CS-512: Computer Vision- FALL 2018

ASSIGMENT 3

Diego Martin Crespo

A20432558

1. PROBLEM STATEMENT

The problem chosen to solve in this assignment is corner detection.

1.1. LOAD AND DISPLAY TWO IMAGES CONTAINING SIMILAR CONTENT. IN EACH IMAGE PERFORM:

- Estimate image gradients and apply the Harris corner detection algorithm.
- Obtain a better localization of each corner.
- Compute a feature vector for each corner point.
- Display the corners by drawing empty rectangles over the original image centered at locations where corners where detected.

1.2. USING THE FEATURE VECTORS, YOU COMPUTED MATCH THE FEATURE POINTS. NUMBER CORRESPONDING POINTS WITH IDENTICAL NUMBERS IN THE TWO IMAGES.

1.3. INTERACTIVELY CONTROLLED PARAMETERS SHOULD INCLUDE: THE VARIANCE OF THE GAUSSIAN (SCALE), THE NEIGHBORHOOD SIZE FOR COMPUTING THE CORRELATION MATRIX, THE WEIGHT OF THE TRACE IN THE HARRIS CORNER DETECTOR AND A THRESHOLD VALUE.

2. PROPOSED SOLUTION

2.1.

First of all, two images are loaded and converted to grayscale but keeping three channels so we can add colors in the future.

Second, a Sobel filter is applied to both x and y axis for calculate the gradient. The values for Ix*Ix, Iy*Iy and Ix*Iy are computed for each pixel in the image. These values are used to calculate the correlation matrix for each window of the image.

Third, once the correlation matrix is computed. Its determinant and trace are calculated and used to compute the C(G) values for each pixel which are stored in list of lists along with each pixel position.

Fourth, the maximum value for C(G) in the list is found to be multiplied by a constant to use it as threshold to find the corners.

Finally, the pixels selected as corners are stored in a new list and then the rectangles are drawn around those pixels.

2.2.

A brute-force matching with ORB descriptors using SIFT is used, in other to compute the feature vectors

This method calculates the feature vectors for both images. It matches similar pixels depending on the distance from both images.

They are sorted with those with less distance. And the smaller ones are drawn in the images in form of lines that connect the similar pixels.

2.3.

The parameters are controlled through track bars in the image (“k”, “Threshold” and “Window_Size”). The variance of the Gaussian it is applied when the Sobel filter is computed. Therefore, it is not controlled from the otherwise of the program.

3. IMPLEMENTATION DETAILS

For this implementation I did reused some of the code from previous homework, so some of the functions where pretty essay to solve. For instance, the loading stage, conversion to gray or Sobel filter computation.

I did mention before that I did not control the gaussian smoothing because I used a Sobel filter on the image, which already has a Gaussian in it. I could have implemented my own as I did in the previous assignment but the program will take much more time to compute. So finally, I decided not to implemented that way.

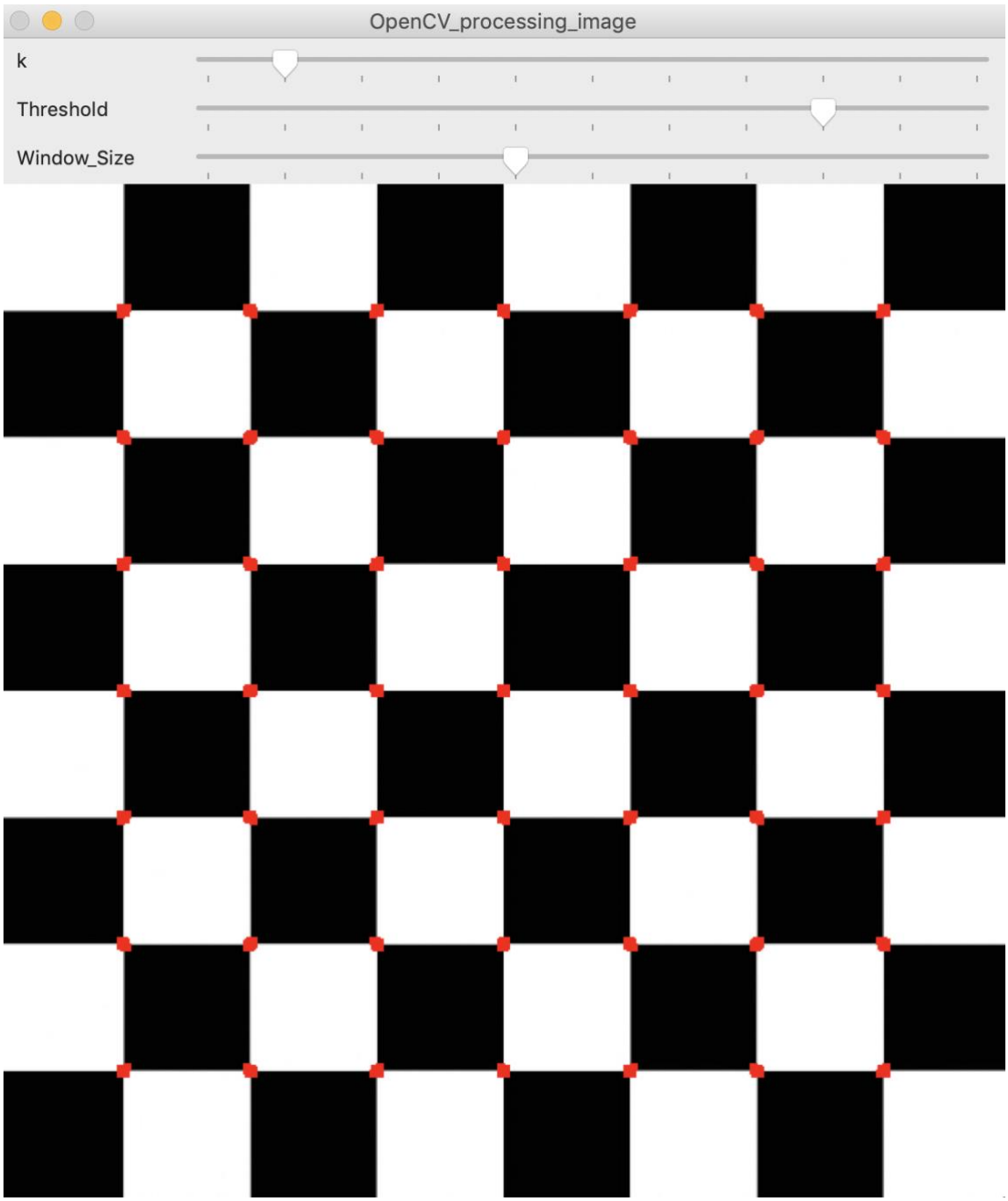
Finally, to compute the feature vectors I used the OpenCV library. I did not manage to do it by myself, with my own implementation

