COMPSCI 773
Intelligent Vision System

Image Stitching
Milestone 2 Report

April 6, 2022

Author: Neville Loh, Nicholas Berg

1 Summary

This report summarises our work on continuing to develop a panorama image stitching application. This "Panorama Application" problem can be summarised as combining several discreet images into a single image with a larger field of view. The complexity of the problem is due to the large amount of possible variation between the images (changes in lighting, translations and rotations in space, and a general human tendency to notice discontinuities). Having explored three methods of feature detection (specifically corner detection) in the first phase, this second phase prepares for the combination of these images by matching features between images. For a feature in one image (e.g. the left image), we aim to produce an efficient and accurate process for matching it to a corresponding

In this phase, we continue using python to implement known algorithms for the matching of features, combined with some experimental optimisations to improve the quality of the mapping. Our primary task was to utilise the corner features identified in the first phase to find matching pairs representing the same feature in two distinct images. To achieve this, we used Normalized Cross Correlation or NCC, which is discussed below.

After completing the NCC implementation, we extended our work by exploring four areas of interest:

Extension 1: Optimisation of the NCC Matching We performed a number of performance optimisations to decrease run-time of our NCC. Additionally, using the intuition that our panoramas would be unlikely to experience drastic rotation around the z axis, we made the assumption that the length and gradient of the lines connecting features should be tightly distributed. We used statistical analysis to design a filtering step which significantly improves signal to noise (reducing false positive matches without reducing true positive matches). Additionally, we combined our low level implementation of NCC with an existing implementation of NCC based on Solem (2012)'s Computer Vision textbook. Finally, we performed "sanity checks" by testing our system on three different pairs of images, and checking the extent to which unrelated images would be mapped, or similarly the extent to which two identical images are mapped. Lines have been plotted in distinct colour to better visually identify true and false positive matches. The filtered output of our normalized cross correlation descriptor matching, lines are parallel as would be intuitively expected for a pair of images suitable for a landscape panorama.

Extension 2: Comparison with HOG feature detector. As our second extension, we researched implementations of Histogram of Oriented gradients, and combined and modified existing implementations of the HOG feature detector to fit into our workflow, allowing comparison with our existing image sets. The filtered output of our normalized cross correlation descriptor matching, lines are parallel as would be intuitively expected for a pair of images suitable for a landscape panorama.

Extension 3: Comparison with SIFT feature detector. As our third extension, we utilised the VLFeat open source library to more easily implement a SIFT feature detector. This implementation was used to perform the same tests as our two NCC implementations and HOG implementation, for a comparison of different feature detectors across a common image set. The iltered output of our normalized cross correlation descriptor matching, lines are parrallel as would be intuitively expected for a pair of images suitable for a landscape panorama.

Extension 4: Parameter exploration and optimisation for different subjects. Continuing on from Phase 1, we had a number of python packages that automated the application of our feature detectors and matching algorithms, and that would produce simple statistical analyses. It is interesting to note that only after two iterations of a relatively small number of different implementations, the number of combinations of parameters and methods already yields tens of thousands of possible experimental conditions. Given the focus of this assignment was on NCC, we chose to focus our optimisation on NCC parameters, and considered six thresholds (0.999, 0.99, 0.9, 0.5, 0.1, 0.01) and seven windows radii (3, 5, 7, 9, 11, 15, 21) for two different implementations of NCC (Berg-Loh and Solem) over three pairs of images (Tongariro, Oxford, Playground) for a large but manageable collection of 252 experimental conditions. Notably, not all combinations yielded matched pairs, helpfully shrinking our data set. Additionally we performed "reasonability tests" along the way, including running the same image for both left and right channels, or unrelated images. Generally, under reasonable values, this produced the desired result of a large and tight distribution (for mapping an image to itself) and no matched pairs (for the unrelated images).

2 Introduction

In our phase 1 report we briefly discussed the context of the project,

3 Overview of the corner correlation

For the basic implementation, the normalized cross correlation is used to computing the corner correlation. . The implementation is written in the module feature_descriotor and the file feature_descriptor.py.

4 Normalised Cross Correlation (NCC)

The feature descriptor represent the uniqueness of region around the corner. By selecting the most similar descriptor, similar corner from two different image can be mapped together.

Listing 1: Corner Correlation Calculation

```
def match_corner_by_ncc(image_data_1: Tuple[ImageArray, List[Type[Corner]]],
                        image_data_2: Tuple[ImageArray, List[Type[Corner]]],
                        feature_descriptor_patch_size: Optional[int] = 15,
                        threshold: Optional[float] = 0.85) -> \
        List[Pair]:
    """
    Match the feature descriptors of the corners.
    """
    left_px_array, left_corners = image_data_1
    right_px_array, right_corners = image_data_2

    left_corners = compute_feature_descriptor(left_corners, left_px_array, feature_descriptor_patch_size)
    right_corners = compute_feature_descriptor(right_corners, right_px_array, feature_descriptor_patch_size)
    pairs = compare_all_ncc(left_corners, right_corners, threshold)

    return pairs
```

The function first calculates the image descriptor surrounding the corner. The image descriptor are saved in the data structure Corner as a n by m numpy array, where n and m are the dimension of the input image. These pair are only included if there exists a at least windows size by windows size pixel around the corner. If the corner is located in the border region, it will be ignored. The function then compare the correlation of each corner, these are done by calculating the NCC of the 2 corner for every corner in the output of the previous step. The best correlated result are than outputted as a new data structure called Pair.

```
class Pair:
    """
    Pair of Corner objects.
    """
    corner1: Type[Corner]
    corner2: Type[Corner]
```

```
    ncc: float
    gradient: float
    distance: float
```

4.1 Computing the NCC

The normalized cross correlation is computed by taking the intensity of the image array and normalized by their standard deviation;

$$NCC(f,g) = \frac{\sum_i (f_i - \bar{f})(g_i - \bar{g})}{\sqrt{\sum_i (f_i - \bar{f})^2}\sqrt{\sum_i (g_i - \bar{g})^2}},$$

where f and g are the feature descriptor vector of the corner in left and right image. The computation are executed both in the feature descriptor generation steps and in the compression steps.

4.2 Preprocessing

By rearranging the equation, we get

$$\begin{aligned} NCC(f,g) &= \frac{\sum_i (f_i - \bar{f})(g_i - \bar{g})}{\sqrt{\sum_i (f_i - \bar{f})^2}\sqrt{\sum_i (g_i - \bar{g})^2}} \\ &= \sum_i \left(\frac{(f_i - \bar{f})}{\sqrt{\sum_i (f_i - \bar{f})^2}} \right) \left(\frac{(g_i - \bar{g})}{\sqrt{\sum_i (g_i - \bar{g})^2}} \right), \end{aligned}$$

this indicated that the standard deviation steps and the normalization steps can be pre-computed in the feature descriptor generation stage.

Listing 2: compute_feature_descriptor

```
def compute_feature_descriptor(corners: List[Type[Corner]], img: np.ndarray, patch_size: int) -> \
        List[Type[Corner]]:
    """
    Get the patches from the image
    The patch is the region of interest around the corner, which is used for the feature descriptor.
    The patch is a square of size patch_size x patch_size. If a contour is too close to the border,
    the corner is not considered.
    -------
    """
    ...
    result_corners = []
    height, width = img.shape
    for c in corners:
        # ignore border
        if not (...border):
            ...

            # pre-compute the standard deviation and normalize the patch
            patch = (patch - np.mean(patch))
            std = (np.sqrt(np.sum(patch ** 2)))
```

```
            # set feature descriptor and handle zero division error
            c.feature_descriptor = patch / std if std != 0 else patch / 1e-10
            result_corners.append(c)

    return result_corners
```

If the standard deviation is 0, then a very small number is return instead of the standard deviation. By pre-computing the heavy operations, this avoid computation being execute more than 1 time during the $O(n^2)$ comparison. The overall performance is increased by roughly 3 times.

4.3 Selecting the best matches

The best matches are calculating by calculating the ncc for each corner and every other corner in the other image.

