Clab-2 Report

ENGN4528

Benjamin Hofmann

u6352049

18/05/2020

ENGN4528 Computer Vision
Computer Lab-2

Task-1: Harris Corner Detector

1. **Understand provided code regarding corner detection**

2. **Complete missing parts, rewrite to 'harris.py' and design appropriate function signature.**

```
79 def harris_corner(img, Iy2,Ix2, Ixy, k):
80     row, column = img.shape
81     result = np.zeros((img.shape))
82     for r in range(row):
83         for c in range(column):
84             #convolution with gradient matrix
85             M = np.array([[Ix2[r,c], Ixy[r,c]],[Ixy[r,c], Iy2[r,c]]])
86             result[r,c] = np.linalg.det(M)-(k * (np.trace(M))**2)
87     return result
```

Figure 1: Code for Harris Corner detection

The function signature takes the image, Iy2, Ix2, Ixy and k as inputs. The image gives information on the shape of the resultant output. This could be omitted using information from inputs 2-4 but is included for greater simplicity. The variable is included as a control.

The code generates the appropriate matrix M from pixel derivatives. This matrix is then used to give a measure of corner response. This "cornerness" is saved in the results matrix. The value 'K' for the measure was set to 0.1. Changing values of K affected the strengths of cornerness values with 0.1 being found to give a sufficient balance between the number of points identified and the points being reasonable corners. Not all values of "cornerness" represent corners and so thresholding and non-maximal suppression is also applied. Thresholding set all values below a threshold limit to 0. The limit was set to 10^7 for figure 2 and 10^8 for figures 4, 6 and 8. Lowering the thresholding value yielded more corners but lead to false identification in some cases. Increasing the threshold value lead to lower numbers of corners being identified and some significant corners being missed.

Non-maximal suppression analysed the neighbouring pixels of each pixel and its value to 0 if it any of the neighbouring pixels had greater value. The location on the image of each corner was then processed through the indexing_func(), producing a tuple list of the x-y coordinate of each corner.

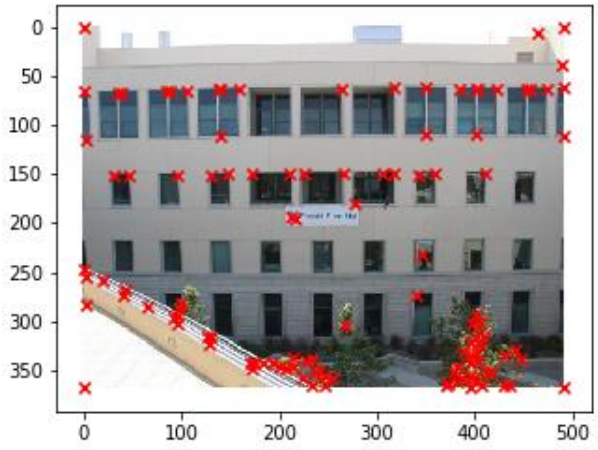
3. **Comment on block #5 and every line of solution in block #7**

Block #5: Block 5 makes use of the fspecial function defined in block 3. This function returns a filter that is used in block 6. The filter is produced through a function of sigma.

Block #7: Comments on solution block included in Task 1, section 2 above.

4. **Test function on the four test images and display results by marking the detected corners on the input images. Compare results to python's inbuilt Harris corner detector.**

The custom function notably includes the corners of the image and some image edges as corners. This could easily be removed if needed.



Figures 2 & 3: Custom Harris Corner detection of Harris_1(left) and inbuilt counterpart (right).

The inbuilt function was generally slightly more consistent for figure 3 in identifying the corners of windows and identified more sections of the sign. The custom function used to produce figure 2, picked up largely the same locations as the inbuilt function but was slightly less consistent, possibly due to the non-maximal suppression and thresholding stages.

