COMPSCI 773
Intelligent Vision System

Image Stitching
Milestone 1 Report

March 23, 2022

Author: Neville Loh, Nicholas Berg

1 Summary

Motivated by the construction of a panorama image stitching application, the relative position of images to each other must be determined. An initial step in the construction of such an application is to identify features, so that these features can be matched between discreet images. In an investigative exercise to achieve optimal image feature identification, three different feature detection methods have been implemented. Two different implementation of Harris Corner Detection have been implemented in the Python programming language. Generally we have used the implementation based on the lecture content from this course (UoA COMPSCI 773), referred to as the Berg-Loh implementation. This implementation utilised Gaussian and Sobel filtering, and selects the 1000 features with the highest coroner response form the Harris Corner Detection function (we refer to this response as "cornerness") after non-max suppression of neighbouring signals (adjacent or diagonal pixels). A second implementation based on Solem (2012) was used for comparison. This implementation uses a threshold approach (10% highest responses) without suppression of neighbouring values. Typically in the images used, this 10% threshold resulted in more than 1000 corners being detected. Finally, a third naive approach was designed and implemented relying on simple mean squared intensity differences. These three corner detection methods were tested on 3 distinct images (a mountain landscape, a city landscape, and a children's playground). We continued the investigation by examining the Harris empirical constant (generally reffered to as k, but here referred to as α between values of 0.04 and 0.06. Noting that this resulted in only minimal change to the output, a wider range of 0.01 to 2.0 was investigated. The Gaussian window size was varied between 3x3 and 9x9, to further optimise coroner detection parameters. Finally, the output data was plotted as a frequency histogram over "cornerness" and distance between corners. This statistical analysis was motivated by a broad distribution offering the maximum diversity of features.

2 Introduction

To enable computers to perform intuitive functions, computer vision must be able to extract information from images in an intuitive manner. One aspect of this information extraction is feature detection. Harris corner detection was originally proposed by Chris Harris and Mike Stephens in 1988, motivated by the "desire to obtain an understanding of natural scenes".

3 Overview of the Harris Corner Detection Algorithm

The Harris Corner Detection Algorithm is used for the following project. Although some optimization have been done, the general algorithm follows the lecture content for the courses. The implementation is written in *./imageProcessing/harris.py* and textit./imageProcessing/harris˙util.py and With alternative implementation written in *./imageProcessing/SolemHarrisImplementation.py* .

3.1 Image prepossessing

The input image is first scaled to a $[0 - 255]$ integer array in grey scales. This step is done to speed up the computation of the algorithm without significant lost of information. The result image still retrains a lot of details feature of the original image. Minor changes in intensity in small regions does not provide significant value in the goal of image stitching. Thus these noise are removed by applying a Gaussian filter with a 3 by 3 windows using 2D convolution technique.

3.2 Calculating the shift in gradient

After the prepossessing, first degree Taylor approximation is used to calculate the derivatives of image I since we assume changes are small for small interval.

$$I(x + \Delta x, y + \Delta y) \approx I(x, y) + I_X(x, y) + I_y(x, y)\Delta y$$

Sobel filter, an Isotropic 3×3 Image Gradient Operator, is used to compute the gradient effectively. The Sobel–Feldman operator is based on convolving the image with a small, separable, and integer-valued filter in the horizontal and vertical directions and is therefore relatively inexpensive in terms of computations.

Listing 1: Gradient Calculation

```
def sobel(px_array: ImageArray) -> Tuple[ImageArray, ImageArray]:
    """
    Apply sobel filter using 2D convolution
    Returns: image i_x, i_y
    """
    image_height, image_width = np.shape(px_array)
    ix_kernel = ([-1, 0, 1], [1, 2, 1])
    iy_kernel = ([1, 2, 1], [-1, 0, 1])

    i_x = computeSeparableConvolution2DOddNTapBorderZero(px_array, image_width, image_height,
                                                         kernelAlongX=ix_kernel[0],
                                                         kernelAlongY=ix_kernel[1])
    i_y = computeSeparableConvolution2DOddNTapBorderZero(px_array, image_width, image_height,
                                                         kernelAlongX=iy_kernel[0],
                                                         kernelAlongY=iy_kernel[1])
    return np.array(i_x), np.array(i_y)
```

The Harris response can be optimize by just computing M, the structure tensor for the image, where

$$M = \sum_{(x,y)\in W} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = \begin{bmatrix} \sum_{(x,y)\in W} I_x^2 & \sum_{(x,y)\in W} I_x I_y \\ \sum_{(x,y)\in W} I_x I_y & \sum_{(x,y)\in W} I_y^2 \end{bmatrix}$$

.