Listing 3: compute_feature_descriptor

```
def compare_all_ncc(corners1, corners2, threshold):
    """
    compare the two list of corners, and return the best matches.
    O(n^2) complexity. Brute force implementation.
    """
    pairs = []
    for c1 in corners1:
        # initialize the best match
        first_corner = corners2[0]
        ncc = compute_ncc(c1, first_corner)
        best = (first_corner, ncc)
        best2 = (first_corner, ncc)
        for c2 in corners2[1:]:
            result = compute_ncc(c1, c2)

            # check if result greater than 2nd best, if yes, replace 2nd best
            if result > best[1]:
                best2 = best
                best = (c2, result)
            # compare the with the second best value
            elif result > best2[1]:
                best2 = (c2, result)

        # check ratio between 2nd best match and best
        ratio = best2[1] / best[1]
        if ratio <= threshold:
            pairs.append(Pair(c1, best[0], best[1]))

    return pairs
```

Listing 4: compute_feature_descriptor

```
def compute_ncc(c1: Type[Corner], c2: Type[Corner]) -> float:
    # compute the normalised cross correlation
    return float(np.sum(c1.feature_descriptor * c2.feature_descriptor))
```

The first and the second best match are retrieved by looping through each corner, and only appended to the result set if the ratio between them is less than or equal to the threshold.

5 Results

The follow section provides the results from our investigation:

5.1 Main Task: Implementation of NCC

The first task of this phase of development was to prove the functionality of our previously detected Harris corners for image stiching by matching them as features. NCC was used (discusssed above) and while there were some correctly identified features, there was also a large amount of false positives. Additionally, with the large number of edges present in the image with our initial threshold (0.9) and window (radius 7, a 15x15 pixel window) the image was difficult to qualitatively examine. Carrying on our investigations from the first phase of the assignment, we used statistical tools (boxplot, histogram) to investigate the images. For all three target pairs of images, the relative distance between corresponding features was reasonably consistent in both direction and magnitude. Therefore we proceeded to develop outlier exclusion methods. Additionally, we made a minor change to the code to enable each plotted line between features to be assigned a different colour. This makes individual feature pairs easier to identify in an otherwise cluttered image, and helped provide more insight into the refinement of our application.

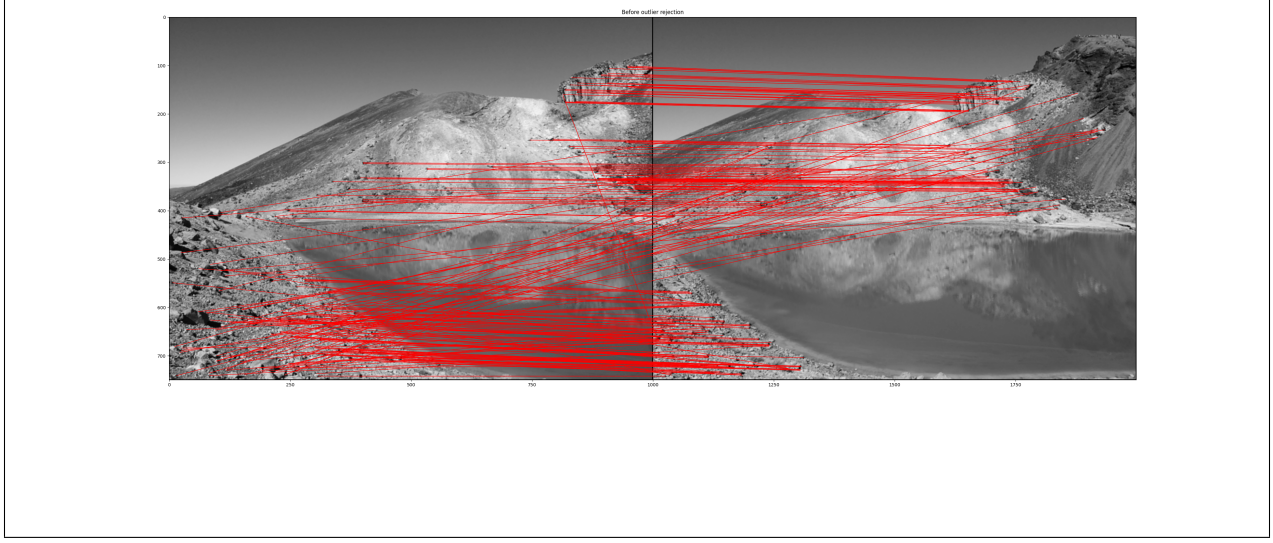

Figure 1: Two images of Mt Tongariro with the Harris corner features from two images mapped via Normalized Cross Correlation Algorithm

5.2 Extension 1: Optimisations of NCC

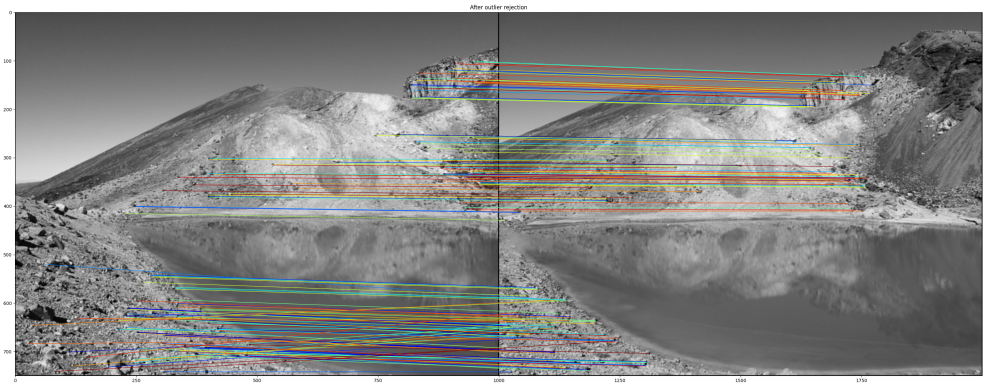

Figure 2: The feature mapping of the NCC features after filtering for Mt Tongariro. It was desirable at an early stage to isolate out real mappings from false positives, and a simple statistical method work effectively as an initial step.