4. RESULTS AND DISCUSSION

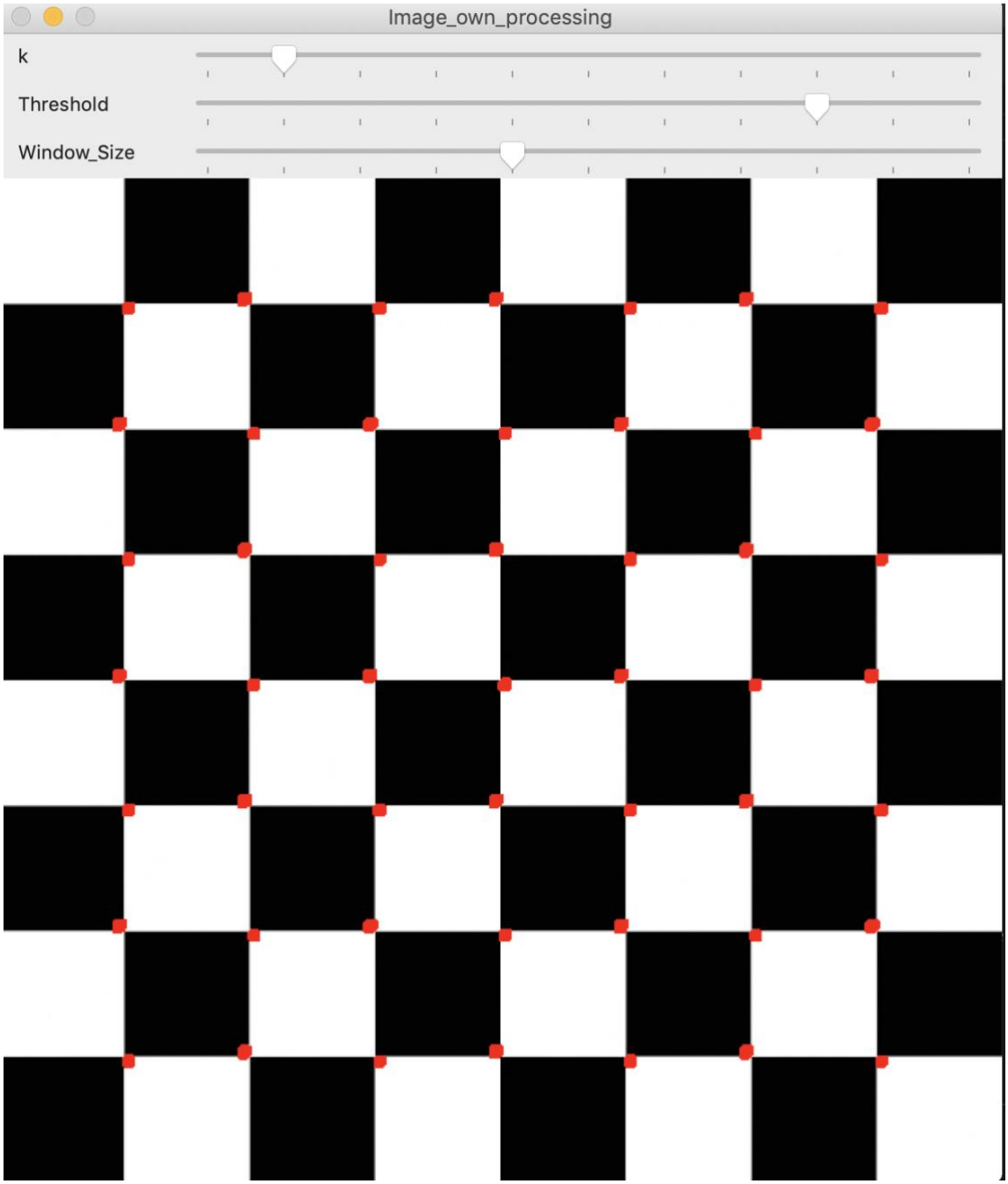
At the beginning I started computing the corners with my algorithm for several images. Also I compared my results with the Harris corner detection OpenCV function to see how was the performance of my function.

