# Visual Computing Assignment 1

Kornidesz Máté

*26th of September 2025 (Deadline)*

## Content

Tasks and Overview ..... 1
Tasks ..... 1
1. Task: Dataset and Capture ..... 1
2. Task: Feature Detection and Matching ..... 2
3. Task: Homography Estimation Experiments ..... 4
4. Task: Blending the images ..... 6
Discussion ..... 6
Sources ..... 7

## Tasks and Overview

The objective of this project is to create a panorama stitching system using feature detection, matching and Homography estimation to create the desired panorama. The description of the task required an input image source of 3 sets of images (each containing at least 2, which 2 images must be around 20%-60% overlapping for the desired results). The description of these images can be viewed later in this document (1. Task Dataset and Capture). Each dataset is captured to test the limits of the used methodologies in terms of number of matches, number of detected key points by varying the light conditions and texture richness throughout the images. The dataset was made with a phone camera in a standing position in high resolution (that is why the images made smaller in the "process pipeline" and that's why they have their specified ratio (∽15:27). The pipeline was made in C++ and used the OpenCV library for the rest of the calculations and methods and for simple diagrams as well. The used methodologies were SIFT and AKAZE where the program prints the details specified in Task 2., there is also a possibility to visualize the matchings (not recommended, very dense for the rich input). For RANSAC the threshold was varying between 1.0, 3.0, 5.0 and 10.0, so from use a strict approach where inliers significantly drop, to a more permissive threshold where almost all matches count as inliers. The good balance is around 3 or 5 as the too small threshold can lead to visually unpleasant result if the images not aligned well, and the too big threshold also can lead to stretching in the result, so there may be a "sweet spot". For the last part of the tasks, the images were blended using 2 different techniques: simple overlay and feathering. The 2 methods are compared to each other in terms of speed and result quality.

## Tasks

### 1. Task: Dataset and Capture

The Images can be found on the GitHub repository (Link to Repository). The following details were observed about these pictures:

| Source Name | Pictures with *-s1-* | Pictures with *-s2-* | Pictures with *-s3-* |
|---|---|---|---|
| Title | Indoor Set 1 | Indoor Set 2 | Outdoor Set 3 |
| Short Description | Indoor picture set which contains a wall a curtain and some light behind the curtain | Indoor picture set which contains a corridor and stairs with mailboxes and a door | Outdoor picture set which contains a street with trees and houses and some parked cars |
| Lighting | Poor Lighting | Moderately Good Lighting | Good Lighting |
| Motion Blur | Moderately Sharp Image | Sharp Image | Sharp Image |
| Texture Richness | Lower Texture richness | Moderately High Texture Richness | High Texture Richness |

## 2. Task: Feature Detection and Matching

These pieces of information were also logged to the console in the C++ program: (The images were scaled to 15% of their original size on both axes for better runtime and better viewability of matches on my device)

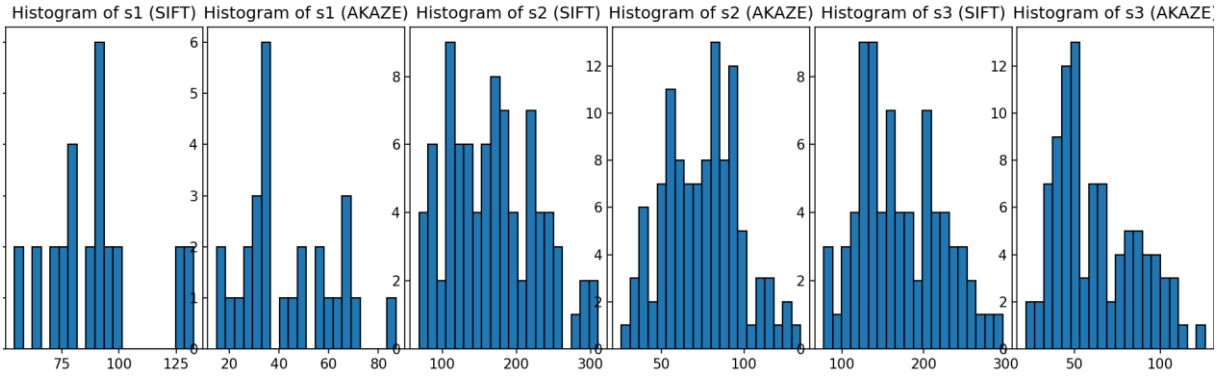
### *SIFT*

It detects key points by finding extrema in scale-space using the Difference of Gaussians. Histograms indicate solid matches even when the images are scaled down as this method is robust to changes in scale and rotation, however it is relatively slow and computationally expensive compared to its modern alternatives.

