Phase 2: Feature Matching

NCC Feature Matching

The first step of this phase of the assignment was to implement NCC feature matching. The first implementation of the function iterated through all of the left corners, calculating each NCC as it went. It produced a reasonable tuple list of every left corner with its matching best-fit right corner. However, this took around 120 seconds to compute. An improved NCC matching function was created which calculated the NCC for each corner in both lists first, then matched them up with their best-fit right corner counterpart. This implementation takes about 11 seconds and produces the same output when the window size is 15. The output can be visualised in Figure 1.

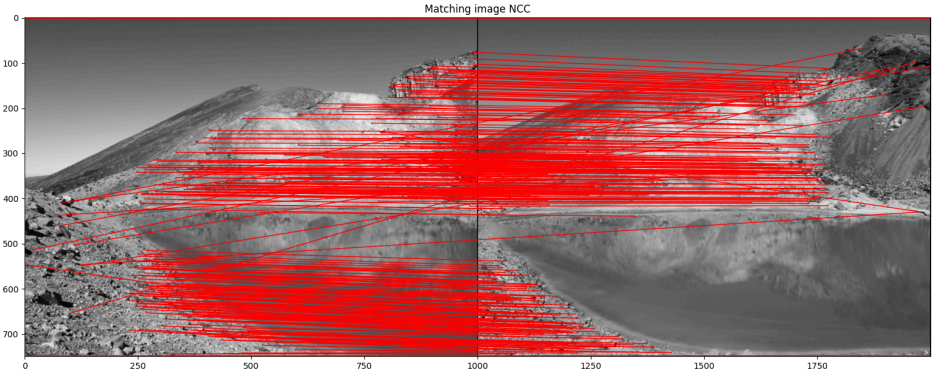

Figure 1: NCC feature matching. NCC window size = 15

This method of characterising the features using NCC appears to work reasonably well. This is based on the significant successful matching between the left and right images. However, it should be noted that there are a noticeable amount of outliers, especially at the edges where the features are extremely unique to each image. An improvement to the NCC function (or a separate function) in phase 3 may need to be made to address these unique edge-case features and their outlying matching. This method of feature matching appears to provide many positive matches to a future function that would look for inliers.

Changing the window size between 5, 15 and 25 appears to have an effect on the matches produced. When set to 5, only a small handful of points are output. This is likely because the characteristics of each point cannot be unique enough with such a small window. Since each feature cannot be unique, it does not pass the ncc ratio comparison threshold.
When set to 25, the number of erroneous matches goes down, but so does the number of correct matches, see Figure 2.

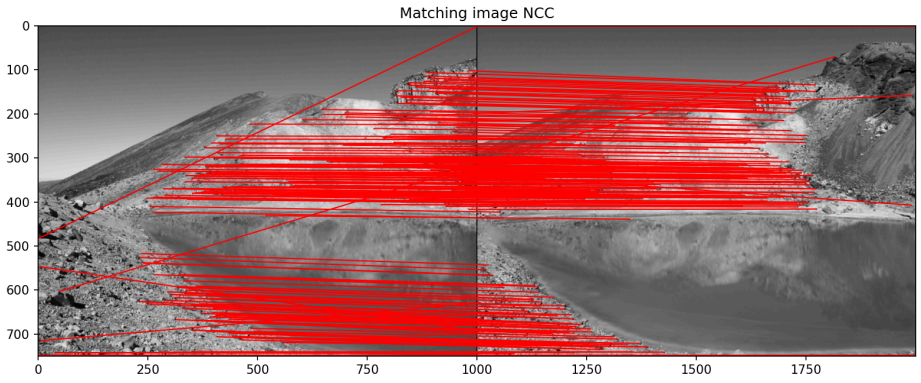

Figure 2: NCC feature matching. NCC window size = 25

This may be beneficial if incorrect matches cannot be removed via another method and/or incorrect matches cause significant problems at later stages. It should be kept in mind that if the number of matches gets too low, the window size may need to be adjusted so that some matches are made (if it has been determined that there definitively should be some). For the time being, the window size will be set to 15 as to maximise the number of positive matches.

DoG Feature matching

We implemented DoG scale invariant feature detection as an extension in phase 1 of this assignment. In addition to detecting feature points, this feature detection method also determines the scale of features. This allows us to calculate the radius of the circle encompassing the detected feature. We can utilise this radius during matching to make the feature-matching process scale invariant as well. This is important as Harris corners are not scale-invariant, meaning if the left or right image is zoomed in or out, it may not detect corners in the same place and therefore those points won't be able to be matched. Furthermore, even if it was able to detect both points, a fixed-sized window around those points could look quite different (i.e. it would include more of the actual feature and less of its surrounding area on the larger feature). Again, resulting in a situation where they may not get matched.

