Lab 2

Task 1

The first part of the task is to implement a matlab (or python) function that will calculate the harris corners of an image. First, in order to explain corners, they need to be clearly understood. An corner is defined as where two edges meet (and an edge is a distinct line which there is a strong difference in coloring). The principle behind the harris corner detection is that the distribution of the derivatives in x and y direction differ between, flat, edge and corner regions. The flat regions will have a distribution close to origin (due to no large gradients in either x and y direction). Edge regions will have a "linear" distribution, meaning that large values of the derivative in the y direction, will also give large values in the derivatives of in the x-direction. With these "known" distributions, the maximum and minimum monents of inertia is calculated. With these, it can simply be put that the distributions of an edge will have a large maximum moment of inertia, and a small minimum moment of inertia (L1 >> L2), while corners will have somewhat similar values (both L1 and L2 is large).

When applying this, the structure tensor (or the second moment matrix) is used. After, the function I have implemented is using a gaussian filter to filter out noise (which often increase performance). After this, the determinant and the trace of the structure tensor is calculated. With the trace and determinant, the following equation is used to compute the harris response:

$$R=det(M)-k\cdot(trace(M))^2$$

With the function above, and a threshold value, the Harris corner response function can detect a corner.

After the corner detection, a non maximum suppression is used to only account for the strongest local corner detected.

Below is a picture of the implemented harris corner detection:

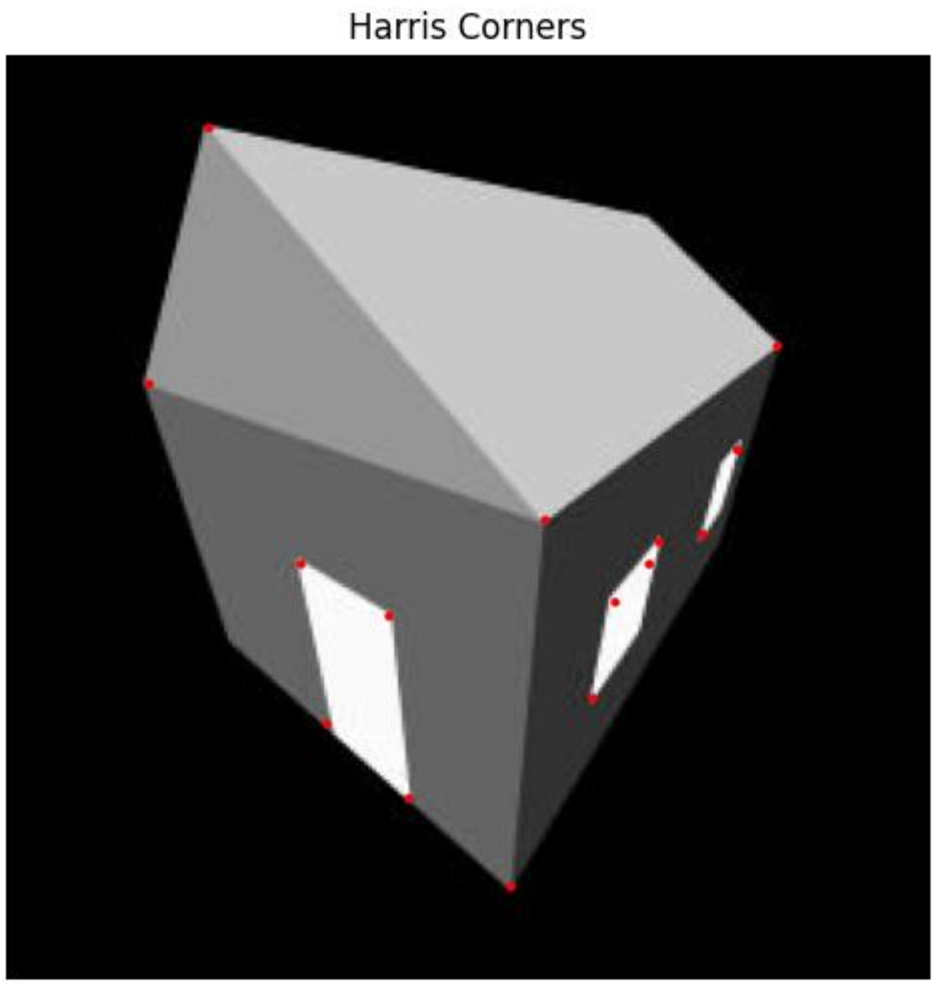

Figure 1 Harris corners of the python function.

And for sub task 2, it was asked to rotate the image by 45 degrees, and then apply the same detection. Figure 2 shows the result of the rotation.