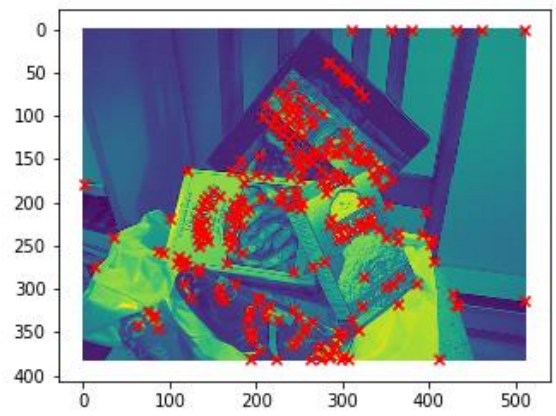
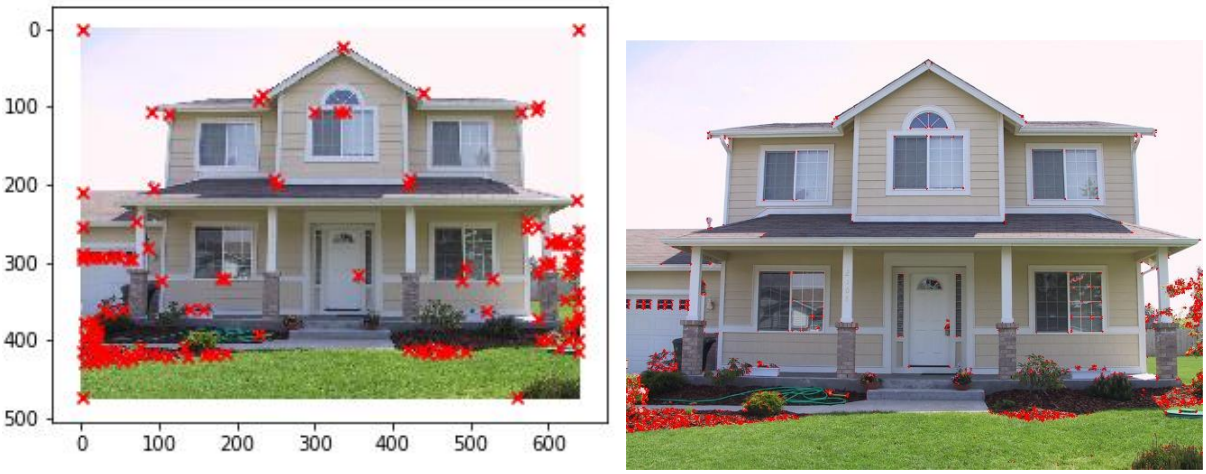


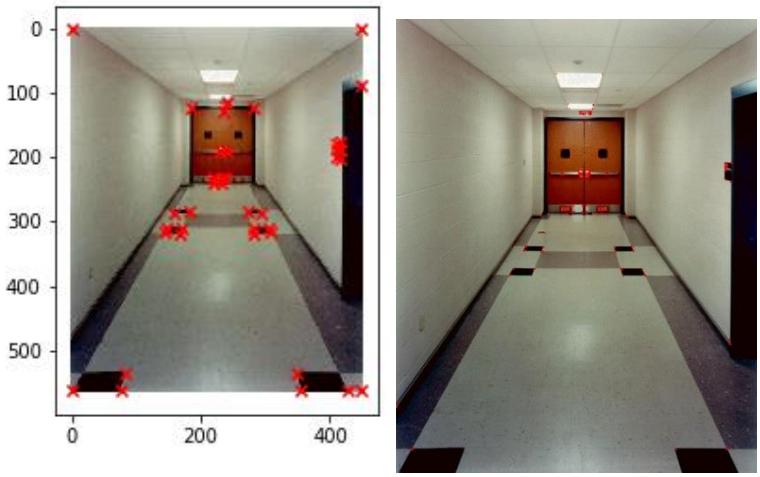
Figure 4 & 5: Custom Harris Corner detection of Harris_2 (left) and inbuilt function (right).

The colour distortion of figure 4 is due to not correctly representing the image with pyplot. Despite the apparent contrast, both figures identify nearly all the same corners. As with all the custom harris corner images, there are some edge cases identified as corners. These were kept as I believe they still identified feature points relevant to the image.



Figures 6 & 7: Custom Harris Corner detection of Harris_3 (left) and inbuilt function (right).

Similarly, to the Harris_2 image, figures 6 & 7 identify most of the same key areas. The bushes in both contain a significant number of corners. The inbuilt function identifies the windows much better than the custom functions, this is probably due to thresholding. In general the custom function appears to have fewer points meaning that the thresholding value was probably greater than the value used in the inbuilt function.



Figures 8 & 9: Custom Harris Corner detection of Harris_4 (left) and inbuilt function (right).

The Harris_4 image show in figures 8 & 9 gives relatively distinct corners to identify. Both functions appear to largely identify the same corners although it is displayed more clearly with the custom-harris function.