By processing the intermediate result from the last step, the following can be computed. A Gaussian filter with a window of 5 by 5 is applied to the derivative. Different widows size is also tested, this will be further discussed in later session of the report.

Listing 2: Corner Activation Calculation

```
def get_square_and_mixed_derivatives(i_x, i_y):
    return np.square(i_x), np.square(i_y), np.multiply(i_x, i_y)

def get_image_cornerness(ix2, iy2, ixiy, alpha):
    result = np.multiply(ix2, iy2) - np.square(ixiy) - (np.square(np.add(ix2, iy2)) * alpha)
    return result


ix2, iy2, ixiy = t_left = get_square_and_mixed_derivatives(ix, iy)
ix2_blur_left, iy2_blur_left, ixiy_blur_left \\
     = [compute_gaussian_averaging(img, windows_size=gaussian_window_size) for img in t_left]
corner_img_array = get_image_cornerness(ix2_blur_left, iy2_blur_left, ixiy_blur_left, alpha)
```

3.3 Thresholding the output

It is expected that all pixels close to the corner region will have high activation. This means that 1 corner might yield multiple pixels that will potentially pass the threshold while not providing significant information regarding the features of the input image. To address this problem, local non-maximal suppression is applied for the entire image. A 3 by 3 pixel windows is iterated over the image while the center pixel is suppress if it is not a local maximum compare to its nearest 8 neighbour. After the non-max suppression, each pixel is made in to a object with its coordinate and Harris response. These object are then pushed in to a heap with a decreasing order.

Listing 3: Outputting top n corner with highest response

```
class Corner:
    def __init__(self, index: Tuple[int, int], cornerness: float):
        self.y, self.x = index
        self.cornerness = cornerness

    def __lt__(self, other):
        return self.cornerness > other.cornerness

    def __eq__(self, other):
        return self.cornerness == other.cornerness


# Non-max suppression
corner_img_array = bruteforce_non_max_suppression(corner_img_array, window_size=3)

# Prepare n=1000 strongest conner per image
pq_n_best_corner = heapq.nsmallest(n_corner, get_all_corner(corner_img_array))

# Preparing n highest response
pq_n_best_corner_coor = [(corner.x, corner.y) for corner in pq_n_best_corner]
```

The top 1000 point is output as result, these can then be plotted using the matplotlib library.

4 Results

The follow section provides the results from our investigation:

4.1 1000 Strongest Harris Corners Detection of 1000 strongest corners with simple non max 3x3 Suppression

[h!]

The main task for this project was the implementation of the Harris corner detection to output the 1000 strongest corners from two images Mt Tongariro. Additionally, applying only the Harris corner function yeilded

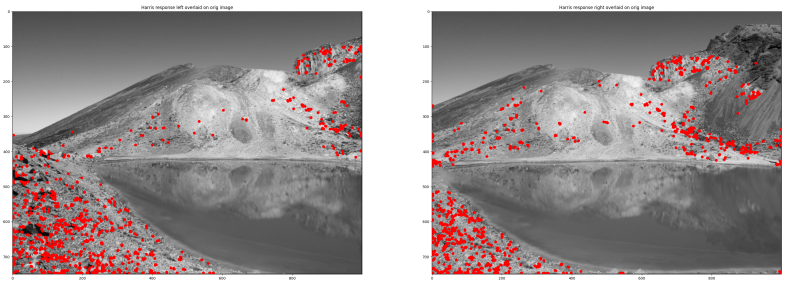

Figure 1: Two images of Mt Tongariro with the 1000 strongest Harris corner features after non-max supression.