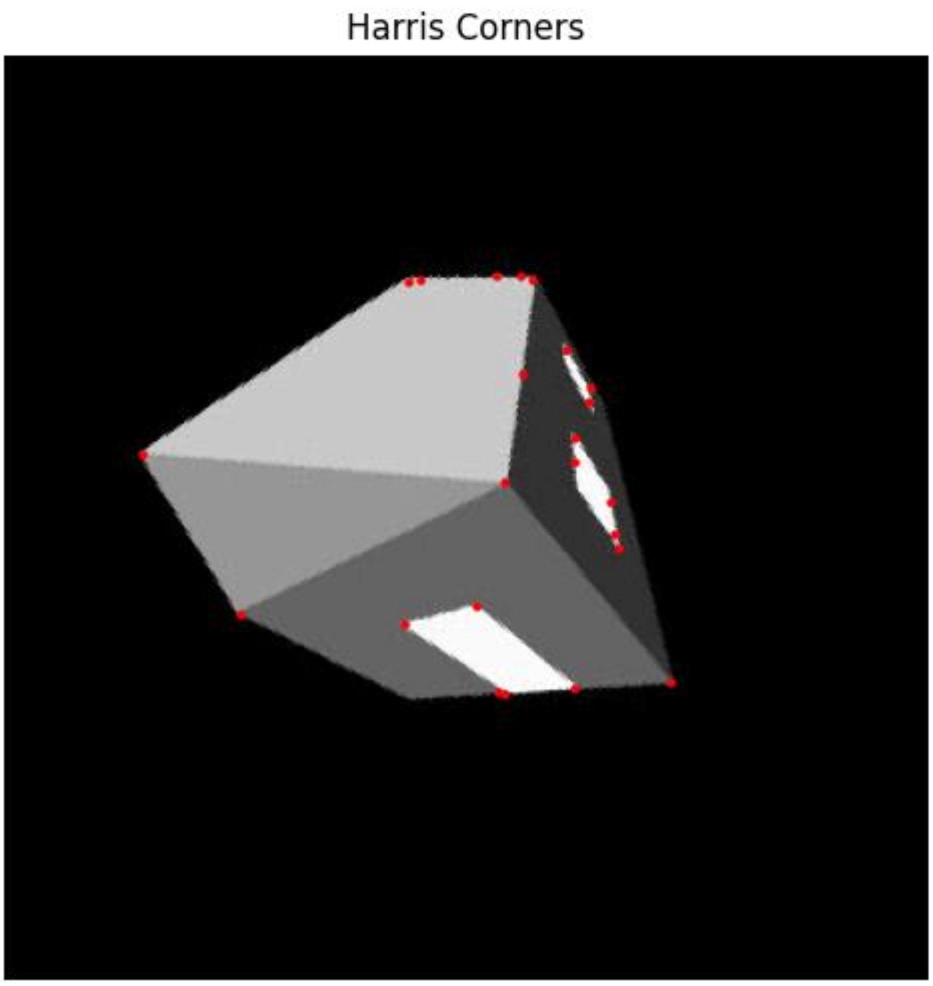

Figure 2 Harris corners where the image has been rotated by 45 degrees.

In Figure 2 it can be seen that new edges has been detected. Some of the new edges are evidently not edges, but are still marked as such. The reason for this can be that the sobel operator used is calculating the image gradients, and since these image gradients are now different, the gradients have now also changed. Also, some of previously horizontal and vertical lines are now diagonals, which also makes it harder for the algorithm to detect the edges.