5. **Discus factors that affect the performance of Harris corner detection.**

There are a number of factors that could affect the performance of Harris corner detection. Naturally, the clarity of the image used and its representation in grayscale. If the image doesn't display the necessary characteristics well then it may be appropriate to apply Gaussian blur or high pass filters. The parameter K value will affect significantly affect the precision of the corner detector. Greater values of K are found to have fewer false positive result, however the greater value may also result in fewer real corners being identified. Similarly, the thresholding value is important in determining how many corners are recorded and what value constitutes a relevant corner. Too great a threshold and many real corners may not be identified while too low a threshold might aid in providing false corners. The aperture of sobel derivative could also be significant in corner detection. If looking for a larger corner or accounting for some noise increasing the aperture could be a reasonable action but it may likely lead to difficulty in finding distinct corners. Increasing the aperture would also slow down the calculation speed. Non-maximal suppression is another important factor of the Harris corner performance. This step helps represent corner locations more clearly and can help identify clearer corners in more noisy images.

Task-2: K-means Clustering and Colour Image Segmentation

1. **Implement K-mean function to process data points and the number of clusters to produce several clusters of point.**

Unfortunately, a valid K-means function was not produced although a number of steps were taken to code some sections of the algorithm.

2. **Apply K-means function to colour image segmentation. Compare segmentation results using different numbers of clusters, with and without pixel coordinates.**

Using a larger number of clusters will slow the algorithm in an almost linear manner (double the pixels, double the time of computation). This is due to the fact that each process within each iteration is generally repeated for each individual cluster. Segmentation with pixel coordinates might skew a segmentation to just exist in a particular section of the image. Segmentation without pixel input would simply converge each pixel value to the nearest cluster colour. Thus, depending on the image there may be multiple segments

3. **Implement K++ means algorithm. Summarize the key steps, implement as new initialisation strategy. Compare segmentation performance of new strategy.**

The key steps a K++ means algorithm would have are:

1. Select the first center point at random

2. Select a new center point assigning it's position to the probability given in equation 1.

$$\frac{D(x)^2}{\sum_{x \in X} D(x)^2} \qquad \text{(Equation 1 [3])}$$

3. Repeat step 2 for all remaining center points required

4. Continue with K-means algorithm

The algorithm with K++ means initialisation is described as)(log-k)-competitive [3] meaning that for large images and high numbers of clusters it has a significant improvement in

performance. The K++ means algorithm should in theory yield more relevant results. This is because choosing cluster positions that are more likely to be different will likely avoid the separating of very similar colours. This is a potential issue as the effect is arbitrary and may hinder interpretation of the final image. If for example, lighting subtly affects a feature's otherwise uniform colour, the feature may be unnecessarily segmented (sometime almost randomly) leading to lose of information. The selection of more different clusters also improves the readability of the output so humans don't have to distinguish between relatively similar colours.

Task-3:

1. **Take 10 different frontal pictures, convert to grayscale, align and resize. Explain why alignment is necessary**

Alignment is important to ensure that all important features of the face that may be detected are present and in a similar position across images. If a feature (such as a nose) which might be detected is out of position then the eigenvector describing this will return a significantly different value and thus not match accurately.
Personal images are included in the training and test sets respectively.

2. **Train an Eigen-face recognition system**
    a. **Perform PCA**
    b. **Display the mean face**

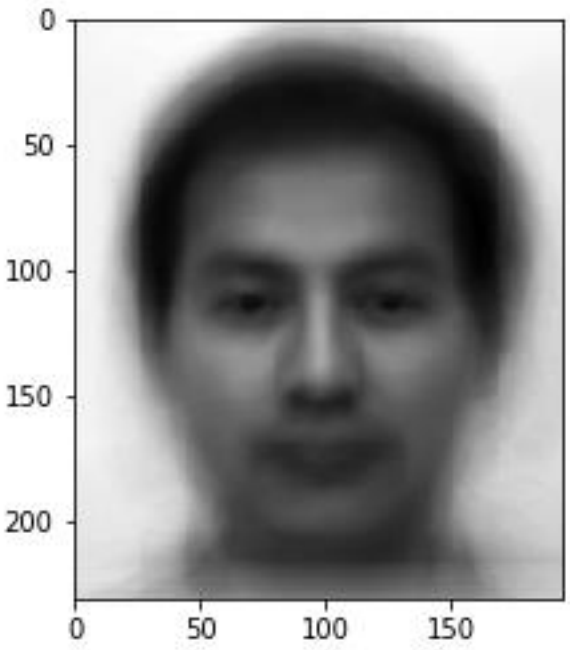


Figure 10: Mean face of training set (including personal pictures)