### *AKAZE*

It builds features in a nonlinear scale-space using diffusion filtering (also nonlinear). This method is better for preserving edges and details compared to the Gaussian blurring. The binary descriptors it produces are compact and allow fast matching with Hamming distance. Faster than SIFT but less distinctive in high texture regions as it is visible from the charts.

| Dataset | Method | Key point 1 | Key point 2 | Matches | Elapsed Time |
|---|---|---|---|---|---|
| s1 | SIFT | 50 | 40 | 14 | 1.2653 ms |
| s1 | AKAZE | 64 | 62 | 28 | 2.8425 ms |
| s2 | SIFT | 486 | 534 | 87 | 73.0941 ms |
| s2 | AKAZE | 496 | 312 | 109 | 8.6341 ms |
| s3 | SIFT | 478 | 708 | 76 | 92.2921 ms |
| s3 | AKAZE | 265 | 421 | 94 | 7.3694 ms |


Histogram of s1 (SIFT)
Histogram of s1 (AKAZE)
Histogram of s2 (SIFT)
Histogram of s2 (AKAZE)
Histogram of s3 (SIFT)
Histogram of s3 (AKAZE)

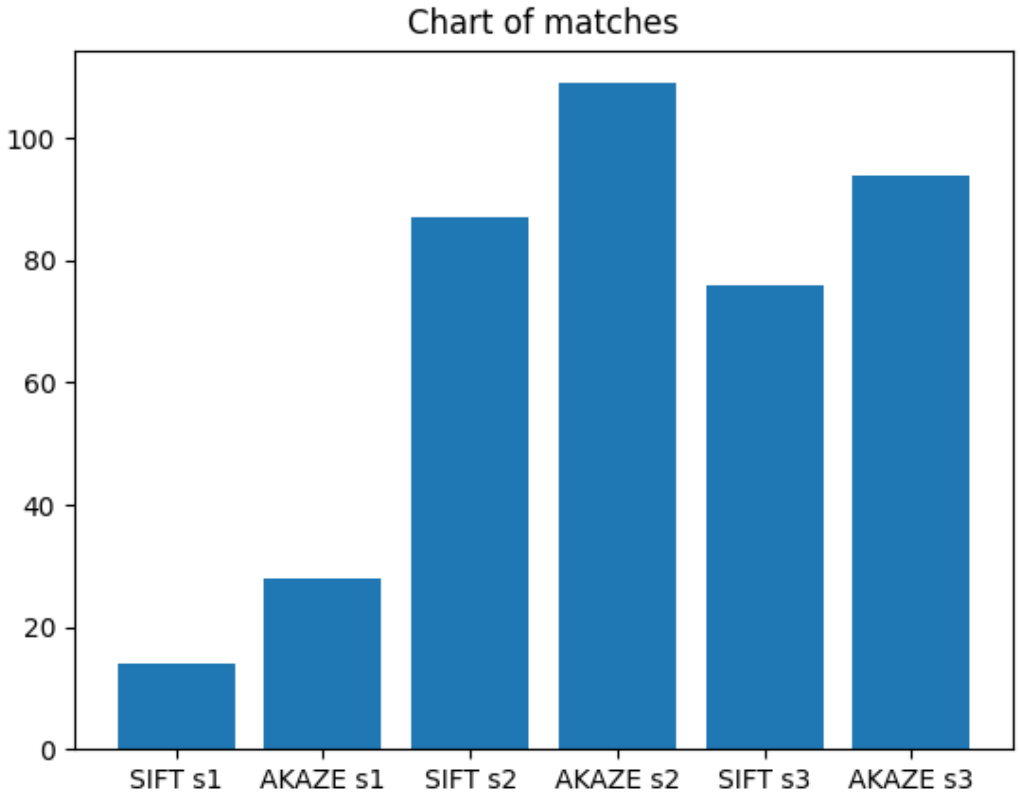
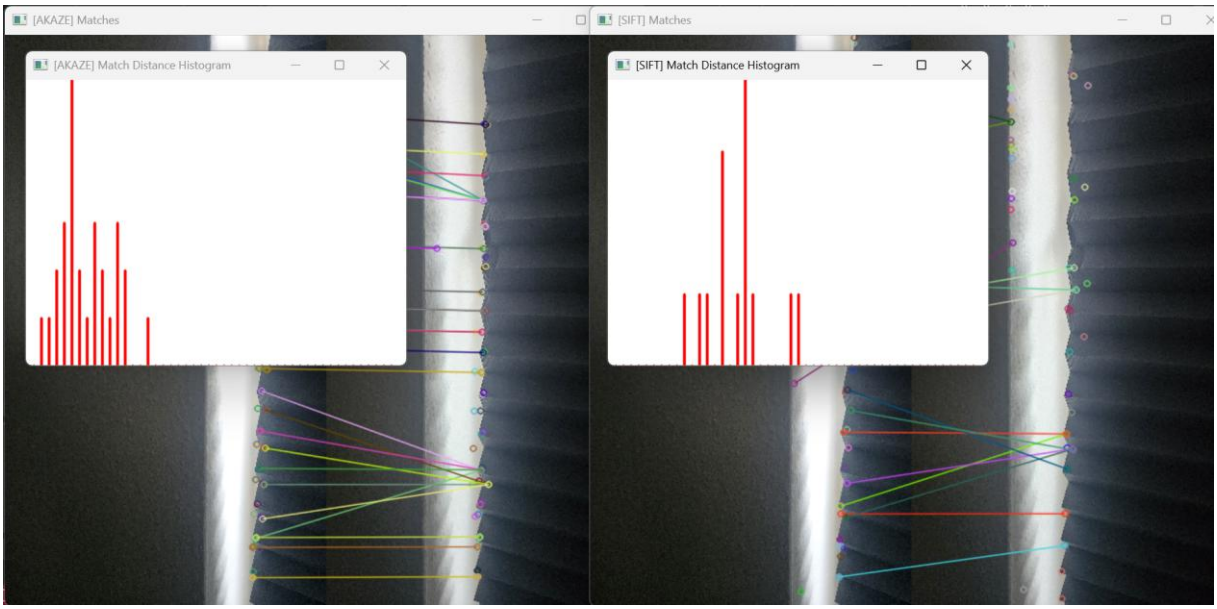
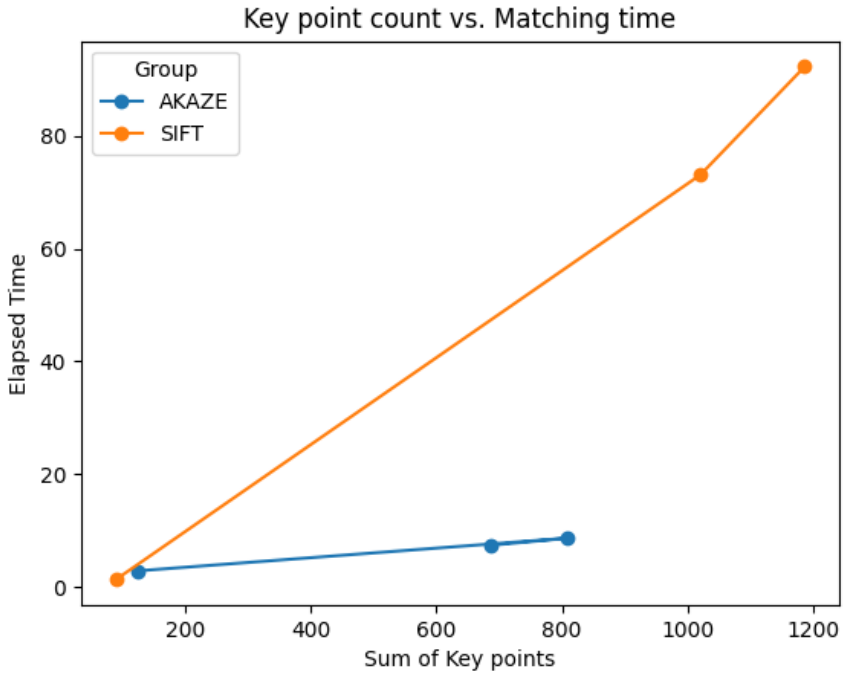


Chart of matches
100
80
60
40
20
0
SIFT s1
AKAZE s1
SIFT s2
AKAZE s2
SIFT s3
AKAZE s3




*Key point count vs. Matching Time*

As it is visible from the charts and table, AKAZE is more efficient among the two but it comes at a cost, the detected number of key points are smaller than the number of key points produced by SIFT. The more key points detected, the more time it takes for the methods to complete, and SIFT prefers key point count instead of speed, and as it is visible it performs better in high texture regions.