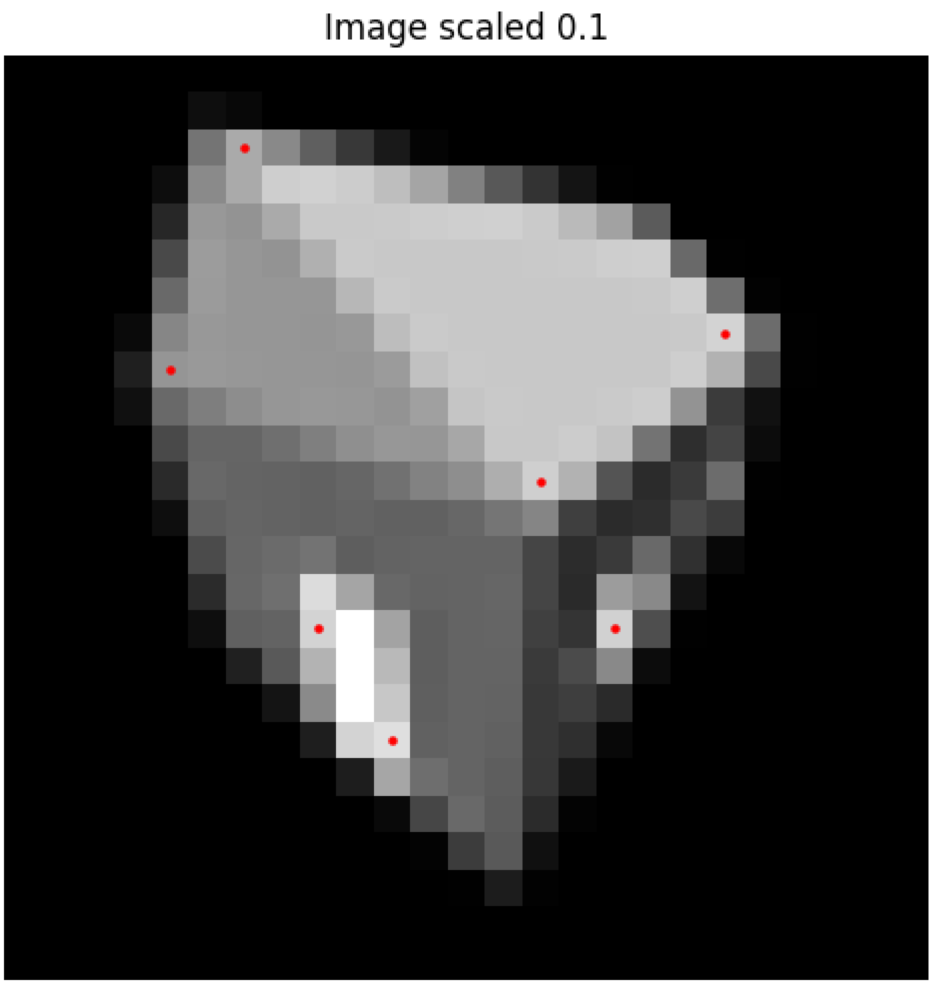

Figure 3 *Harris corners when the image has been scaled by 0.1.*

Figure 3 shows how the edge detector performs when the image is scaled down to 0.1 of its original size. The edge detector evidently works worse than on the original image. This is due to that the edges are not as sharp anymore, which also yields weaker image gradients.

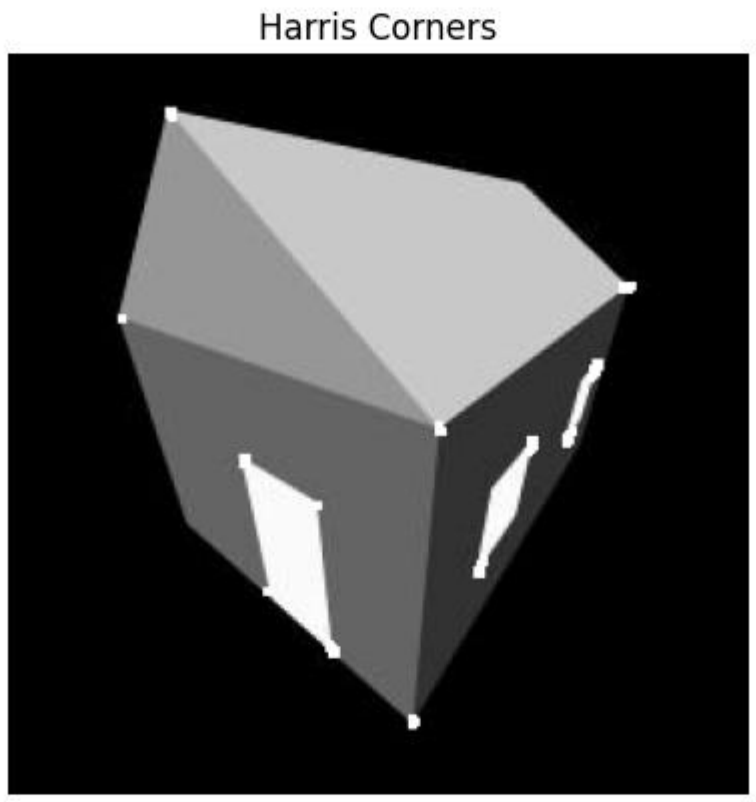

Figure 4 Harris corners with the built-in python function in the library "opencv".

Figure 4 shows how the corner detection when the built in openCV function "cornerHarris" was used. It can be seen that the built in function is providing similar results as to what the self-built version does, which indicates that the self-built function was done correctly.