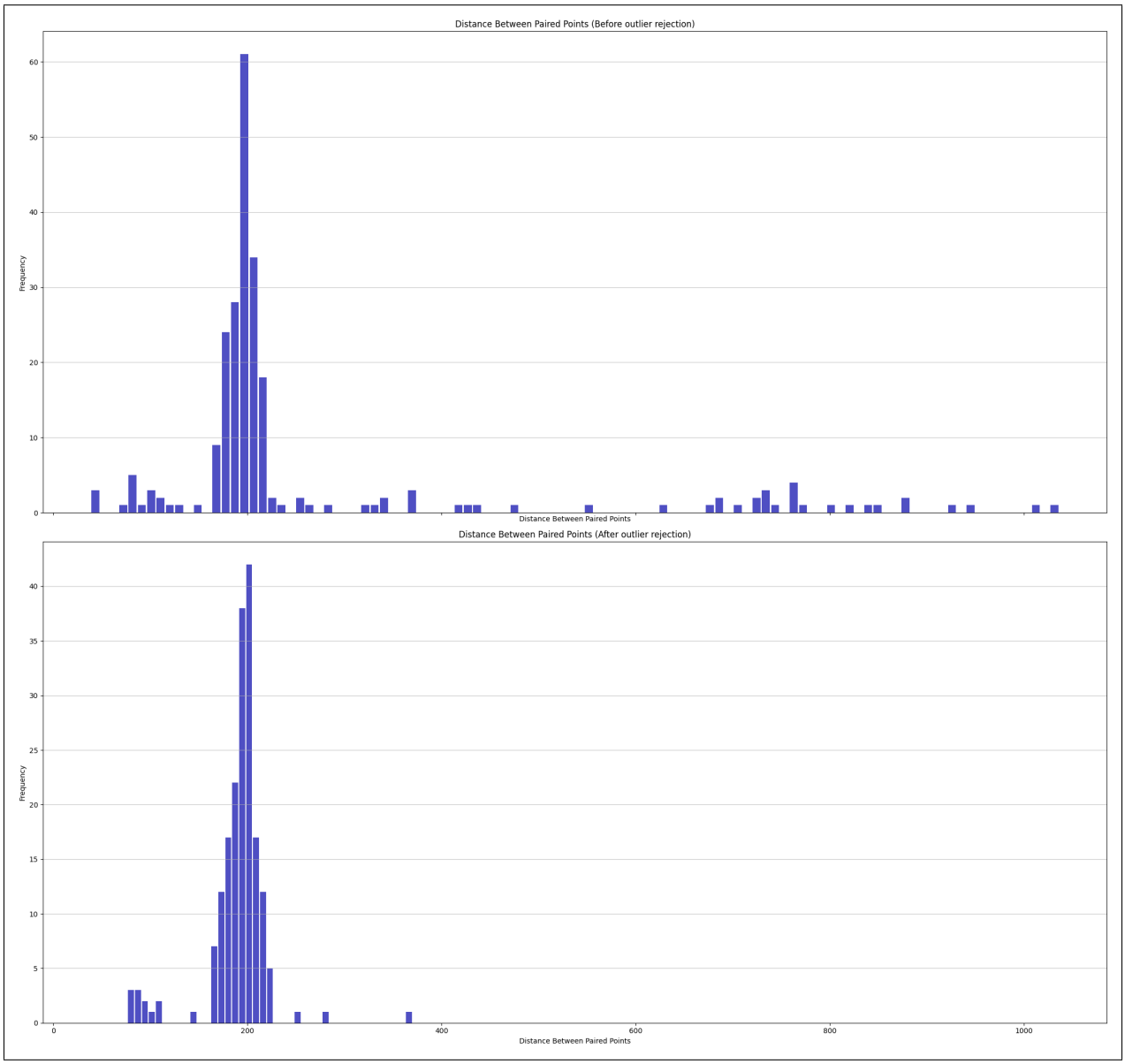

Figure 3: We found histograms like this figure, with the x axis locked between the two, revealing in highlighting changes to the feature pairings after a filtering step such as the one implemented on the images for Mt Tongariro above.

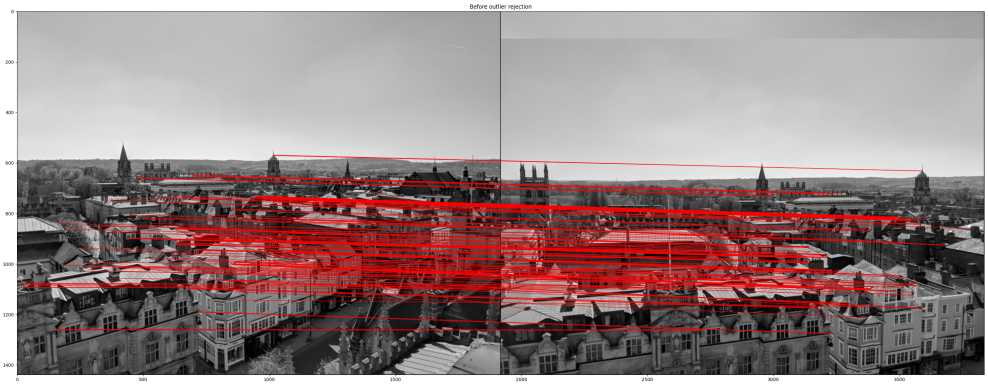

Figure 4: We felt it was important to not over-optimise a system for one image, and consistently worked with the three image sets we used in phase 1. These are the NCC features mapped before filtering for the Oxford image pair.

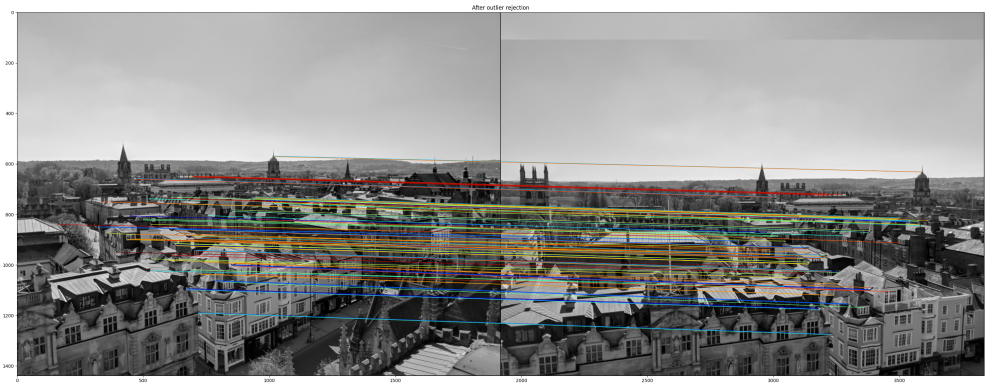

Figure 5: And the same two images after filtering.

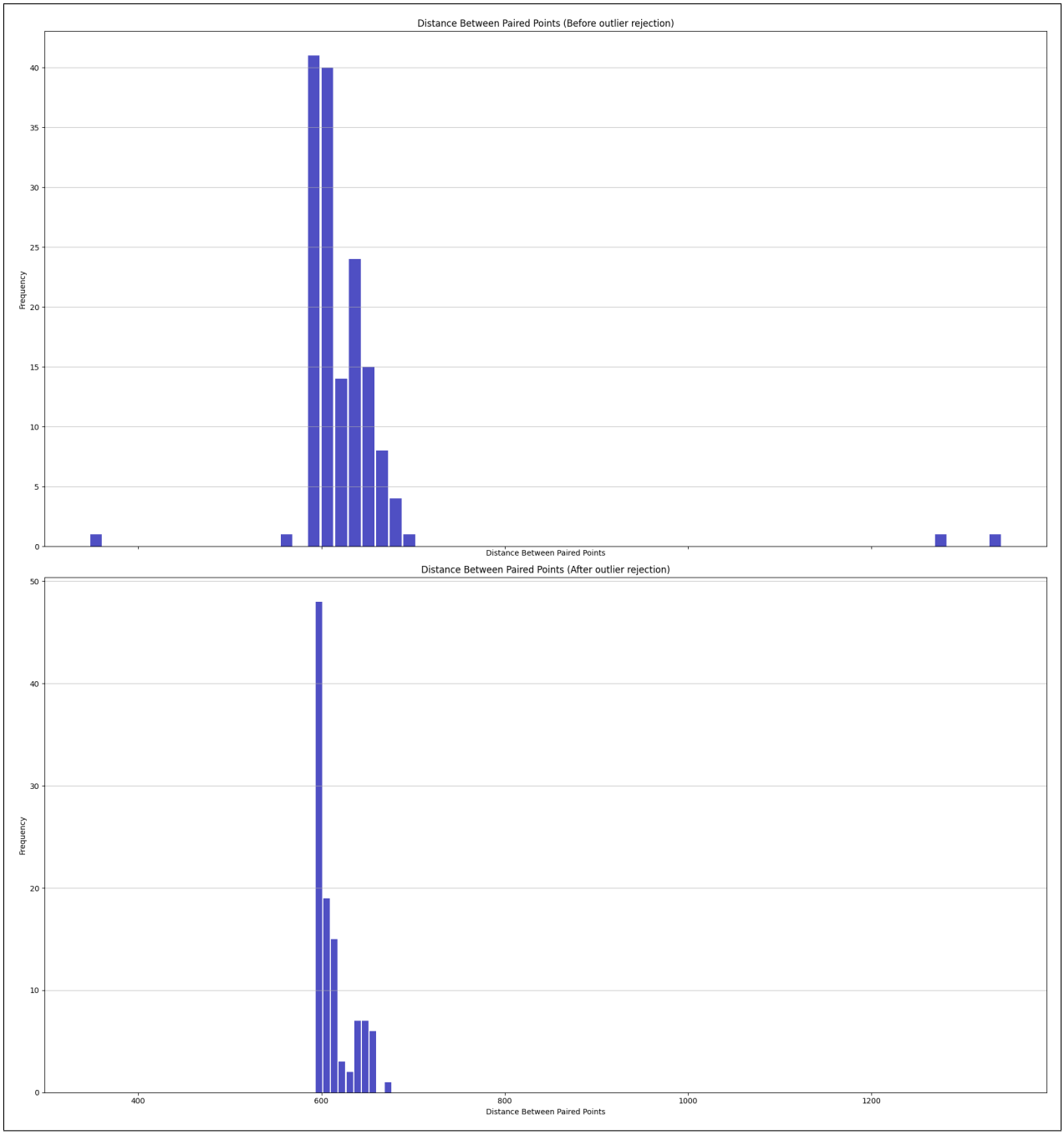

Figure 6: Two images of Mt Tongariro with the 1000 strongest Harris corner features after non-max suppression.