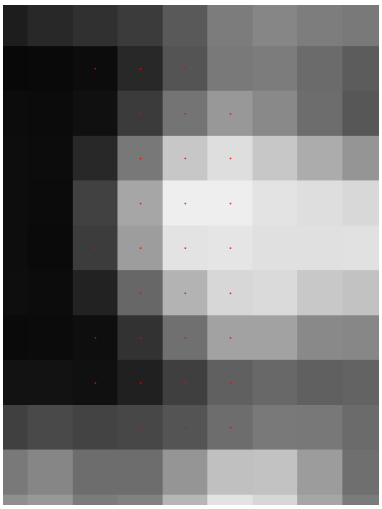

Figure 2: A enlarged view of a corner feature from the Mt Tongariro image. Without non-max suppression applied, the sharp change in intensity in this region results in most pixels being classified as a corner feature. Simple 3x3 window non-max suppression produces the desired output.

4.2 Comparison of Three Corner Detection Implementations

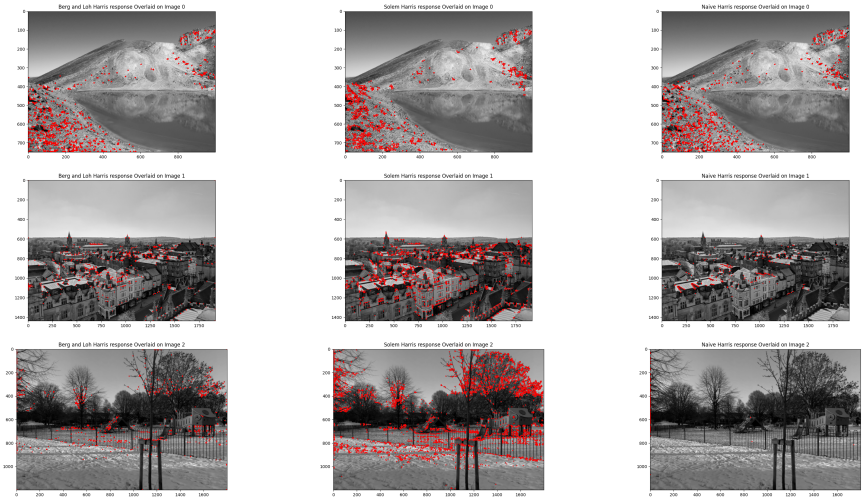

Figure 3: Three different corner detection implementations (Berg-Loh Harris Corners, Solem Harris Corners, and a novel naive corner detection algorithm) applied to three different subjects (Outdoor natural landscape of Mt Tongariro, Outdoor city landscape of Oxford, Children's playground in a snow covered park.)

4.3 Investigation of Harris Corner constant α (also k)

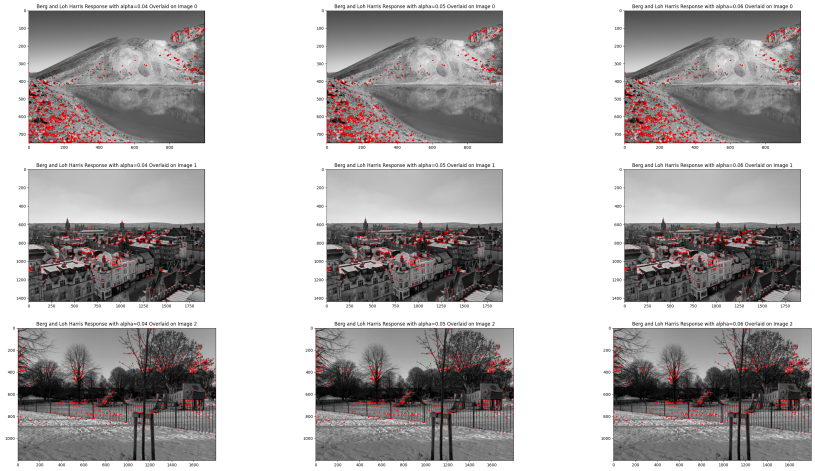

Figure 4: The Harris constant, referred to by either α or k, is a scalar constant that shifts the "edge"-ness of the output by scaling the value of the trace of the matrix of derivatives. It is typically varied between 0.04 and 0.06, but there is not a consensus on which applications which values are more suitable for. This combined image of Alpha values of 0.04, 0.05, and 0.06 applied to three different images suggests only minor changes in the corners detected with these changes in α.