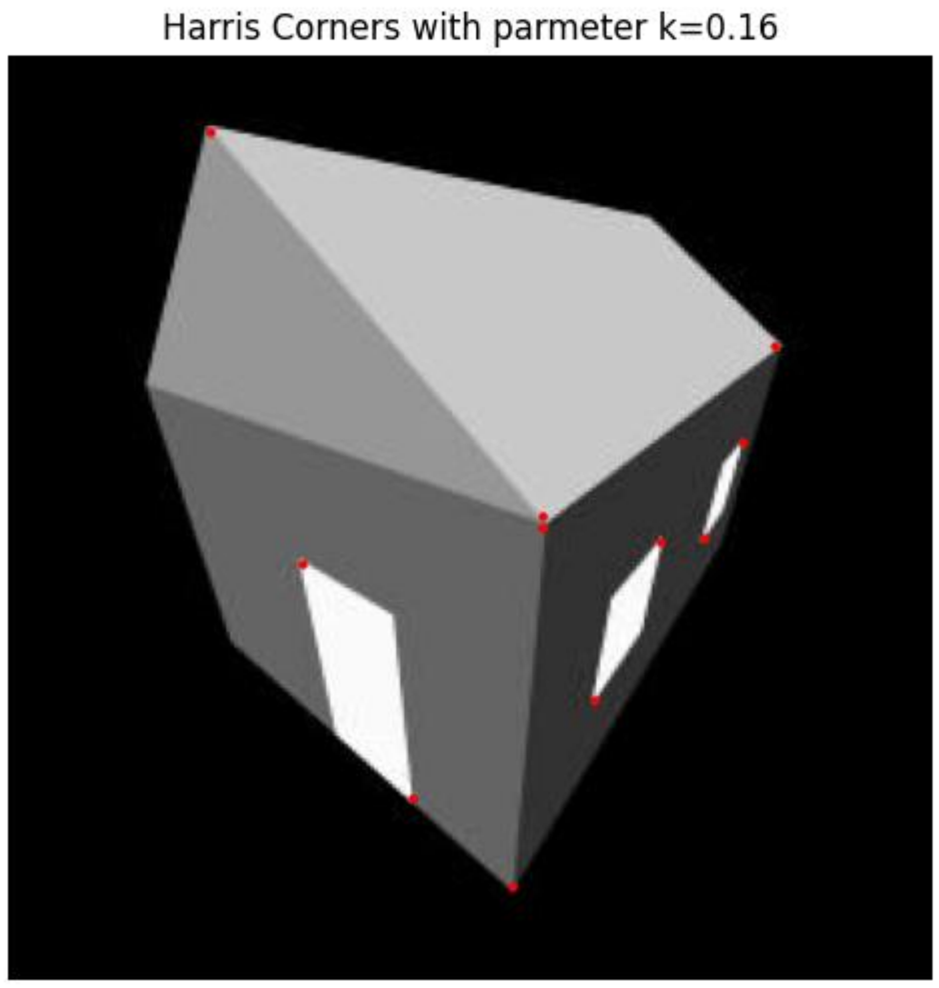

Figure 5 Harris corners where the k-value is set to 0.16

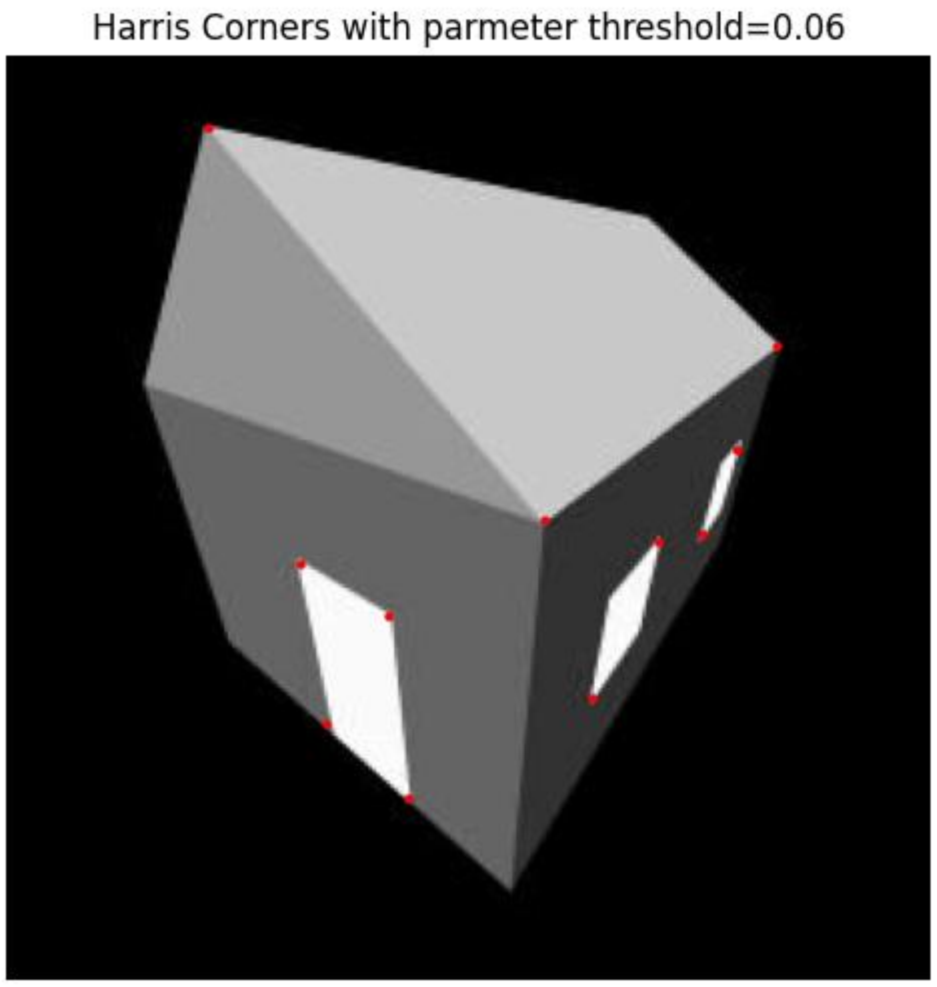

Figure 6 *Harris corners where the threshold is set to 0.16.*

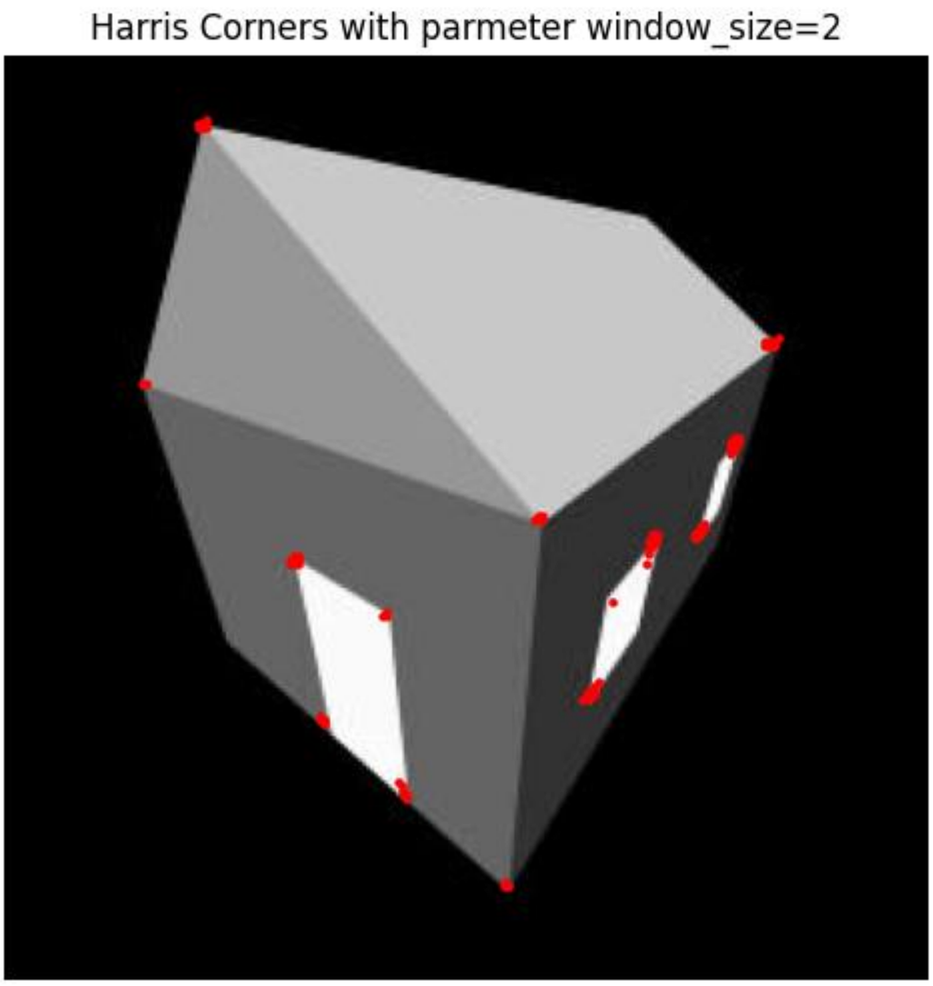

Figure 7 Harris corners where the window size is set to 2..

Figure 5 and Figure 6 shows a small proportion of some of the tests that were made with different parameters. When increasing the parameters k and the threshold will decrease the performance of the harris detector. It can also be seen that when the search space for the window size is smaller, the non maximum suppression does not work as well, which results in more corners detected (and all newly detected corners are wrong!).

Task 2

Task two is about using the SIFT detector and harris corners to detect features and match them to images. The SIFT features is calculated with the help of “stacking” the image in the Gaussian scale space. The images are stacked in the scale space with regards to smoothing, where the “bottom” image is not smoothed, and the images above are scaled with:

$$\sum_{k=0}^{n} \theta * s^{k}$$

Where s specifies the smoothing of the image.

For the stack of images, there is a new stack of images calculated, which is the difference between the two images, which is the normalized Laplacian of the gaussian of the image. With these new stack of images, there is (similar to the non-maximum suppression algorithm) a grid which is used the find the extremes in the grid (in this case the grid is n*n*n size). When the extremes have been calculated, SIFT interest points are calculated by removing all the weak extremes. In the python code the library OpenCV has been utilized to implement the SIFT detectors. In the python library, the function returns the keypoints as well as descriptors. These descriptors are numerical representations of the area around the keypoints, and are used to match features between images. Then they are matched with a brute force matcher, which compare the descriptors of one set with the descriptors of another set with the help of a distance metric.