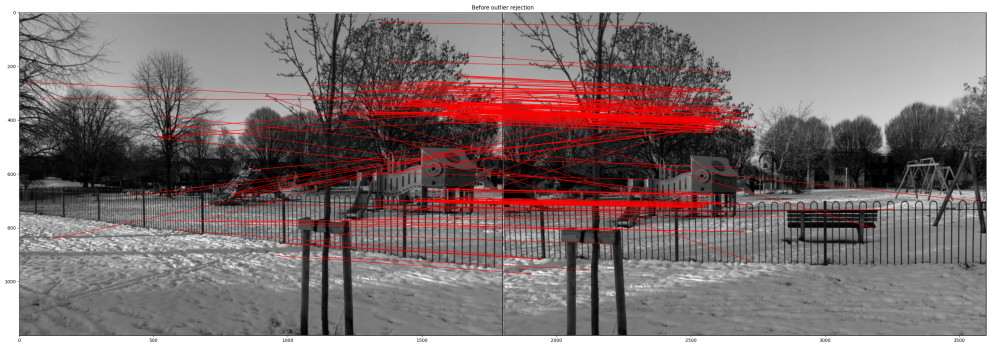

Figure 7: The feature mapping of the NCC features after filtering for the snowy playground image pair.

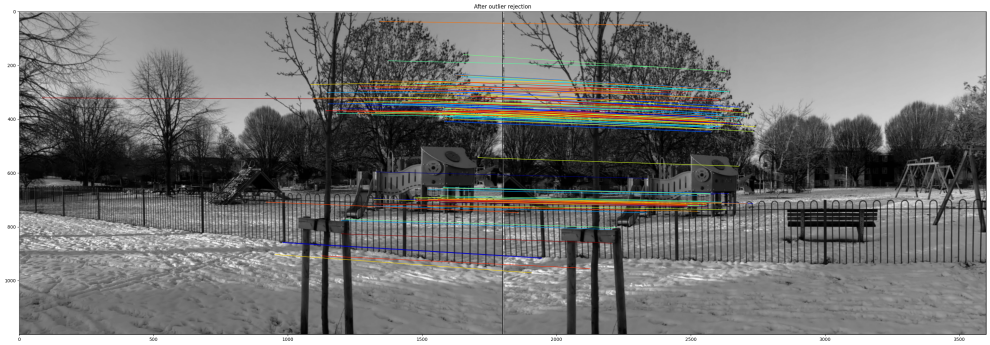

Figure 8: The feature mapping of the NCC features after filtering for the snowy playground image pair. Note the effects of rotation on the relative position of features in the image.

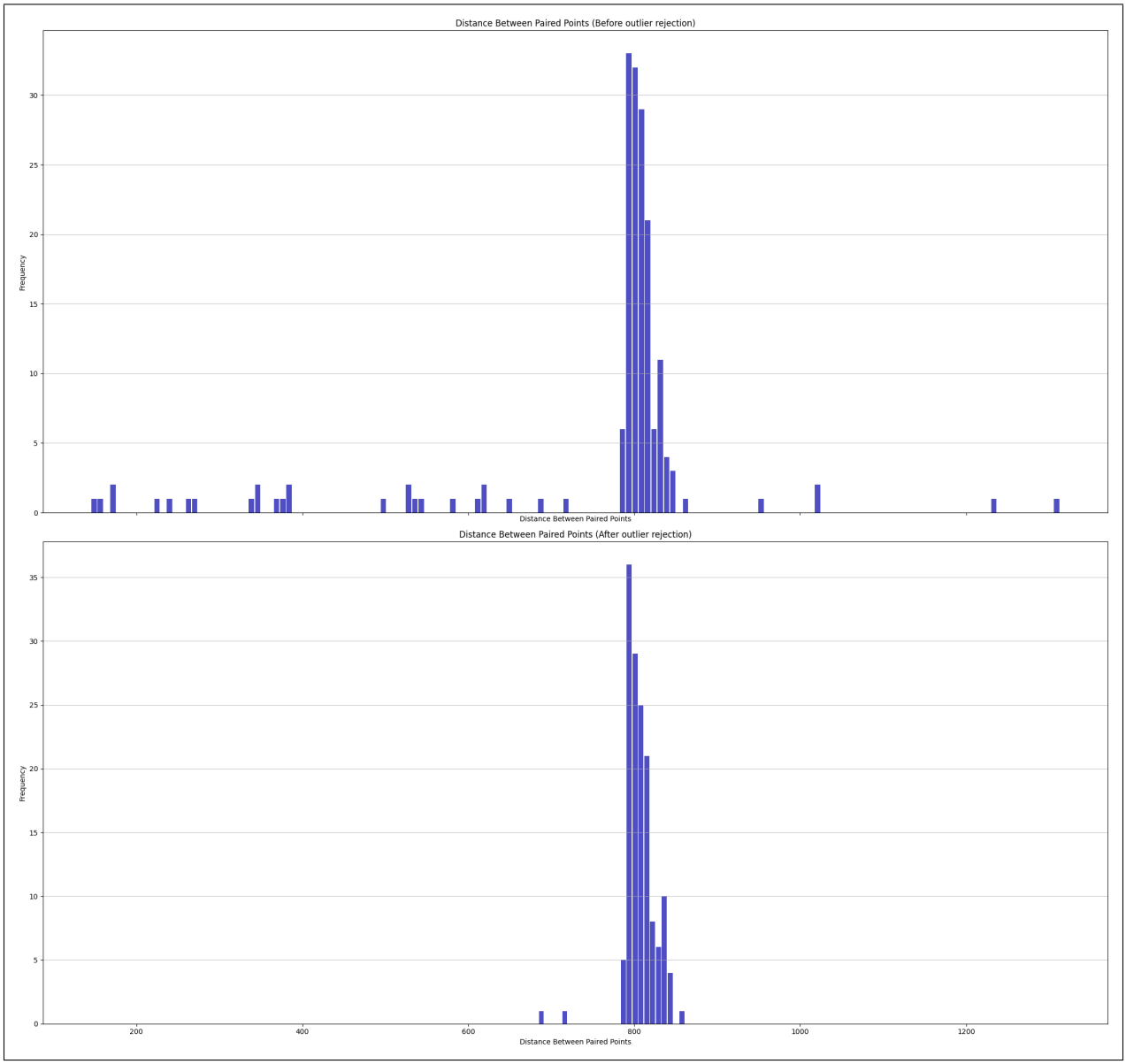

Figure 9: The histogram of the NCC features before and after filtering for the snowy playground image pair.

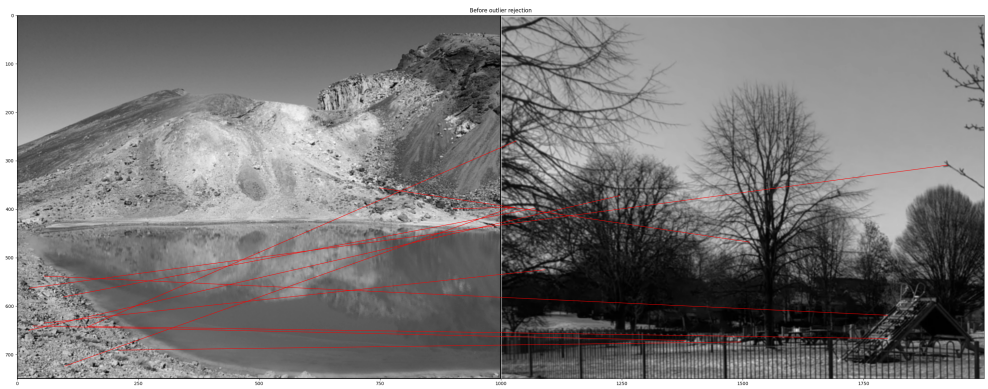

Figure 10: We attempted to regularly perform tests to ensure our program was behaving as expected. One such test is the attempt to map two unrelated images together. In this instance, the "noise" of small features (small rocks on Mt Tongariro and leaves on Playground equipment) happen to match. Later we looked at optimising window size and thresholds to minimise these false positives, but the most effective toll was the stochastic method described above.

