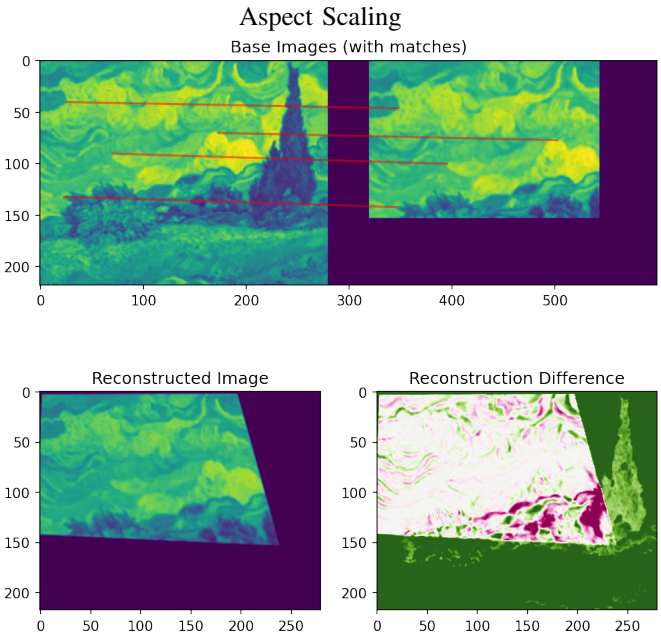


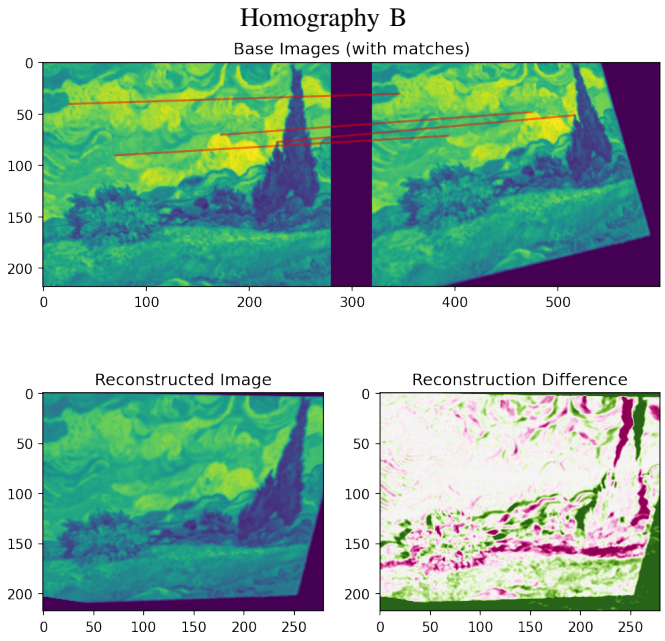










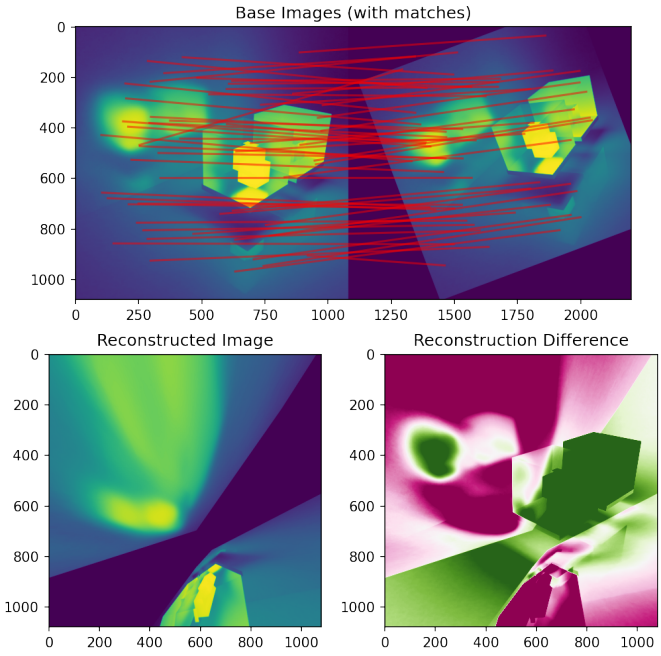


Homography A

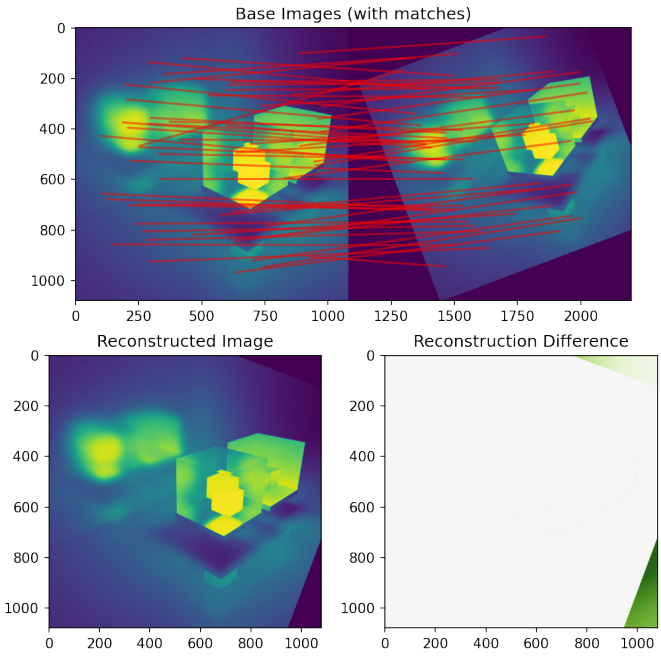
B. Noisy Matches

For Noisy matches the solve_homography_ransac() was utilized because the regular solve_homography() function includes outliers which makes it incapable of calculating the homography. The ransac function excludes those outliers.

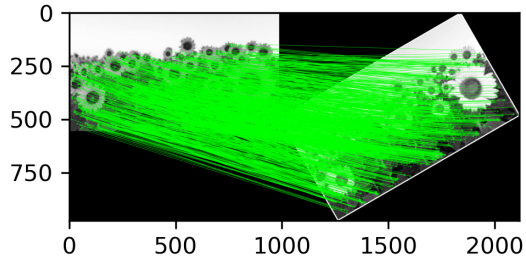
Regular homography function



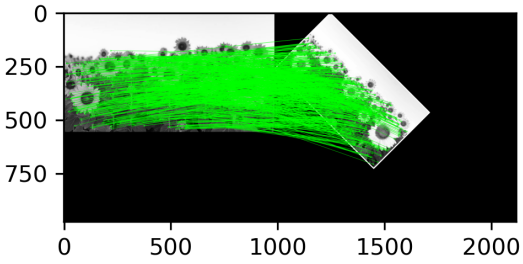
Ransac homography function



Sunflower matching



Sunflower matching



IV. FEATURE MATCHING PIPELINE

In the last project I was unable to figure this part out, and I couldn't figure it out this time either. The only thing I had from last project was the Log_filter() function.

V. FEATURE MATCHING WITH OPENCV

Although the feature matching pipeline was not implemented I still wanted to implement this part of the code. It was very interesting to see matching done by this library. It can not be compared to mine since I was unable to implement it, but I am sure it is much faster since open CV uses C code to be as efficient as possible.