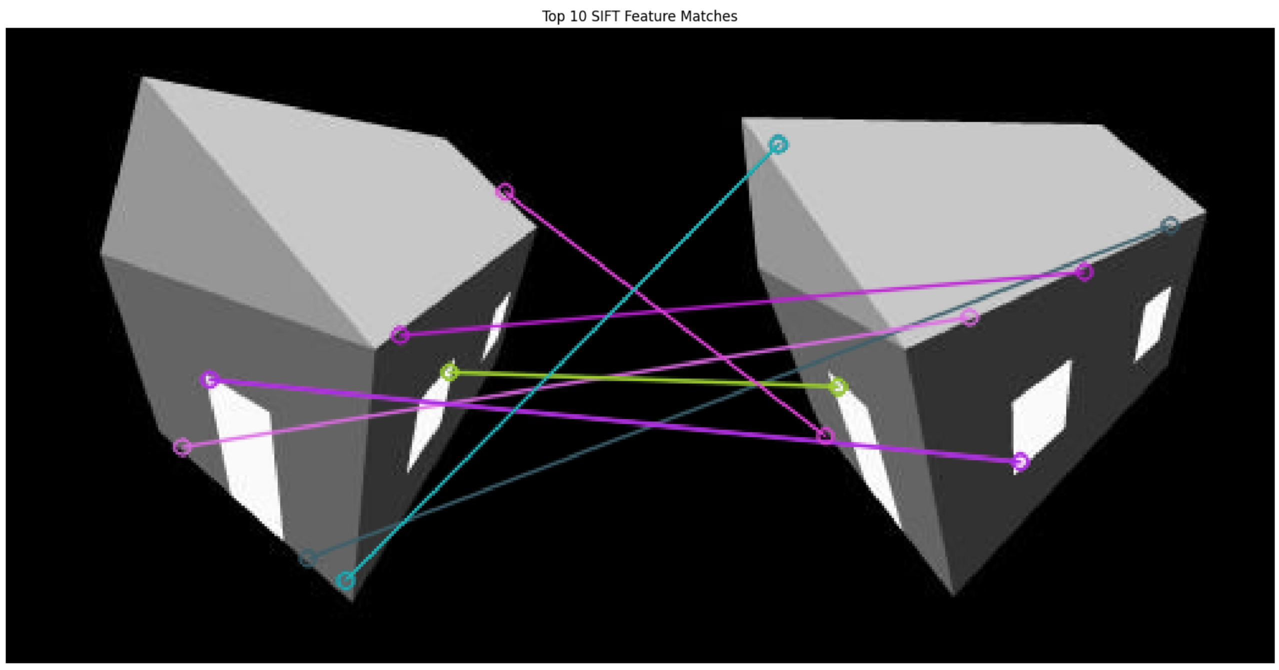

Figure 8 Feature detection using the SIFT algorithm for the images “left” and “right”.