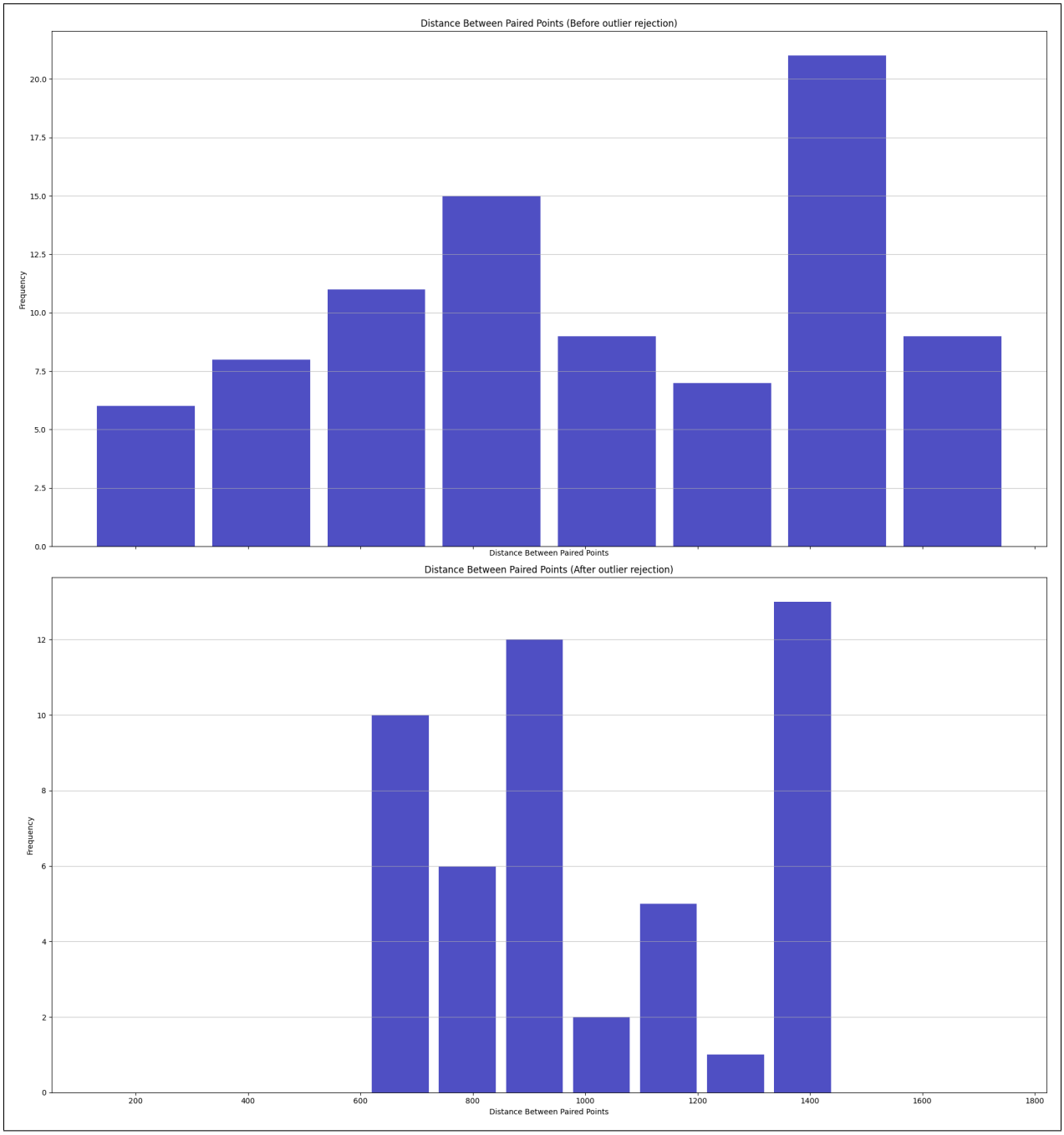
Distance Between Paired Points (Before outlier rejection)
Frequency
Distance Between Paired Points
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Distance Between Paired Points (After outlier rejection)
Frequency
Distance Between Paired Points
0
2
4
6
8
10
12
200
400
600
800
1000
1200
1400
1600
1800

Figure 11

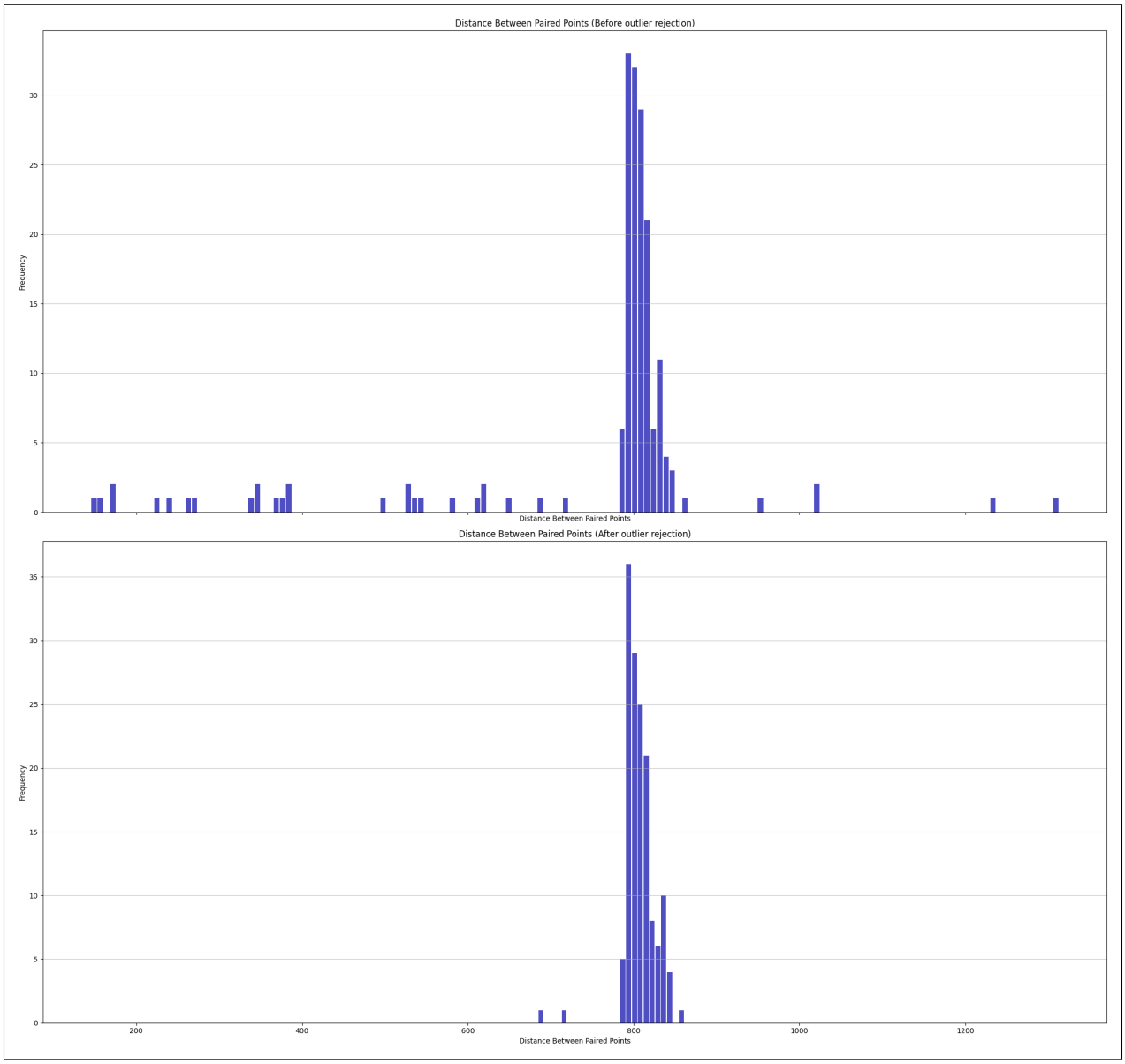

Figure 12: Histogram of the magnitude of vectors mapping matched feature descriptors. It is notable that even though we would expect these matches to be purely randomly distributed, because we have used the assumption that the images sit to the left and right of one another, our gradient and distance based stochastic methods are insufficient to filter out these false positives. Later work suggested a larger window and more appropriate threshold would minimise the occurrences of these false positives, which are generally rare, however in systems where users will not have control over the implementation, it is important to note unexpected behaviour such as this. It is a particularly good case for not allowing a merge of images to over-write the original image files.