To implement DoG matching we use the same feature-matching method with NCC feature descriptors, except the window around the points is calculated in a slightly different way. Instead of taking a fixed window size, we determine the window size using the blob radius. We extract this window and then resize it to be 15x15 using open cv's resize function. This is needed as comparing patches with NCC requires them to be the same size. After some experimenting, we found the matching was not working with small-scale features either. This was likely due to the window size being too small. To deal with this, we took the window size to be the radius (rounded up) plus 2. This helps to make smaller windows slightly larger so that the feature descriptors include enough relevant local information to perform matching while not having much of an effect on larger windows. This can be enabled by setting the variable *DoG_matching* to True in the file CS773StitchingSkeleton.py.

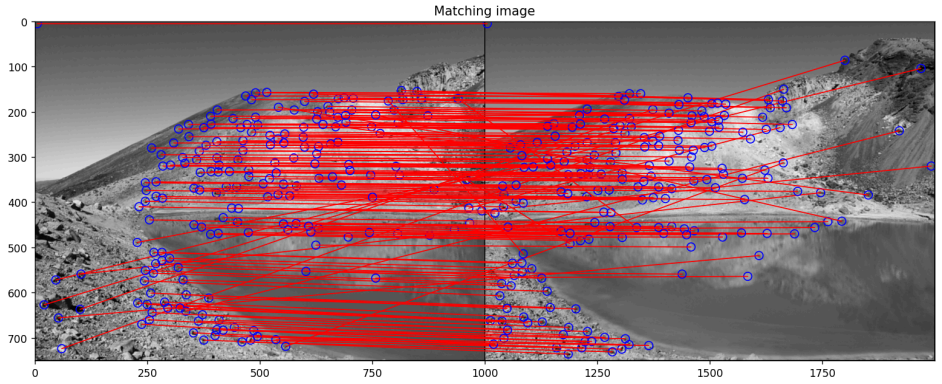

Figure 3: DoG feature matching on images with no scale change. Blue circles are the blobs of the matched features.

It is also important that the DoG feature matching method performs just as well as the Harris method on images with no scale. After testing with various images, we found that this method of feature matching performed very similarly to the original method when there was no scale change. This can be seen by comparing Figure 1 and Figure 3.

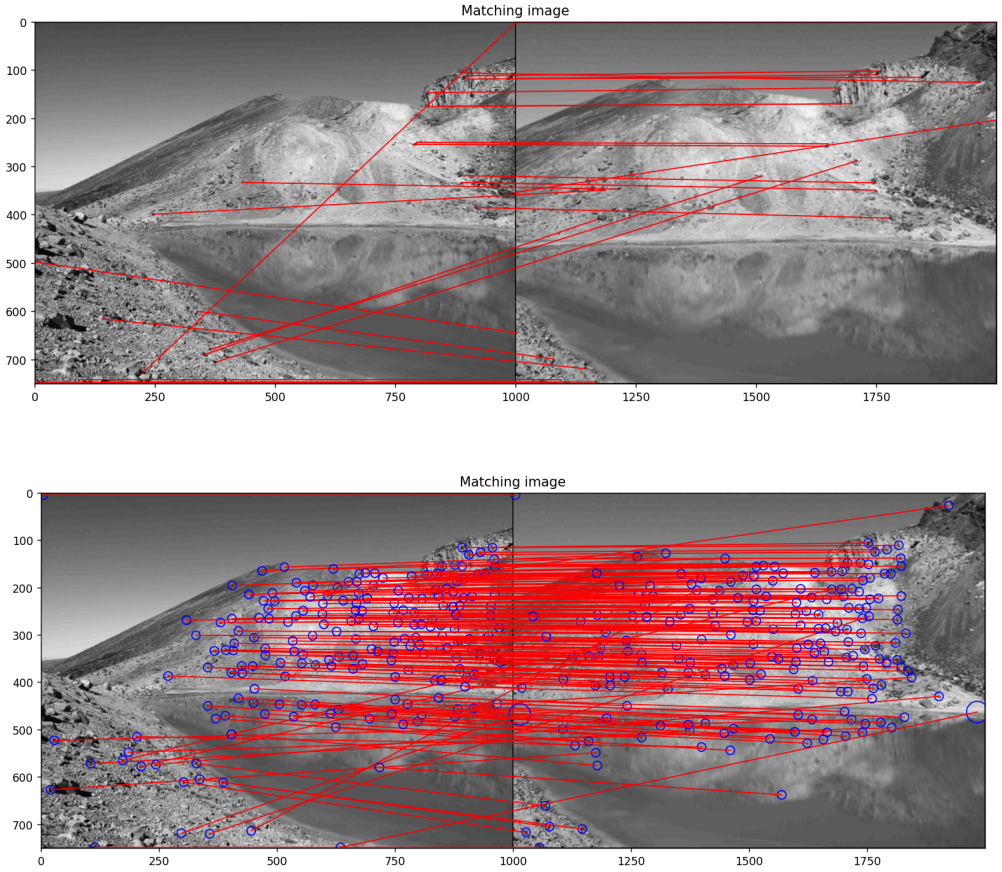

Figure 4: Comparing Harris feature matching (top) to DoG feature matching (bottom) with scale change (the right image is a zoomed-in version of the original)