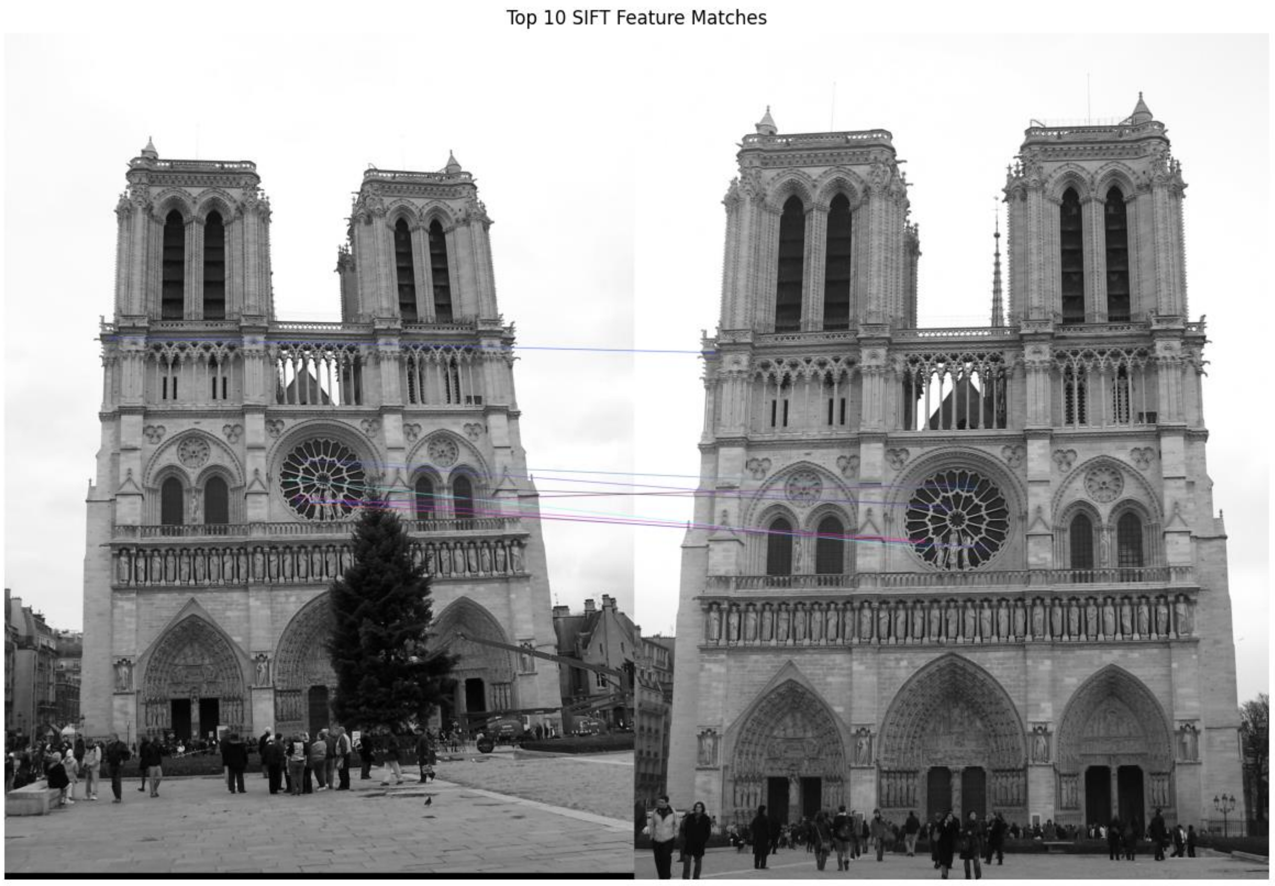

Figure 9 Feature detection using the SIFT algorithm for the images "nortredame1" and "notredame2".

Figure 8 and Figure 9 above shows that the SIFT algorithm works somewhat well for detecting features and match them between the pictures. For the picture of the house (Figure 8) some of the detected features are not accurate (where for example some doors are detected as a window). The feature detection works better for the picture of Notre Dame, probably because the images are more detailed.

Top 50 Harris Feature Matches

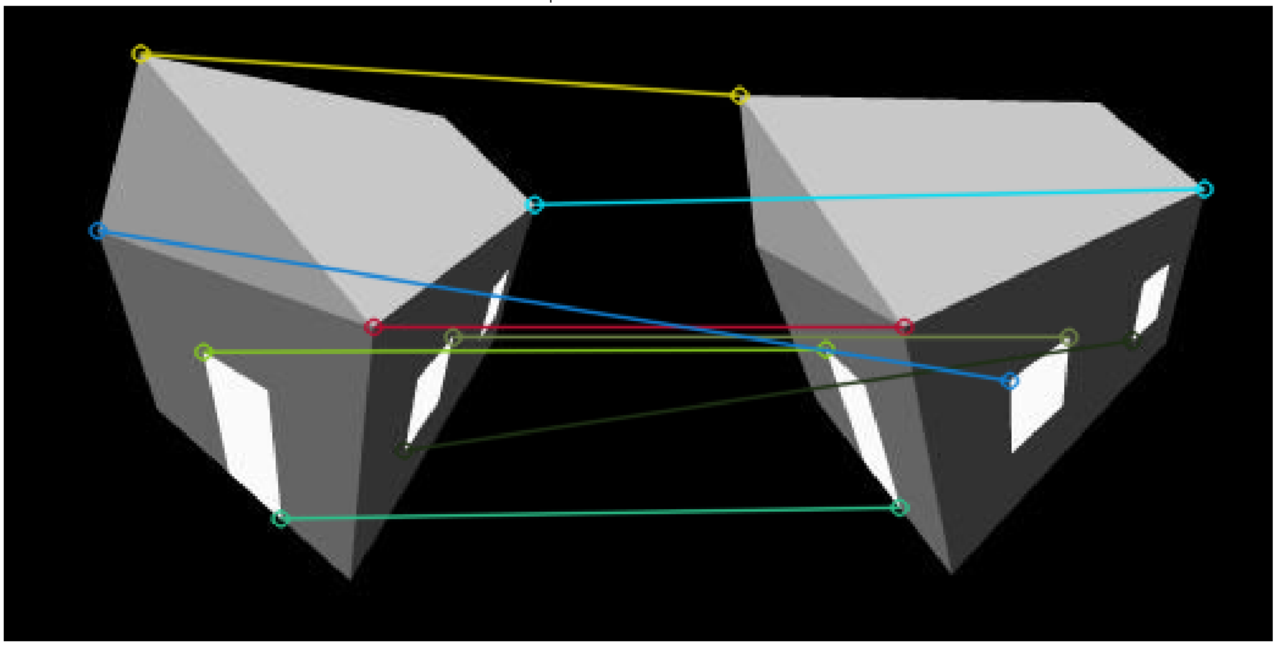

Figure 10 Feature detection using the Harris corners algorithm for the images "left" and "right".

