Electrical and Computer Engineering
University of Thessaly (UTH)

ECE457 - Computer Vision

Fall Semester — Educational year 2024-2025

Image Stitching Project

Vasileios Stergioulis - AEM: 03166

Dimitrios Tsalapatas - AEM: 03246

Emmanouil Pantopoulos - AEM: 03222

Abstract

The goal of this project is to create a program that takes N images with partial overlap and returns the images stitched together into a panoramic image.

Contents

1 Algorithm

The algorithm of this project can be summarized in the following steps:

Algorithm 1 Image Stitching

1: **Input:** N images
2: **Output:** Stitched image $I_{stitched}$
3: **Step I:** Read two images.
4: **Step II:** Compute SIFT features (keypoints and descriptors) for both images.
5: **Step III:** Find matching feature points between I_1 and I_2.
6: **Step IV:** Estimate the best Homography Matrix **H** using RANSAC.
7: **Step V:** Warp one image onto the other using **H**.
8: **Step VI:** Blend the images using one of the following methods:

- Simple Blending
- Gaussian Blending
- Laplacian Blending

9: **Step VII:** If $N > 2$, repeat.
10: **Return:** $I_{stitched}$

The images, are loaded as `numpy` arrays, using `imageio`'s function `imread`. In our program we load the first two images and after stitching is performed, we continue the process with the the stitched and the next N-2 images.

2 Feature Computation & Selection

In order to extract SIFT features from our images, we used `opencv`'s integrated Sift detector. The detector, returns two lists, one with the keypoints and one with keypoint descriptors. The resulting SIFT features are illustrated in **Figure 1**. After extracting our features, the next step is to find the matches between them, using `cv2.BFMatcher` or Brute-Force Matcher. The matcher takes two sets of feature descriptors (one set per image) and finds the closest matches based on Euclidean distance. In our case, we also apply Loewes thresholding technique or ratio test, as shown in `opencv`'s tutorial. The resulting matches between images, are shown in **Figure 2**.

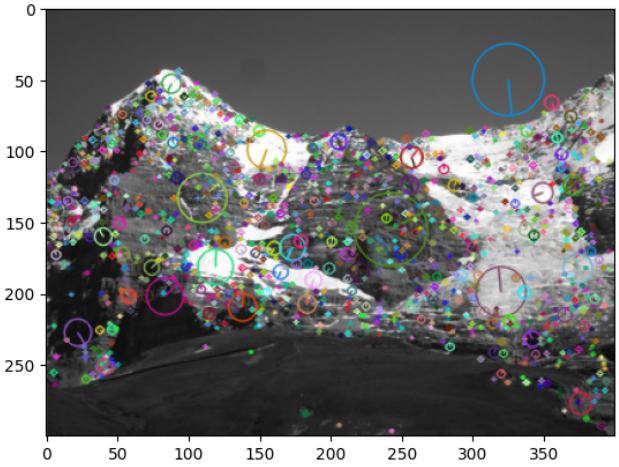


Figure 1: SIFT features

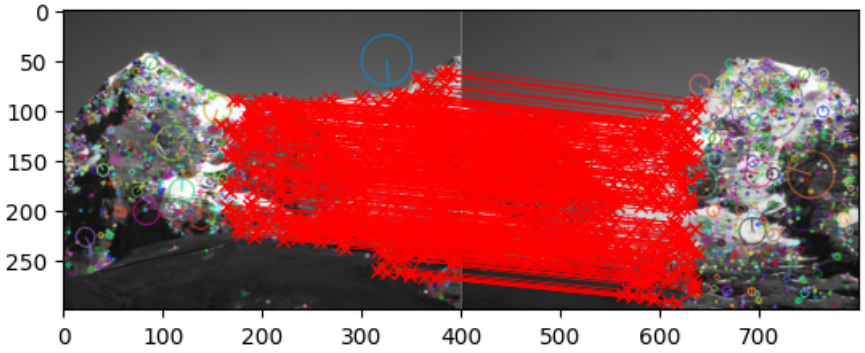


Figure 2: Matches and SIFT features

3 RANSAC

The first step for implementing RANSAC, is to choose four random points (the least number of points needed), in order to compute the homography matrix H. In order to compute H, we use the following formula:

$$\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

$$A = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_2x_1 & -x_2y_1 & -x_2 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y_2x_1 & -y_2y_1 & -y_2 \end{bmatrix} = 0$$

which is solved using the constrained squares technique. For constrained least squares, we transpose matrix A to A^T, compute A^TA and find the minimum eigenvalues, eigenvectors using `numpy`'s `linalg.eigh` function. The error function estimates the keypoints in the second image using H and calculates the Euclidean distance between these estimated points and the corresponding ground-truth keypoints. Note that for our error function, we don't use the initial four random keypoints, but all of them! That means that every point is assigned an error. To identify inliers, we set a threshold

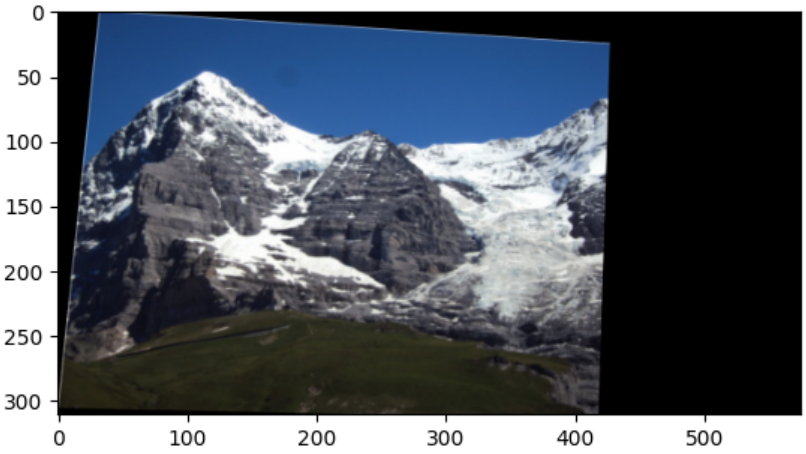
and classify a point as an inlier if its error is below this threshold. The best homography matrix, is the one with the most inliers.

4 Warping and Stitching

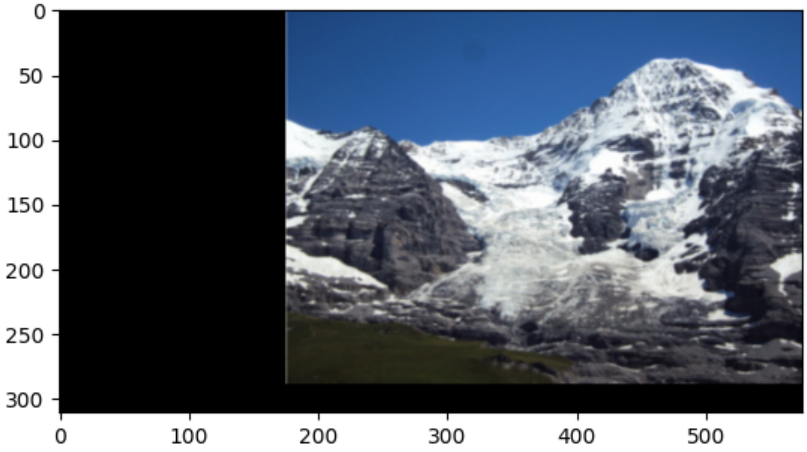
After executing RANSAC, we find the best homography matrix, which means that now we need to warp our images. Before applying the warp transform we need to resize our images and compute the translation matrix *T*, for our non-warp image. Essentially T, allows as to correctly resize the image, with its formula being:

$$T = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix}$$

, where dx, dy denote how much we want to resize our image. Finally, for warping, we use `open-cv` in-built function `warpPerspective`. An example of our warped image can be seen in **Figure 3**.



