# SYDE 671: Assignment 2 Questions

Pascale Walters

**E1:** Webcam Fourier decomposition demo

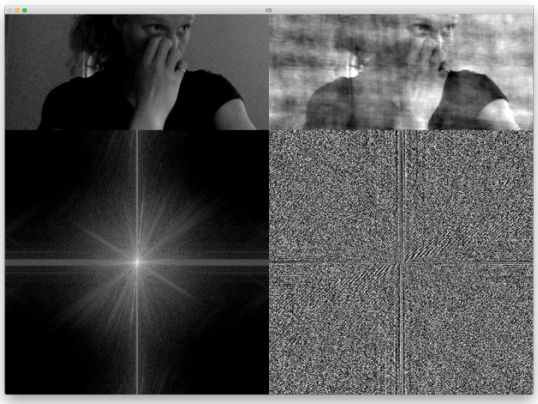


Figure 1: Amplitude replaced with that of another image.

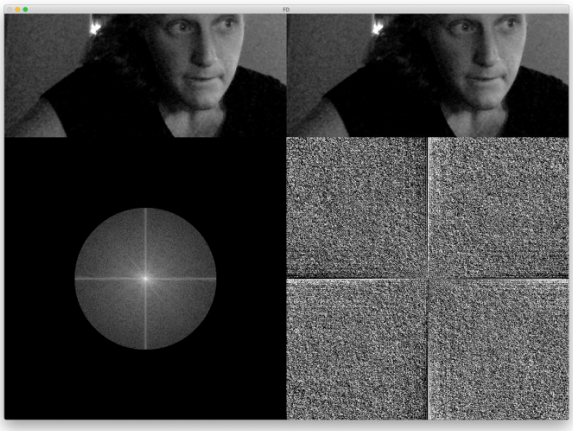


Figure 2: Low pass filter applied over amplitude.

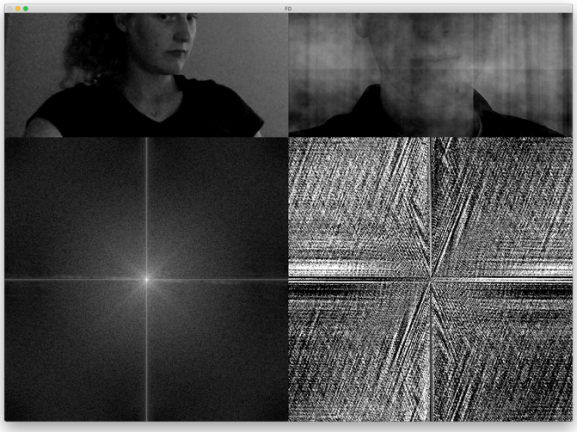
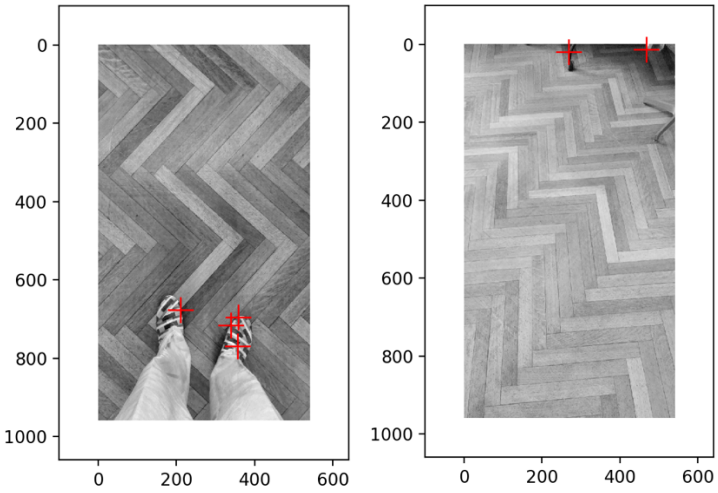


Figure 3: Phase replaced with that of another image.