Top 50 Harris Feature Matches

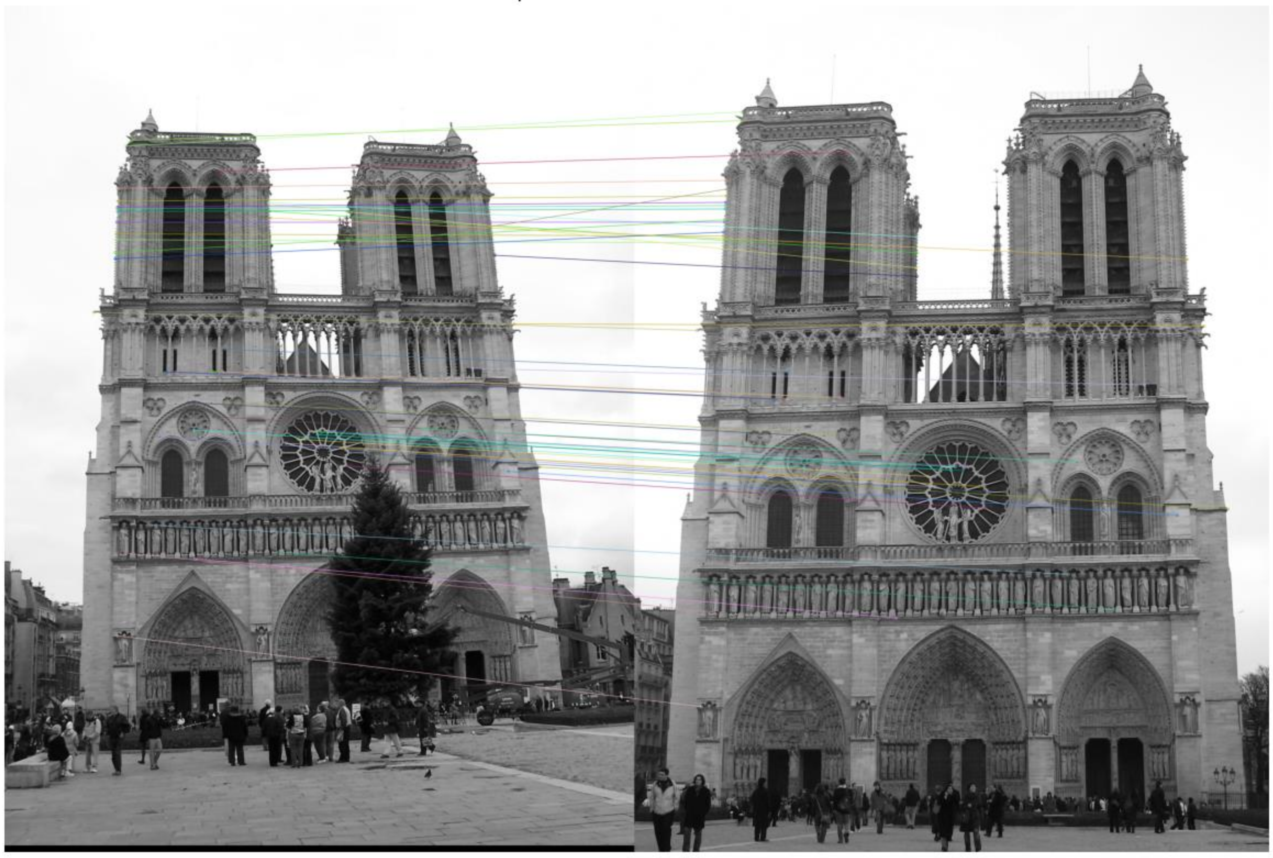

Figure 11 Feature detection using the Harris corners algorithm for the images "nortredame1" and "notredame2".

Figure 10 and Figure 11 shows the feature detection with the help of Harris corners. Here it is evident that the harris corners are performing significantly better, and most of the features in both the images are successfully matched.

Task 3

Task 3 is about applying the feature detection to a robotic like scenario. Here, we have been given two folders which have the same video (where one is tilted to the left, and on to the right). We were supposed to choose a feature detector, where I choose SIFT. The fps of my function with plotting was determined to be around 3.3 fps (due to slow plotting) and around 16 fps without plotting (meaning, only the matches between the images were made). For the custom images, my algorithm got around 81 matches per image, and no image had no features detected. For the PennCOSYVIO dataset, there were on average 578 pairs, and no image had no matches. For further look into how this was implemented, please review the notebooks appended to this pdf.