In Figure 4, we can see that after even a small scale change, the Harris detector performs very poorly and finds very few matches. However, the DoG detector is still able to find many points and match them correctly.

HOG Feature matching

After implementing the NCC feature-matching algorithm, we wanted to explore other popular feature descriptors. We chose to implement the Histogram of oriented gradients (HOG) descriptor. We chose this algorithm since it is a very popular method within the field; often used in combination with the popular SIFT feature detector (Lindeberg, 2012). In hindsight however, this was not a good method to implement with Harris corners since these are not rotation invariant like SIFT features are. As a result, even a small rotation difference between two images can result in a huge difference in the HOG descriptor for the same corner in both images. We can see the ineffectiveness of this feature implementation in Figure 5, with there being no identifiably good matches between keypoints.

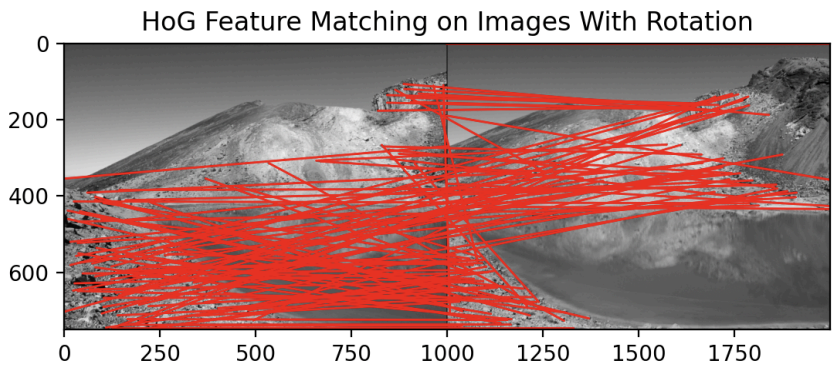

Figure 5: Ineffectiveness of HOG on images with rotational difference

To verify the validity of our assessment that HOG was simply not a good fit for these images, we decided to test if the approach worked as expected on two images that were only a translation away from each other. We cropped one of the Tongariro images such that each was 750 pixels wide, with 500px of overlap between the two images. We can see in Figure 6 that the HOG implementation on these images works very well, showing several easily identifiable feature matches, which confirms our assessment that HOG is simply not viable for these images without rotation invariant features.

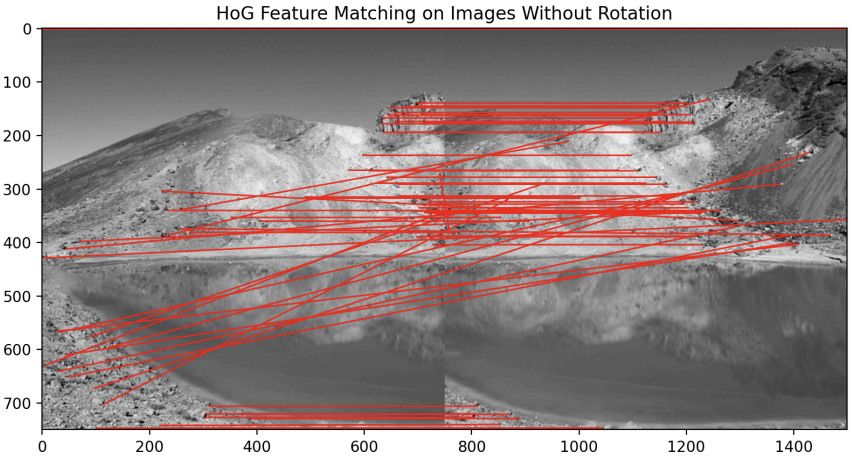

Figure 6: HOG results on images which only have a translation effect

Stephen:

- HoG matching

Zak:

- NCC matching
- Minor project refactoring

Kelham:

- DoG matching
- Testing and fixing feature matching

References

Lindeberg, T. (2012). Scale invariant feature transform. *Scholarpedia*, *7*(5), 10491. https://doi.org/10.4249/scholarpedia.10491