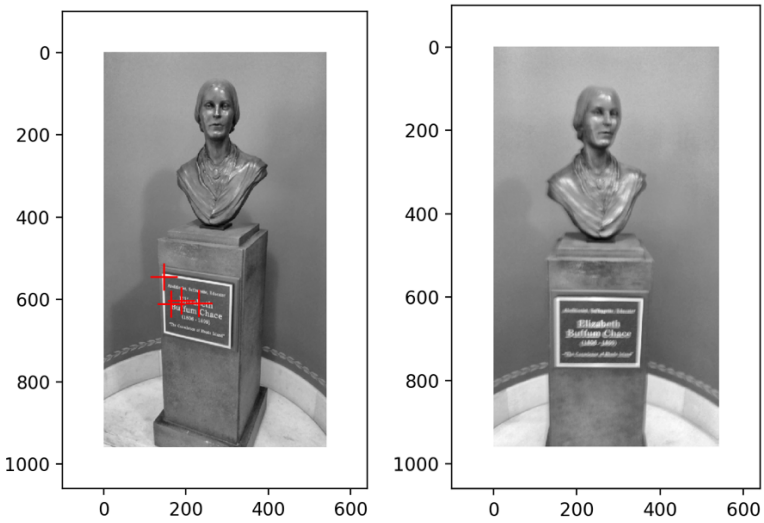
**Q1:** Imagine we wished to find points in one image which matched to the same world point in another image—so-called feature point correspondence matching. We are tasked with designing an image feature point algorithm which could match world points in the following three pairs of images. Please use the included python script plot_corners.py to find corners using Harris corner detection. Discuss the differences in the returned corners (if any) for each image pair and what real world phenomena or camera effects may have caused these differences. Then discuss which real world phenomena and camera effects might cause us problems when matching these features. Please provide at least one problem per pair.

*RISHLibrary*



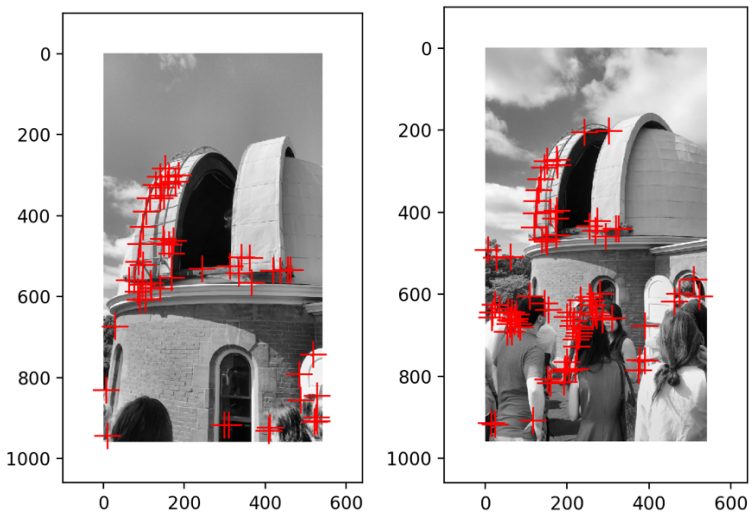

In this pair of images, there are no corners found on the floor, only on the edges of the shoes and the chair legs. Since the corners detected in the first image are not found in the second, it would be impossible to find the correspondences between these two images. Matching corners are not found between the two images because of the change of the field of view causes the corners to be lost. Low contrast in the flooring likely caused there to be no corners detected there.

*Chase*



No corners were detected in the second image, however there were several detected in the first image. The lack of corners is likely because the second image is so blurred that no corners could be found with the Harris corner detector. In the first image, corners were detected in the area of high contrast on the plaque.

*LaddObservatory*




This pair of images had some more success detecting corresponding points, especially around the left side of the observatory. The second image has many confounding corners detected on the people in front of the observatory, which cover some of the corners detected in the first image. The curved surfaces also face some issues due to scaling (i.e., more corners are detected on a curve at a higher scale). There may be problems matching the points due to scaling and occlusions.

**Q2:** In the Harris corner detector, what do the eigenvalues of the 'M' second moment matrix represent? Discuss both how they relate to image intensity and how we can interpret them geometrically.

The values in the M matrix represent the spatial derivatives of the intensities in a local area, as defined by a window function. The eigenvalues and the corresponding eigenvectors represent the magnitudes and directions of maximum change of the intensity in the neighbourhood. Corners in the Harris corner detector are defined as areas that have high change in intensity in all directions. When both eigenvalues of the M matrix are large (beyond some cutoff value), there is sufficient change in intensity in all directions, and the patch is identified as a corner.