5.3 Extension 2: Implementation of the Histogram of Gradients (HoG) descriptor.

5.3.1 Overview of HOG

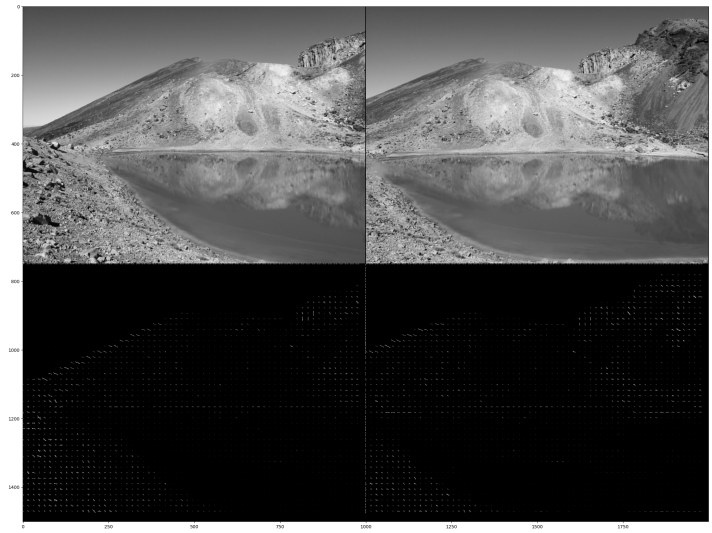

Figure 13: Two images of Mt Tongariro (above) and the computed HoG descriptors (below). Window size is 16x16 pixels, and the resulting oriented feature is displayed in white.

Figure 14: The Oxford image pair (above) and the computed HoG descriptors (below). Window size is 16x16 pixels, and the resulting oriented feature is displayed in white.

Figure 15: The snowy playground image pair (above) and the HoG descriptors below. Window size is 16x16 pixels, and the resulting oriented feature is displayed in white.

5.4 Extension 3: Implementation of the Scale Invariant Feature Transform (SIFT) descriptor.

5.4.1 Overview of SIFT

Scale Invariant Feature Transform.

Whilst simple template matching, such as NCC, is sufficient for relatively static panorama images (e.g. a mountain range, or a urban landscape) it may be less effective when confronted with repeating patterns or a shifting background (e.g. moving traffic in a panorama of a city street). Using a template image to match when the camera's placement or the scene has changed significantly might require extensive testing of rotations and scalar versions of the template, which would be computationally expensive. One solution widely adopted in industry is SIFT.

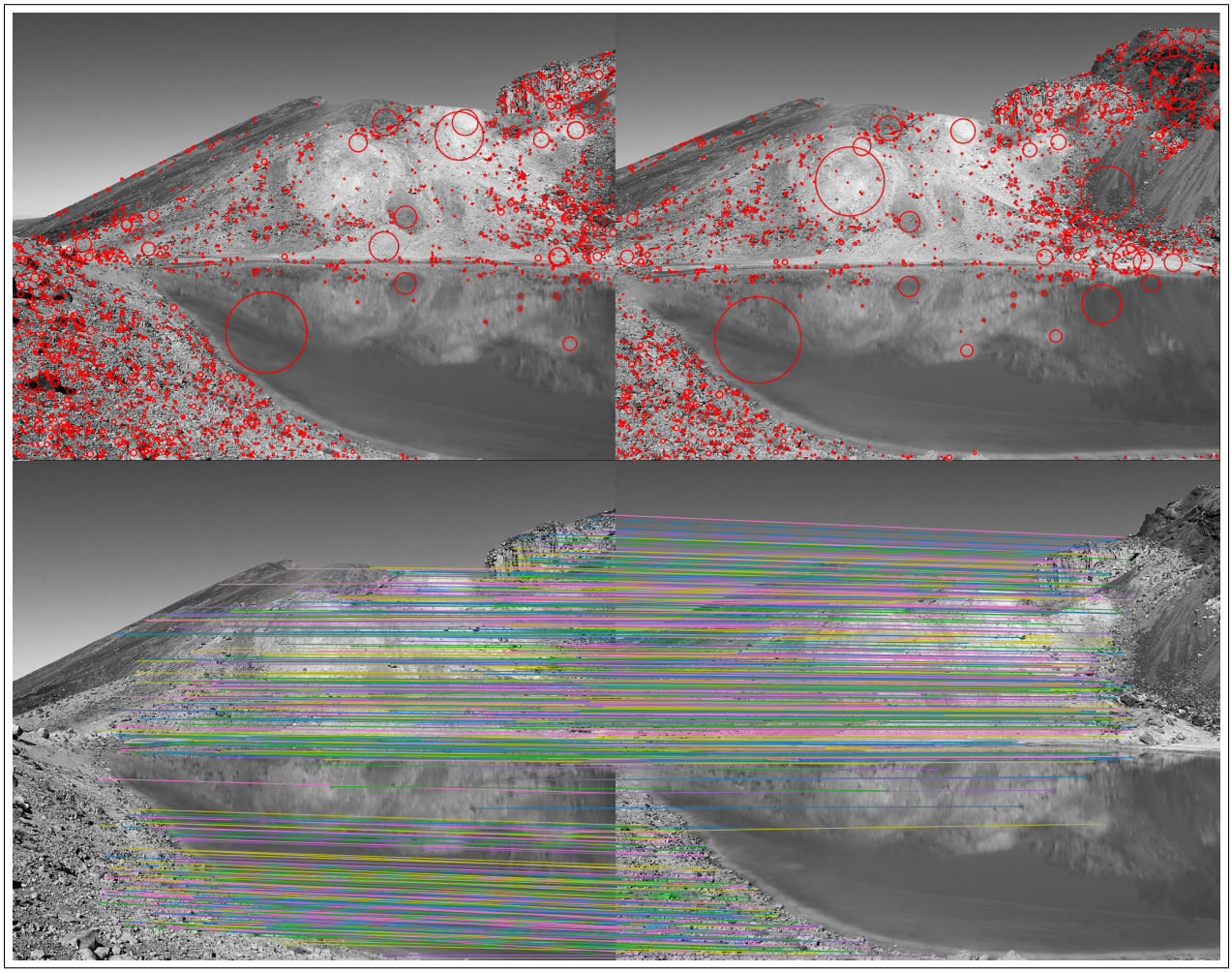

Figure 16: Two images of Mt Tongariro with the SIFT features and the corresponding mapping. SIFT significantly outperformed NCC and HoG in both performance (runtime) and quality of feature mapping. We used an open-source implementation of SWIFT from vlfeat for our comparisons. Whilst we would have enjoyed the challenge of fully implementing our own SIFT functionality, time constraints limited our ability to complete this work.