## 3. Task: Homography Estimation Experiments

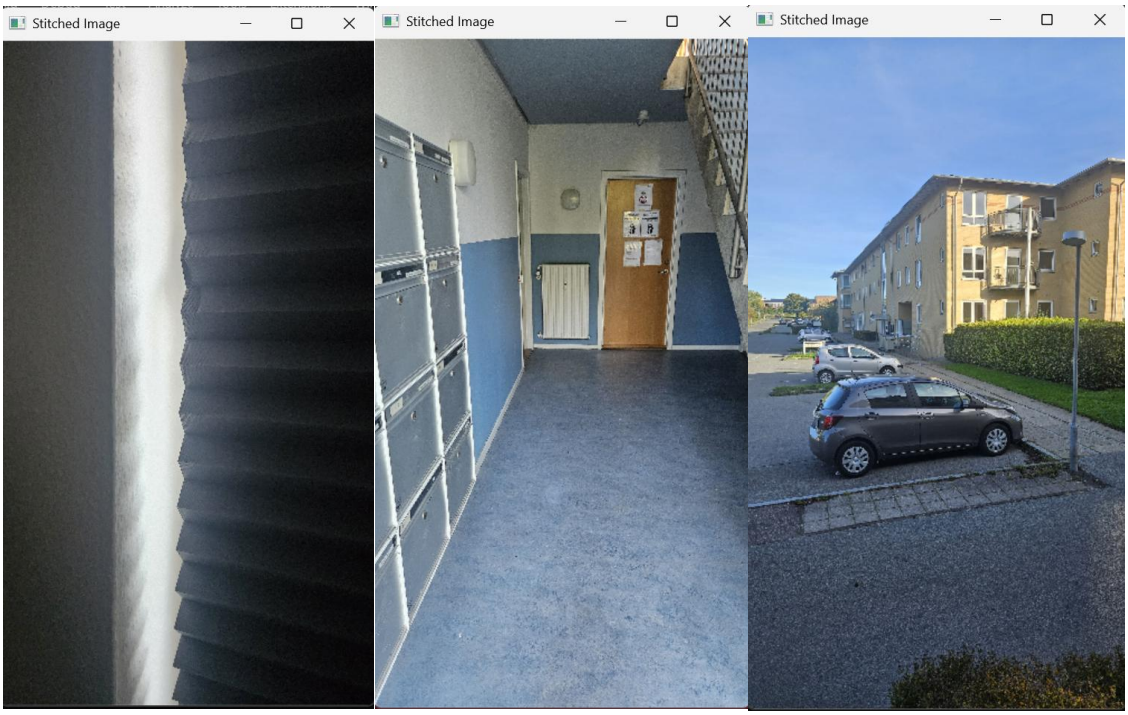
These pieces of information were also logged to the console in the C++ program: (The images were scaled to 15% of their original size on both axes for better runtime and better viewability of matches on my device).

The estimation used RANSAC, with 4 different thresholds. The alignment quality is measured through inliers/matches ratio. Through the experiments, the number of inliers, the runtime, the result quality (stitched images) and ratio were measured. For the experiments, SIFT method was used, as it produced higher key points and higher matches after knnMatch.

| | Threshold = 1 | Threshold = 3 | Threshold = 5 | Threshold = 10 |
|---|---|---|---|---|
| s1 Inliers | 5 | 6 | 6 | 7 |
| s1 Visual Quality | Poor | Poor | Good | Moderately Good |
| s1 Runtime | 26.7742 ms | 20.6113 ms | 11.0417 ms | 6.4134 ms |
| s1 Ratio | 0.357143 | 0.428571 | 0.428571 | 0.5 |
| s2 Inliers | 22 | 42 | 54 | 57 |
| s2 Visual Quality | Poor | Poor | Good | Moderately Good |
| s2 Runtime | 93.3337 ms | 9.337 ms | 3.0928 ms | 2.8697 ms |
| s2 Ratio | 0.252874 | 0.482759 | 0.62069 | 0.655172 |
| s3 Inliers | 27 | 48 | 64 | 65 |
| s3 Visual Quality | Poor | Good | Good | Moderately Good |
| s3 Runtime | 39.2874 ms | 4.7665 ms | 1.7911 ms | 1.324 ms |
| s3 Ratio | 0.355263 | 0.631579 | 0.842105 | 0.855263 |