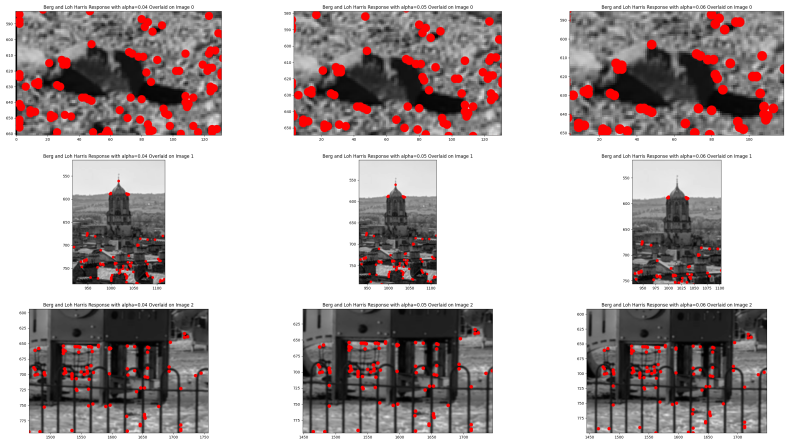

Figure 5: This is the same data rendered in Figure 4, but focusing on a small subsection of each image. A more obvious difference occurs in the central row, where the tower spire is detected as a corner for α values of 0.04 and 0.05, but not 0.06. We expect it to have a strong response in the x direction, but a weaker response in the y direction, subsequently higher values of alpha lower its corner response.

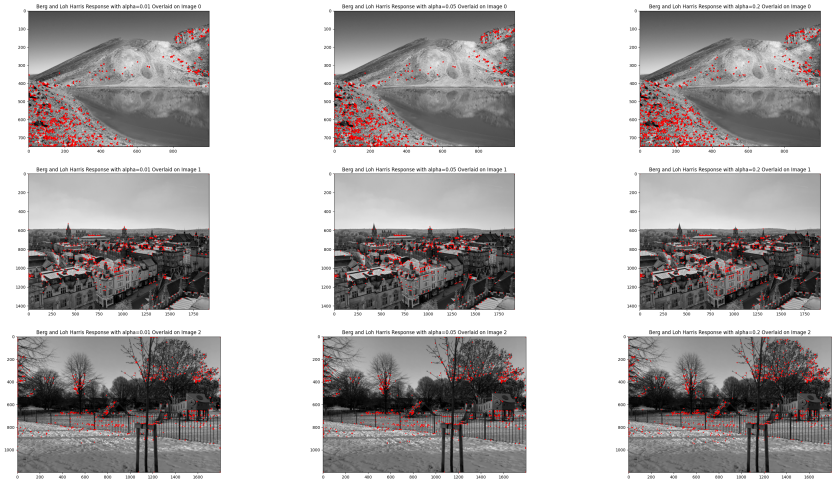

Figure 6: Given our initial investigations into α did not yield a significant difference in corner output, we chose to extend the values well beyond the conventional 0.04-0.06 range, choosing 0.01 (which more closely approximates the naive corner detection implementation) and 0.2 (strong edge suppression). Even this large variation in alpha produces similar results, with the strongest change being the images of the city of Oxford. This is expected, as man-made structures form regular shapes which will have the largest difference in the derivative of intensity change in x and y. A following figure focusing on a subsection of the image highlights this.