    c. **Find faster way to compute eigen values and implement**

The improved manner to compute eigen values is to avoid computing the correlation matrix and instead compute a different matrix with the same eigenvalues [2]. Instead compute the eigenvectors of:

$$A^T A$$

This matrix is MxM where M is the number of training images. This produces eigen vectors v_i and the relationship of these eigenvectors to the eigenvectors of the convolution matrix is:

$$A^T A v_i = u v_i$$
$$u_i = A v_i$$

Therefore, by implementing this method of calculating the top eigenvectors one is able to avoid the large correlation matrix and instead generate a faster process of finding eigenvectors.

    d. **Determine top K principle components**

K was set to 15 for the purposes of this report. It was observed that lower values of K had less accuracy while greater values were slightly more accurate. Past k=15 increasing the value low marginal return.

e. For each 10 images, find corresponding 3 most similar faces and present next to the image in report.

All faces matched correctly except for 2 notable cases. Figure 15 shows that the 2nd and 3rd match with subject 5 are not correct. Match 1 of subject 8 (figure 18) also return an incorrect result. The accuracy of the program given this sample is approximately 91% (including personal photos). This is relatively good especially considering that at least 1 correct response was given for each case. This accuracy could be improved by either expanding the training set and/or increasing the number k of eigenvectors used.