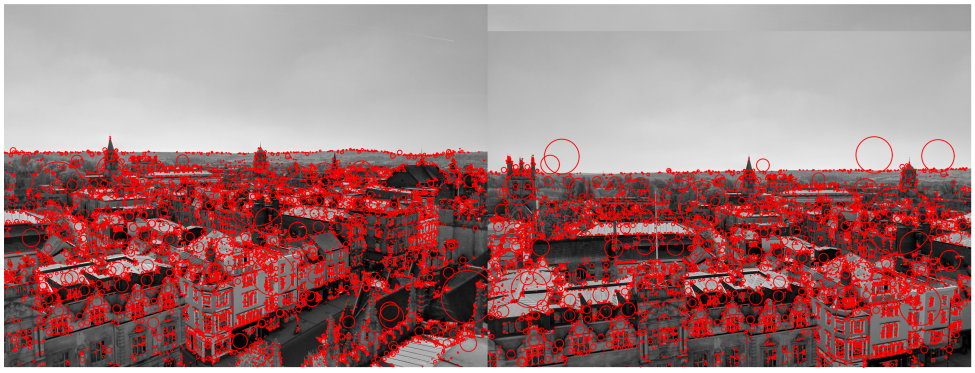

Figure 17: The Oxford image pair labeled with the open source SIFT implementation we adopted and modified slightly to enable comparison with our existing implementations.

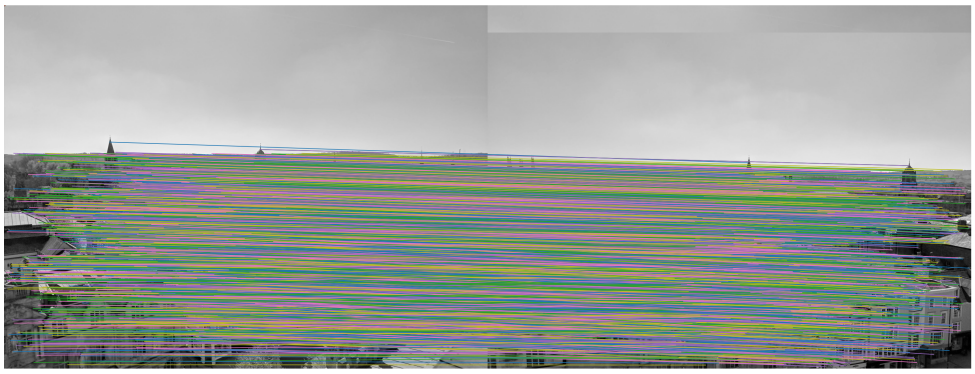

Figure 18: A very dense collection of feature pairs from the Oxford images, corresponding to the high density of features in Figure 17. Although SIFT offers significant benefits in robustness to rotation when compared with NCC and HoG, further investigation may be warranted if in images with a high occurance of repeated structures (such as the windows and chimneys in these images) SIFT creates more false positive pairings than NCC or HoG.

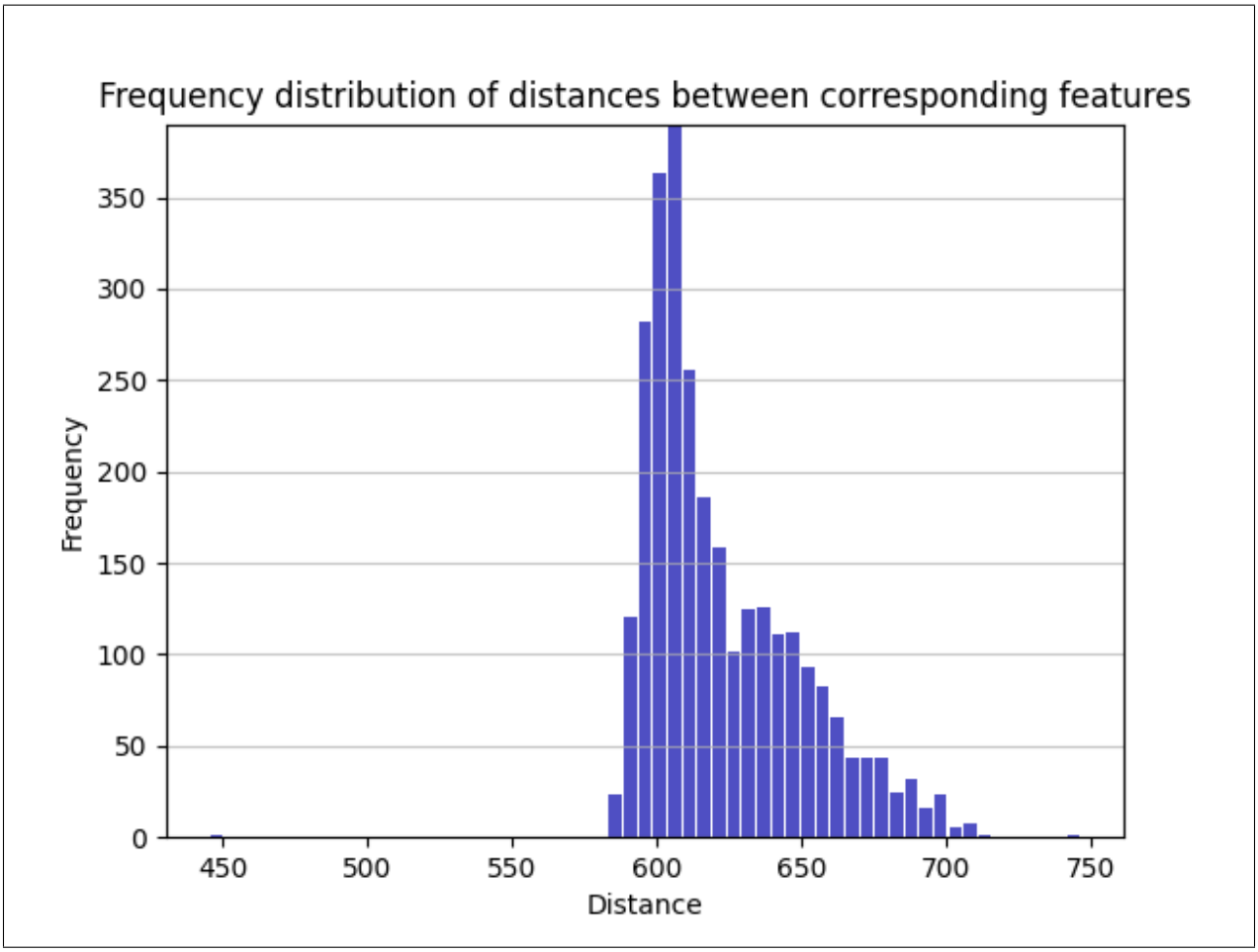

Figure 19: The Oxford Panorama image pair provides an interesting and subtle difference compare with Tongariro in that it features a higher degree of rotation, as well as more significant transformation of the relative position of the subject due to the larger ratio of distance between foreground and background. Given the generally high accuracy of the SIFT descriptor, the unusual distribution of distances in this figure may be a product of the geometric reality of the images rather than an artefact. Further research is required to confirm or refute this hypothesis.

Figure 20: Impressively, despite the very high density of SIFT features, SIFT performed extremely well on false positives when asked to map features between two unrelated images. Combined with the impressive speed of the implementation, it is empirically clear from our comparison why SIFT has been so successful in computer vision applications.