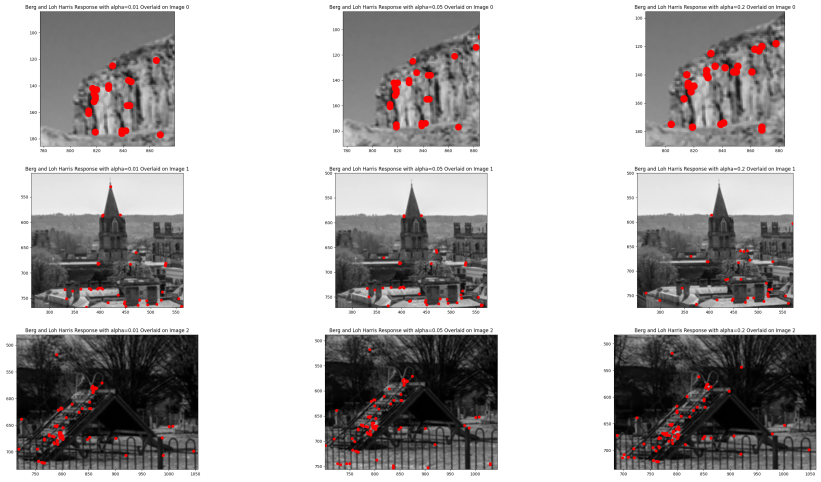

Figure 7: The larger variation in α corresponds to more variation in corner locations in this figure when compared with figure 5. Again, the regular geometric shapes in the image of Oxford in the center row provides the clearest example. With increasing values of α, we correspondingly lose the points at the top and then the right of the central tower in this subsection of the image.

4.4 Distribution of Distance between Corners as a possible heuristics for Effective Image Stitching Feature Detection

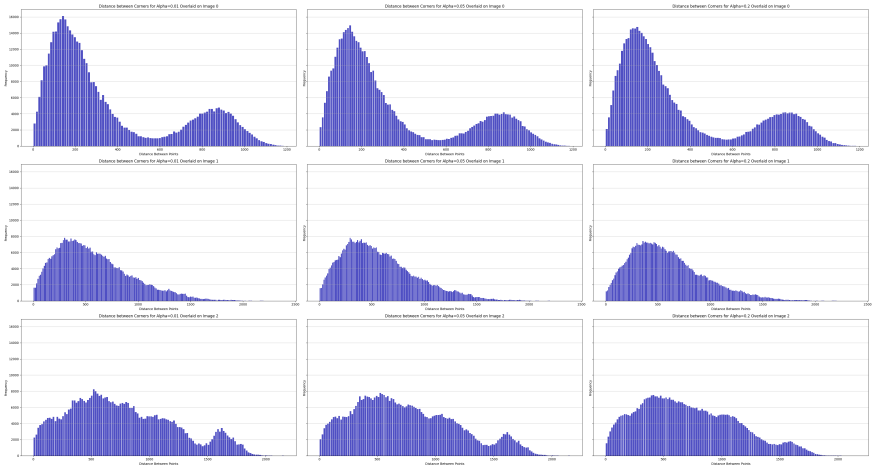

Figure 8: To further investigate how the α parameter was affecting the corner response of the image, several statistical analyses of the output were considered. One such output, displayed in this figure, is the frequency histogram of the distances between corners, plotted here for different values of alpha and different images. As a heuristic, is is presumed that a broader histogram will yield more distinct corners, and therefore a better signature for image stitching. The changes in alpha result in

4.5 Investigation of Gaussian Window Size on Corner Detection

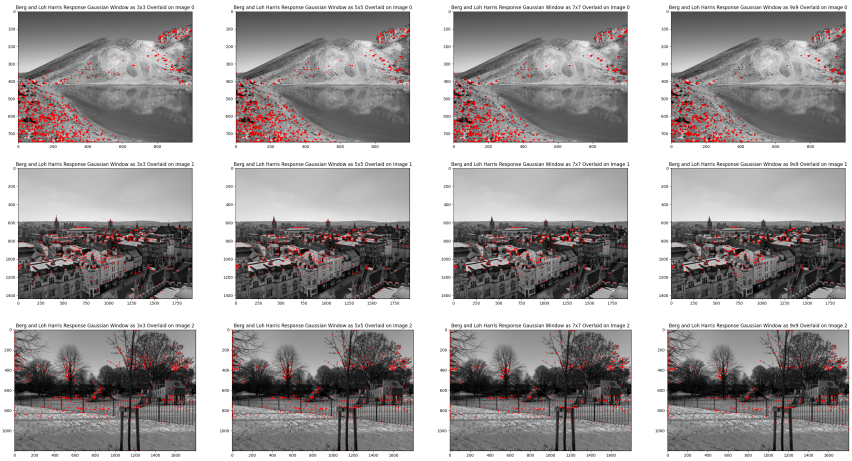

Figure 9: Having established α has only a slight impact on the corner distribution, we next investigated the size of the Gaussian Window. Values of 3, 5, 7, and 9 were selected and applied to each of the three images, with the results shown in the figure above.