**Q3:** Given a feature point location, the SIFT algorithm converts a 16×16 patch around the feature point into a 128×1 descriptor of the gradient magnitudes and orientations therein. Write pseudocode with matrix/array indices for these steps.

*Notes:* Do this for just one feature point at one scale; ignore the overall feature point orientation; ignore the Gaussian weighting; ignore all normalization post-processing; ignore image boundaries; ignore sub-pixel interpolation and just pick an arbitrary center within the 16×16 for your descriptor. Please just explain in pseudocode how to go from the 16×16 patch to the 128×1 vector. You are free to simplify the gradient computation.

*# You can assume access to the image, x and y gradients, and their magnitudes/orientations.*

```
image = imread('rara.jpg')
grad_x = filter(image, 'sobelX')
grad_y = filter(image, 'sobelY')
grad_mag = sqrt( grad_x .^2 + grad_y.^2 )
grad_ori = atan2( grad_y, grad_x )
# Takes in a feature point x,y location and returns a descriptor
def SIFTdescriptor(x, y):
def SIFTdescriptor(x, y):
    descriptor = zeros(128,1)

    for i in range(0, 4):
        for j in range(0, 4):
            a, b = i * 4, j * 4
            c, d = (i + 1) * 4, (j + 1) * 4

            # Construct histogram for the 4x4 patch
            hist = zeros(8, 1)
            for k in range(a, c):
                for l in range(b, d):
                    # Would need to be normalized based on the reference orientation
                    bin_index = int(grad_ori[k, l] * 8 /(2 * pi))
                    # Weight the vote by the magnitude of the gradient
                    hist[bin_index] += grad_mag[k, l]

            descriptor[((i * 4 * 8) + (j * 8)):((i * 4 * 8) + (j + 1) * 8)] = hist
```

**Q4:** Explain the difference between the Euclidean distance and the cosine similarity metrics between descriptors. What might their geometric interpretations reveal about when each should be used? Given a distance metric, what is a good method for feature descriptor matching and why?

The Euclidean distance metric is calculated as follows:

$$d_E(\bar{x}_1, \bar{x}_2) = \sqrt{\sum_{i=1}^{n} (x_{1,i} - x_{2,i})^2}$$

The cosine similarity metric is calculated as follows:

$$d_C(\bar{x}_1, \bar{x}_2) = \frac{\bar{x}_1 \cdot \bar{x}_2}{|\bar{x}_1||\bar{x}_2|}$$

The Euclidean distance metric calculates the pairwise distance between each element in the two vectors, whereas the cosine similarity metric calculates the angle between the two vectors. The Euclidean distance metric determines the magnitude of the distance between the two points and the cosine similarity metric does not consider the magnitudes of the vectors. The Euclidean distance metric should be used when the magnitudes of the vectors are important and the cosine distance metric should be used when the magnitudes are not important, or the vectors should be normalized.

Given a distance metric, feature descriptor matching would occur between descriptors that are a small distance from each other, implying that they have similar features. A good method for matching would ignore keypoints in one image that do not have a match in the second image. This could be done by checking the ratio between the two nearest neighbours in the second image and if it is within some threshold, choosing the nearest neighbor as a match.

**Something to think about:** In designing a feature point matching algorithm, what characteristics might we wish it to have? How might two world points change in appearance across photographs? Consider that we might allow brightness or contrast changes, or texture changes, or lighting changes, or

geometric changes in appearance like rotation and translation in three dimensions or camera perspective effects. All exist between some two photographs of real-world points.
We are faced with a fundamental trade-off between feature point invariance (how much variation in appearance I allow and still say that two points are the same) and discriminative power (our ability to say that two points are different or the same at all).
How should we design for this trade-off?

When designing a feature point matching algorithm, it should work well, even when there are changes in appearance between the two images. There could be changes in appearance due to a change in the viewpoint, field of view, or photometric or geometric effects. The trade-off between feature point invariance and discrimination would have to be accounted for depending on the specific application of the point matching algorithm.