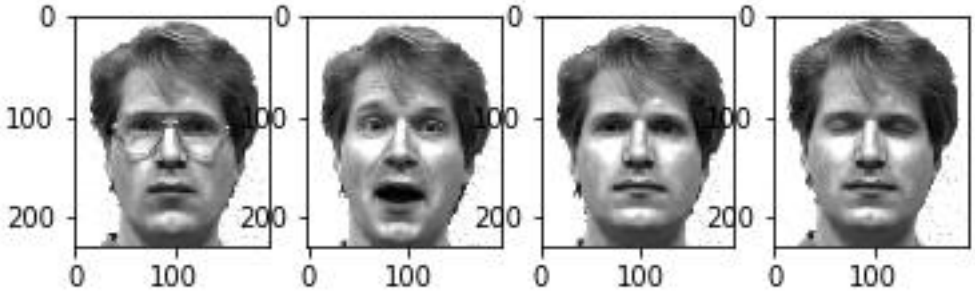


Figure 11: Test face 1 with top 3 matches

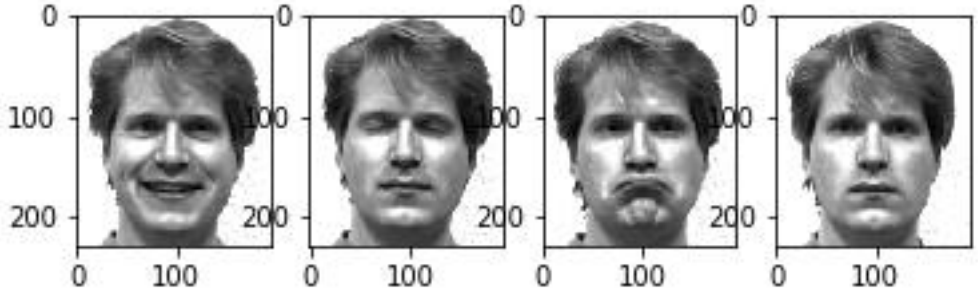


Figure 12: Test face 2 with top 3 matches

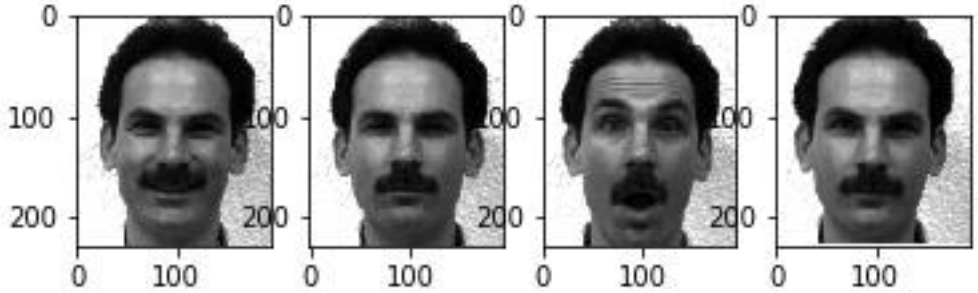


Figure 13: Test face 3 with top 3 matches

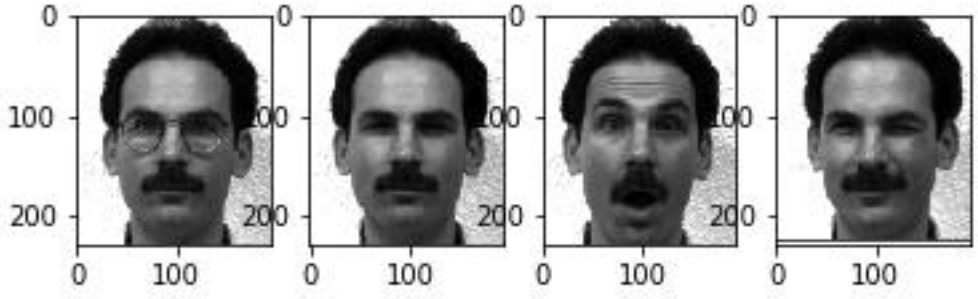


Figure 14: Test face 4 with top 3 matches

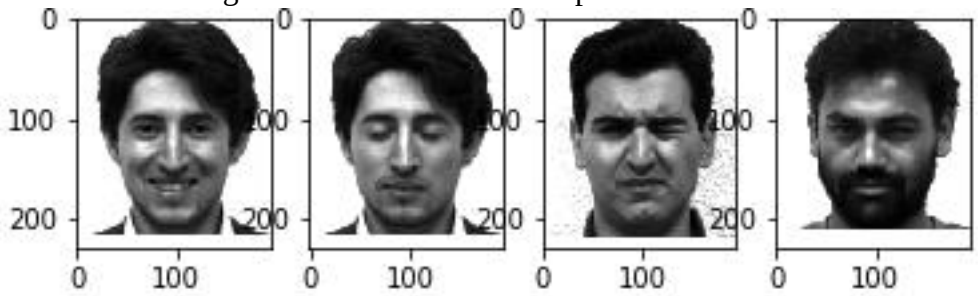


Figure 15: Test face 5 with top 3 matches

Figure 16: Test face 6 with top 3 matches

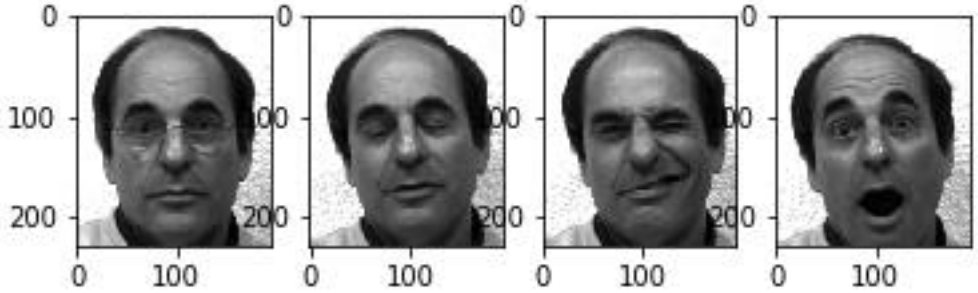


Figure 17: Test face 7 with top 3 matches

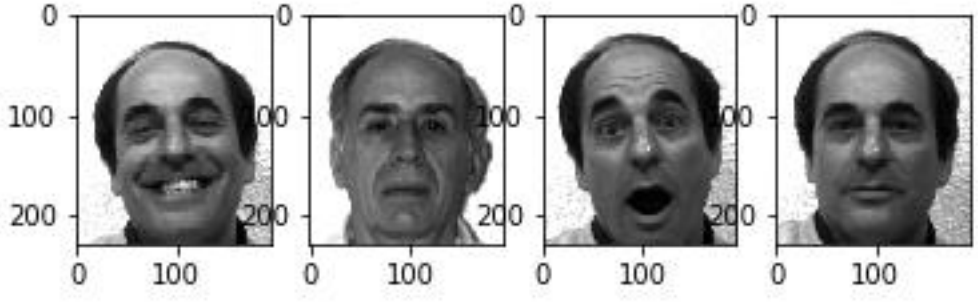


Figure 18: Test face 8 with top 3 matches

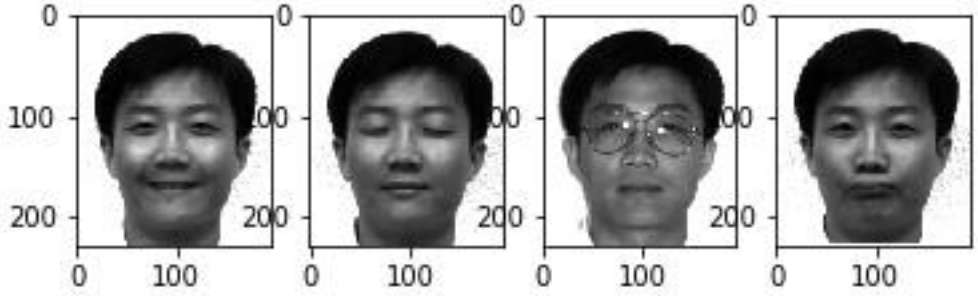


Figure 19: Test face 9 with top 3 matches

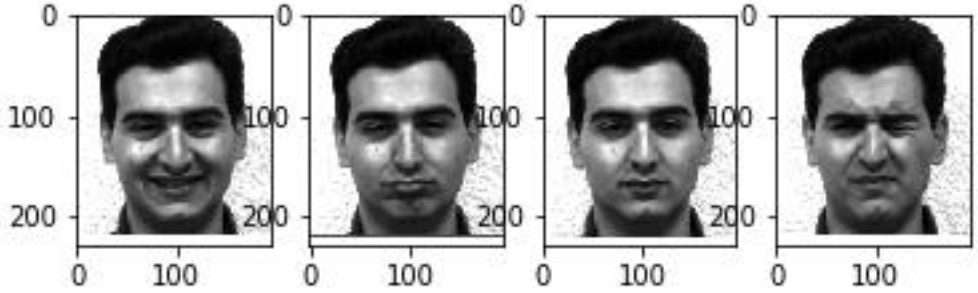


Figure 20: Test face 10 with top 3 matches

f. Repeat including personal picture

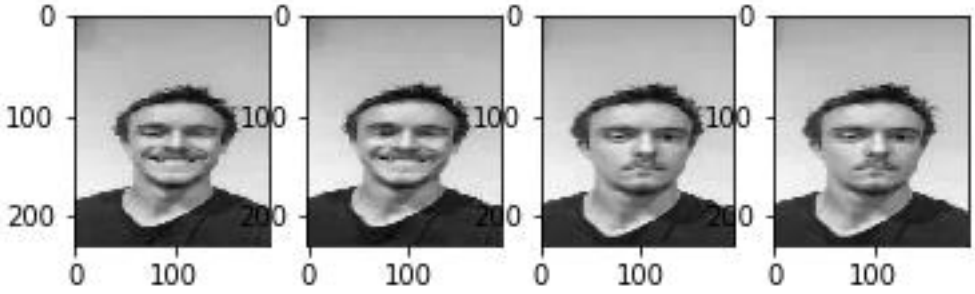


Figure 21: Personal photo with top 3 matches from training set

It is perhaps not surprising that the personal photos match given they are slightly different in nature to the original images. The difference in background likely is significant enough to affect the matching process.

Resources:

[1] N. Barnes, "ENGN4528: Feature_Points, Segmentation, Eigenfaces", 2020.

[2] M. Turk and A. Pentland, "Eigenfaces for Face Detection/Recognition", Vision.jhu.edu, 1991. [Online]. Available: http://www.vision.jhu.edu/teaching/vision08/Handouts/case_study_pca1.pdf. [Accessed: 17- May- 2020].

[3] D. Arthur and S. Vassilvitski, Ilpubs.stanford.edu, 2020. [Online]. Available: http://ilpubs.stanford.edu:8090/778/1/2006-13.pdf. [Accessed: 17- May- 2020].

Other Coding resources

[4]“Harris Corner Detection — OpenCV-Python Tutorials 1 documentation", Opencv-python-tutroals.readthedocs.io, 2020. [Online]. Available: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_features_harris/py_features_harris.html. [Accessed: 17- May- 2020].

[5] B. script, "Best way to load image from list python script", *Blender Stack Exchange*, 2020. [Online]. Available: https://blender.stackexchange.com/questions/135956/best-way-to-load-image-from-list-python-script. [Accessed: 17- May- 2020].