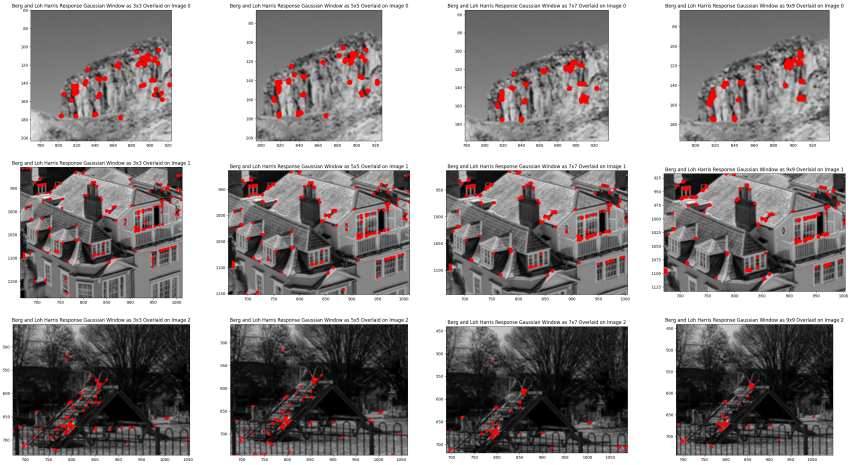

Figure 10: With a smaller region selected, it becomes clear that the Gaussian Window size is having an effect on the distribution of points, causing significant displacement in all three images between Gaussian window size. However, without further implementing a feature matching or stitching function, it is not obvious which set of corners would provide the best mapping for the desired function.

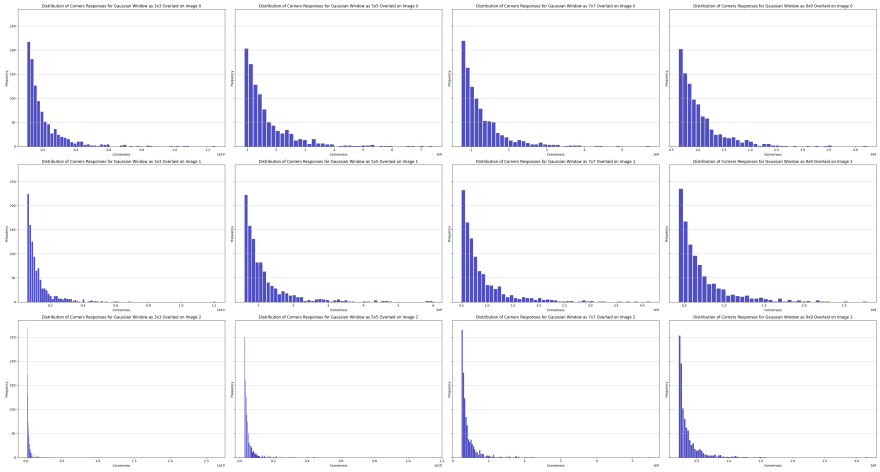

Figure 11: Applying the previously discussed statistical analysis, the presumed heuristic of a more diverse set of corners resulting in a more straightforward feature mapping was applied. The result can be seen that in all images, a larger Gaussian window results in a narrower distribution. This makes intuitive sense, as a larger Gaussian window effectively results in a higher degree of blurring for the image, and therefore a lower "cornerness" score at each pixel. It will be further investigated in future assignments if this results in a better mapping between images.

5 Conclusion

Motivated by the desire to use computer vision to combine two images with some overlap into a single larger image (panorama stitching), we have investigated corner detection through the implementation of three corner detection algorithms in the Python programming language. We then applied these 3 detection implementations to 3 pairs of images, which while distinct, represent typical subjects that a user might want to capture in a wider panoramic image. We then performed statistical analysis on different parameters (Gaussian window size, Harris constant value) and examined the distribution of "cornerness" of the corners, as well as the distribution of distances between the points. Our investigation revealed that different parameters and methods will vary in effectiveness based on the desired main subject of the image. Further investigation is warranted for optimising these parameters generally, for wider computer vision applications as well as image stitching specifically.

References

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

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

- End of document -