(a) Warped Image with H



(b) Warped Image with T

Figure 3: Warping and resizing images

For stitching our images, we used three simple blending techniques:

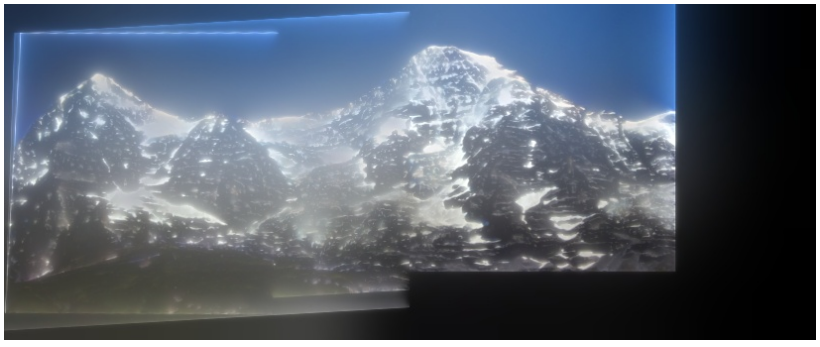
- Simple Stitching
- Gaussian Stitching
- Laplacian Stitching

For simple stiching we simply overlay the first image to the second. If a pixel is only in the first image, it replaces the corresponding pixel in second, if a pixel is in both images, it blends them evenly (50-50 mix).

For the last two methods, Gaussian and Laplacian pyramids are used. The Gaussian pyramids create multi-scale representations of the images, while Laplacian pyramids to blend images at different scales smoothly. The resulting stitched images can be seen in **Figure 4**.

(a) Simple Blending

(b) Gaussian Blending



(c) Laplacian Blending

Figure 4: Stitching Results

5 Stitching Results and Challenges

We can clearly see that the best from our blending methods is **Gaussian Blending**, due to its smoothness in mixing, as we can clearly see the smoothed out image edges or margins. Simple blending, produces a similar result with the gaussian, but the margins are still visible, while laplacian produces the worst image so far, due to its multi-scale decomposition to blend images at different frequencies. As it can be easily seen by our code, in terms of computation cost, simple blending is the least computationally expensive algorithm.

From our figures, we deduct that our implementation is satisfying the project goal. Unfortunately, we had some problems when generalising to N images:

I After stitching our first two images, we had a problem with resizing and correctly centering the the third or fourth image. In our algorithm we warp the first image in order to fit it to our second, which usually is located to the far right corner (see **Figure 3b**). After applying the Translation transform, our problem was solved

II Another re-occuring problem, was the weakness of identifying SIFT features on the stitched image, which also meant lesser accuracy when stitching the third or fourth image.

III Finally, feature matching became less reliable due to perspective and lighting changes.

An example of the above, is illustrated in **Figure 5**, where after stitching our two images, the SIFT features aren't computed correctly and the resulting panorama looks a bit off.