5.5 Extension 4: Exploration and Tuning of NCC Parameters.

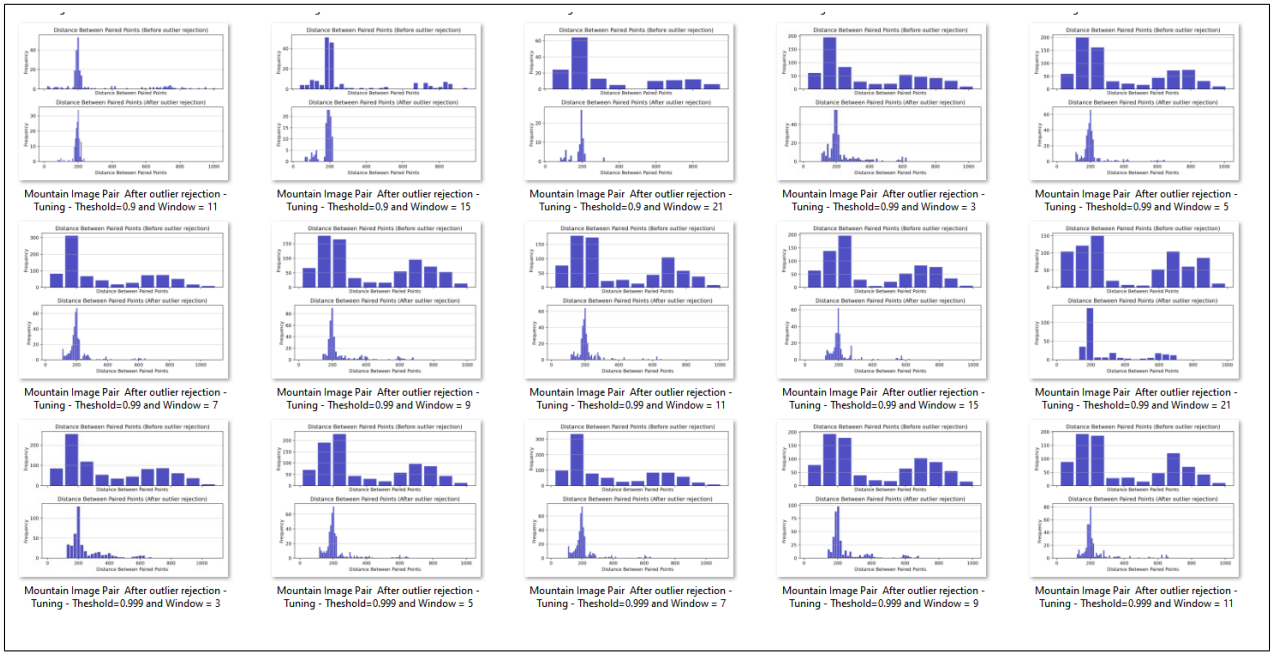

Figure 21: While the colour gradient over the mapping lines helps to identify more clearly individual feature pairs, it is difficult to make a quick qualitative assessment of the quality of the fit. Given our image is predominantly translated, a histogram of the distance between features is indicative of the signal to noise ratio, or false positive rate. Furthermore, using these statistical insights allows us to filter by gradient and length of line, which improved the quality of our mapping.

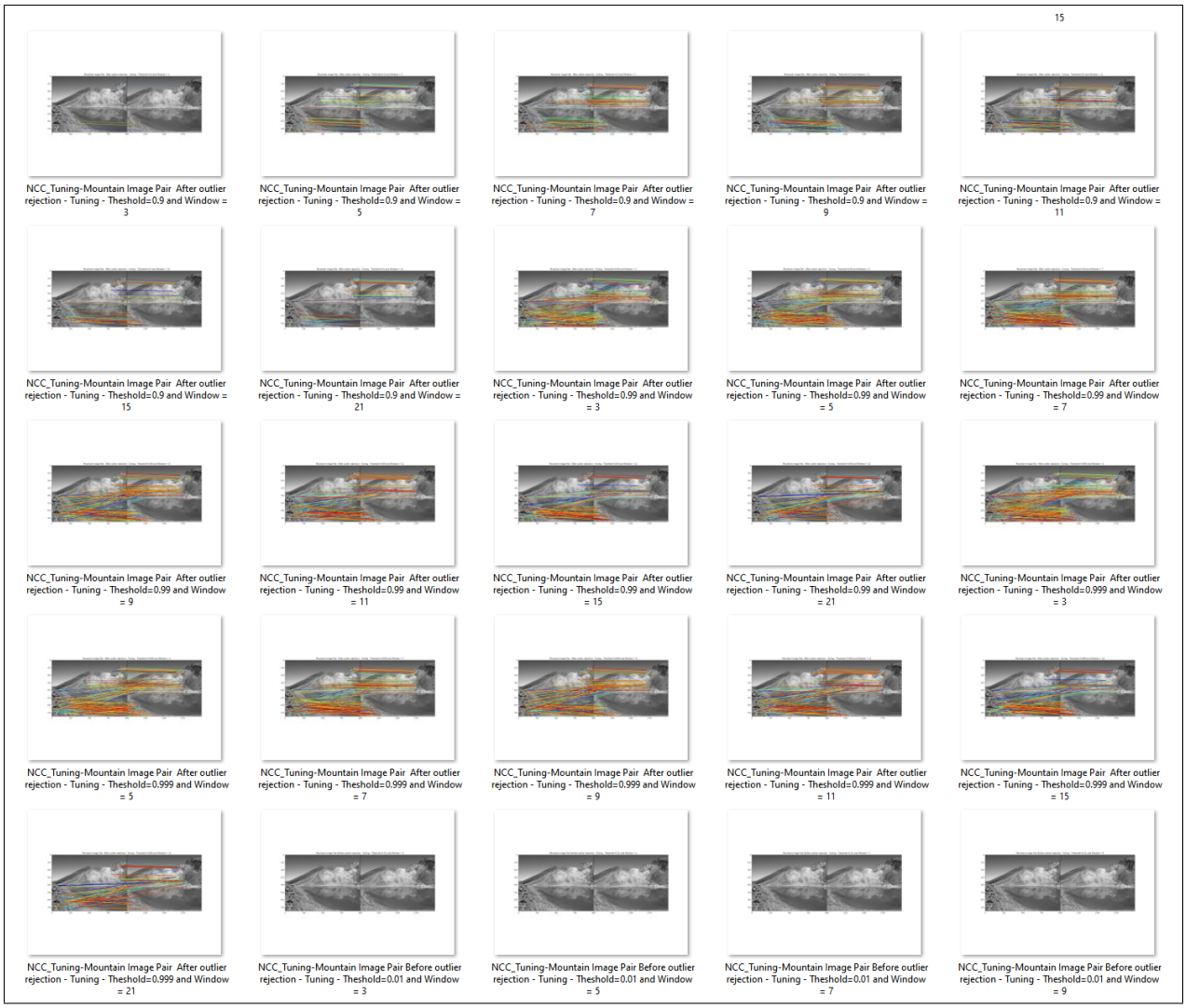

Figure 22: A set of NCC descriptor mappings, with different Thresholds and Window sizes used. A combined animated gif has been uploaded as part of the readme.md file on GitHub. Ideally we would develop good heuristics for an appropriate set of thresholds and window sizes, however the variation between window sizes primarily acts as a demonstration of the necessity and advantages of scale independent features (e.g. SIFT) for real world image handling applications.

6 Conclusion

In conclusion, we have conducted a large body of preliminary work into understanding image feature descriptors and applying them towards the combination of images into a single larger panorama, mimicking a larger field of view or higher resolution sensor. Whilst the relatively simple feature descriptor underlying NCC is not suitable for rotation,

In future work we aim to develop our work into a suitable image stitching application. Additionally, we would like to develop better heuristics and analysis tools to automate the tuning of parameters within the context of a known parameters (e.g. the resolution and field of view of a common smartphone camera). Within that context, there is scope for general heuristics (e.g. distant landscape vs. capturing a large group of people too close to fit in frame) where we believe different methods would be more applicable. Within the scope of this assignment we did not have the time available to go deeper into understanding the implementation of SIFT, and further work there would shed light on how we might optimise it for our applications. It was particularly notable the speed with which SIFT was able to process relatively large images, especially compared with our non-optimised NCC. Additionally, with the growing accessibility of machine learning resources, an interesting potential application is the predictive extrapolation beyond what was capture on sensor of these panoramic images, which is an exciting potential research topic.

References

[1] Dalal, N., & Triggs, B. (2005, June). Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)* (Vol. 1, pp. 886-893). IEEE.

[2] Y.M. Fouda and Abdul Raouf Khan, 2015. Normalize Cross Correlation Algorithm in Pattern Matching Based on 1-D Information Vector. *Trends in Applied Sciences Research*, 10: 195-206.

[3] Harris, C., & Stephens, M. (1988, August). A combined corner and edge detector. In Alvey vision conference (Vol. 15, No. 50, pp. 10-5244).

[4] Solem, J. E. (2012). *Programming Computer Vision with Python: Tools and algorithms for analyzing images.* " O'Reilly Media, Inc.".

[5] Jiang, Y., Ruan, L., Xiao, L., Liu, X., Yuan, F., Wang, H. (2018, April). THTM: A template matching algorithm based on HOG descriptor and two-stage matching. In *AIP Conference Proceedings* (Vol. 1955, No. 1, p. 040131). AIP Publishing LLC.

- End of document -