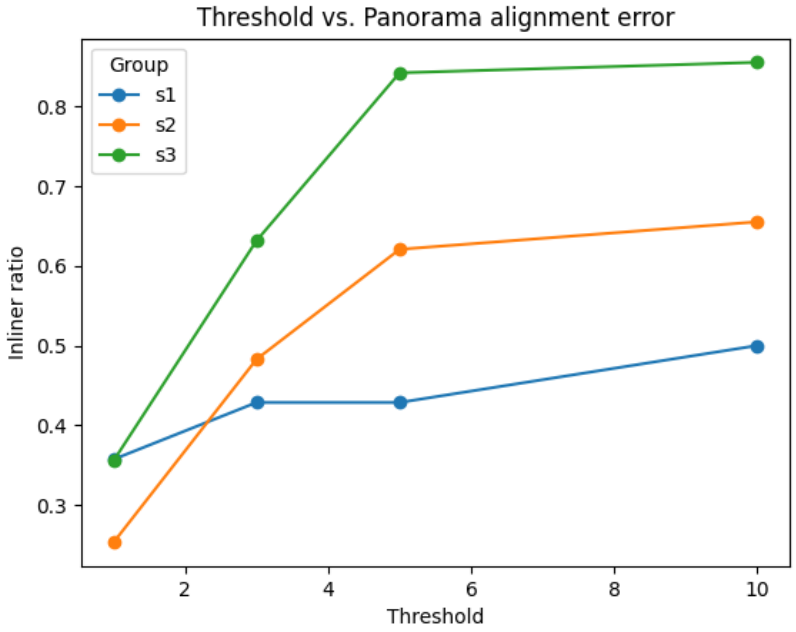
### *The threshold t=10 images*

For every dataset, after stitching, in respective order: s1, s2, s3.




### *Threshold vs. Panorama alignment error*

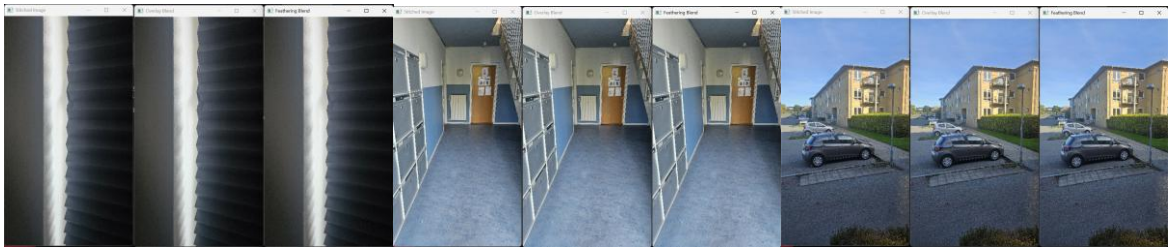
In this part the error is measured by the inlier ratio.




As the threshold is getting bigger, more and more matches will be counted as inliers that's why the ratio increases, this is beneficial up to a point, but if too big distances are allowed between pixels, it can lead to alignment error and poorer visual quality.

## 4. Task: Blending the images

For the blending the Homography result with threshold t=5.0 were used as the produced acceptable results, so t=5.0 is considered a sweet spot. The order of the images is: s1, s2, s3 and in each picture the following is true: stitched image, Overlay blend, Feathering blend



### *Overlay blend*

Sharpness may be kept on a good level, but sometimes fail on lighting adjustments.

### *Feathering blend*

Resulted in a smoother blending with fewer visible seams, especially in areas such as the sky.

### *Comparison*

Overlay blending may leave some seams where lighting is adjusted, however Feathering blend created better transition between the images. This measure is subjective but it may play an important role.

## Discussion

The best quality images were created when SIFT was used as key point detector, as it could identify a vast number of key points (trade off to being computationally expensive a significantly slower). The

threshold t=5.0 created nicely stitched images as it filtered out just as many points to have a good stitch alignment and using the Feathering blend on it resulted in smoot transition between the images.

## Sources

The link to the GitHub repository: GitHub Repository, this contains the short video and the source images as well.