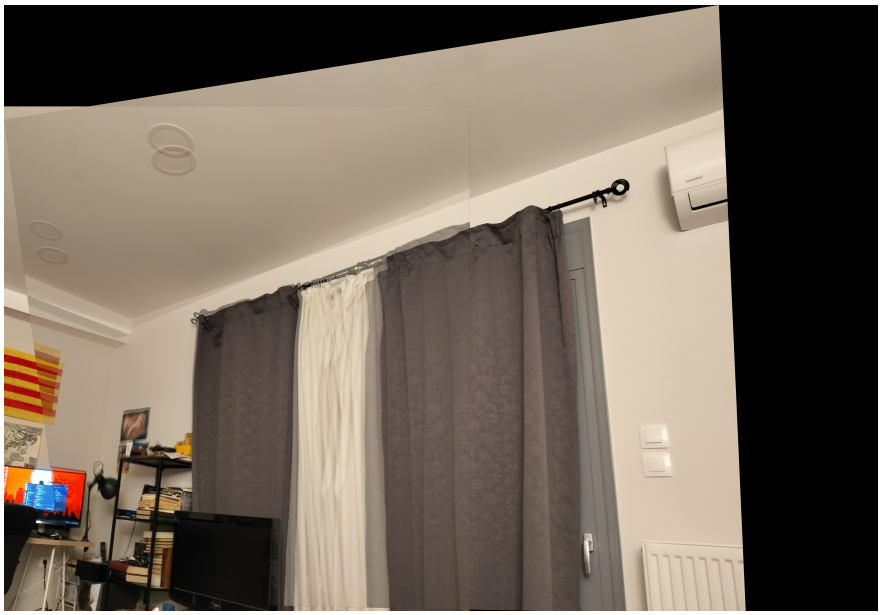


Figure 5: Example of bad Stitching

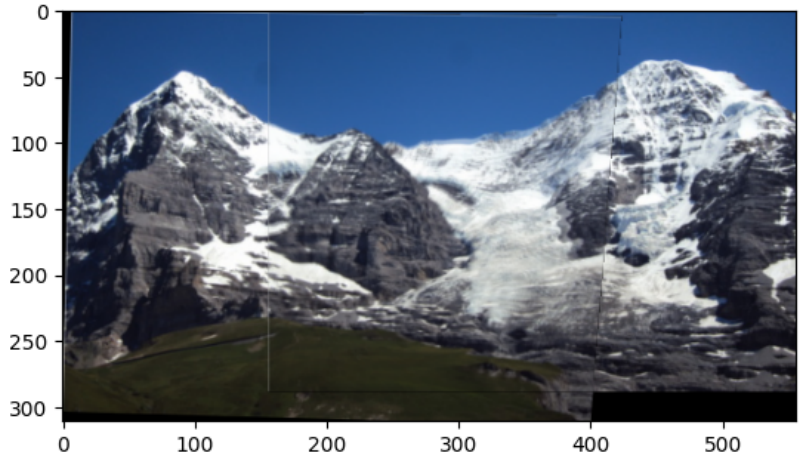
6 Effect of Using a Random Subset of Keypoints

Using a random subset of matching keypoints for homography estimation can lead to several problems. With fewer keypoints, the computed homography is less accurate, often causing mis-

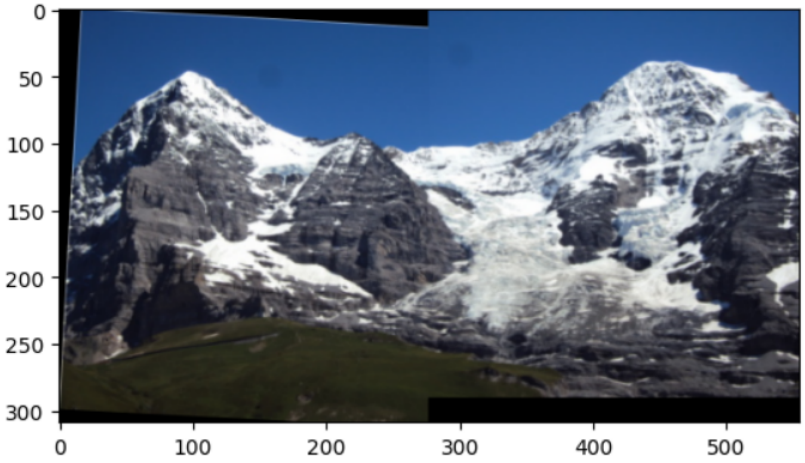
alignment. If the selected keypoints are not well distributed, the estimation might fail entirely. Generally, using more keypoints improves accuracy and stability, but beyond a certain point, the extra computational cost outweighs any real benefit. The above can be easily seen from the following Homography Matrices 1 and 2:

$$H_1 = \begin{bmatrix} 1.0303 & -0.0664 & -146.7754 \\ 0.0729 & 0.9805 & 5.7520 \\ 0.0001 & -0.0002 & 1.0000 \end{bmatrix}, \text{Example of bad Homography Matrix } H_1$$

$$H_2 = \begin{bmatrix} 0.9893 & -0.0501 & -138.6070 \\ 0.0553 & 0.9873 & 9.1882 \\ 0.00001 & -0.00004 & 1.0000 \end{bmatrix}, \text{Example of good Homography Matrix } H_2$$



(a) Bad H_1



(b) Good H_2

Figure 6: Example of a bad and a good Homography matrix

The implemented RANSAC method selects a subset of four points per iteration and iteratively refines the homography by minimizing errors, ensuring a balance between accuracy and efficiency.