Here are some examples:

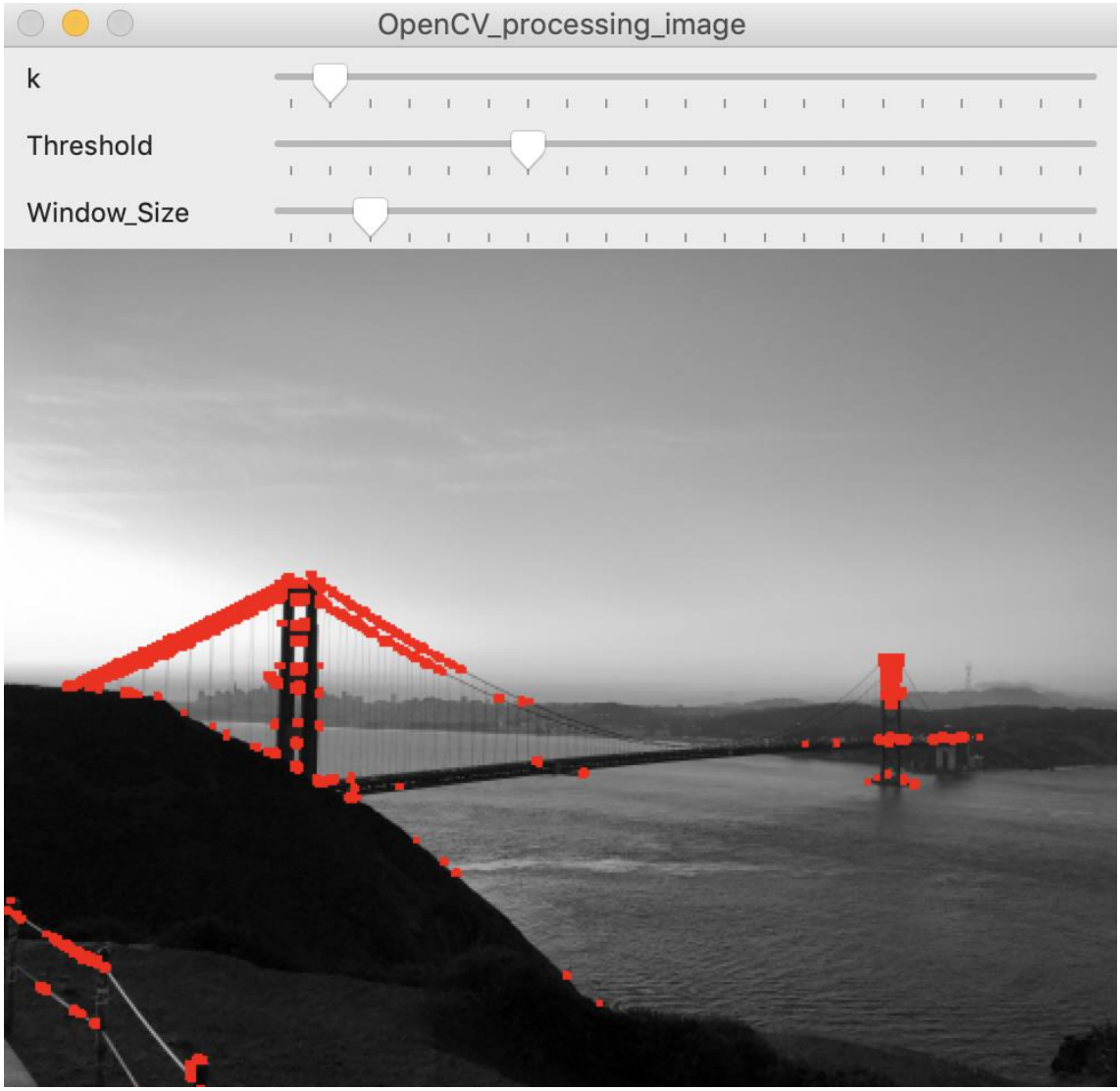
USING OPEN_CV



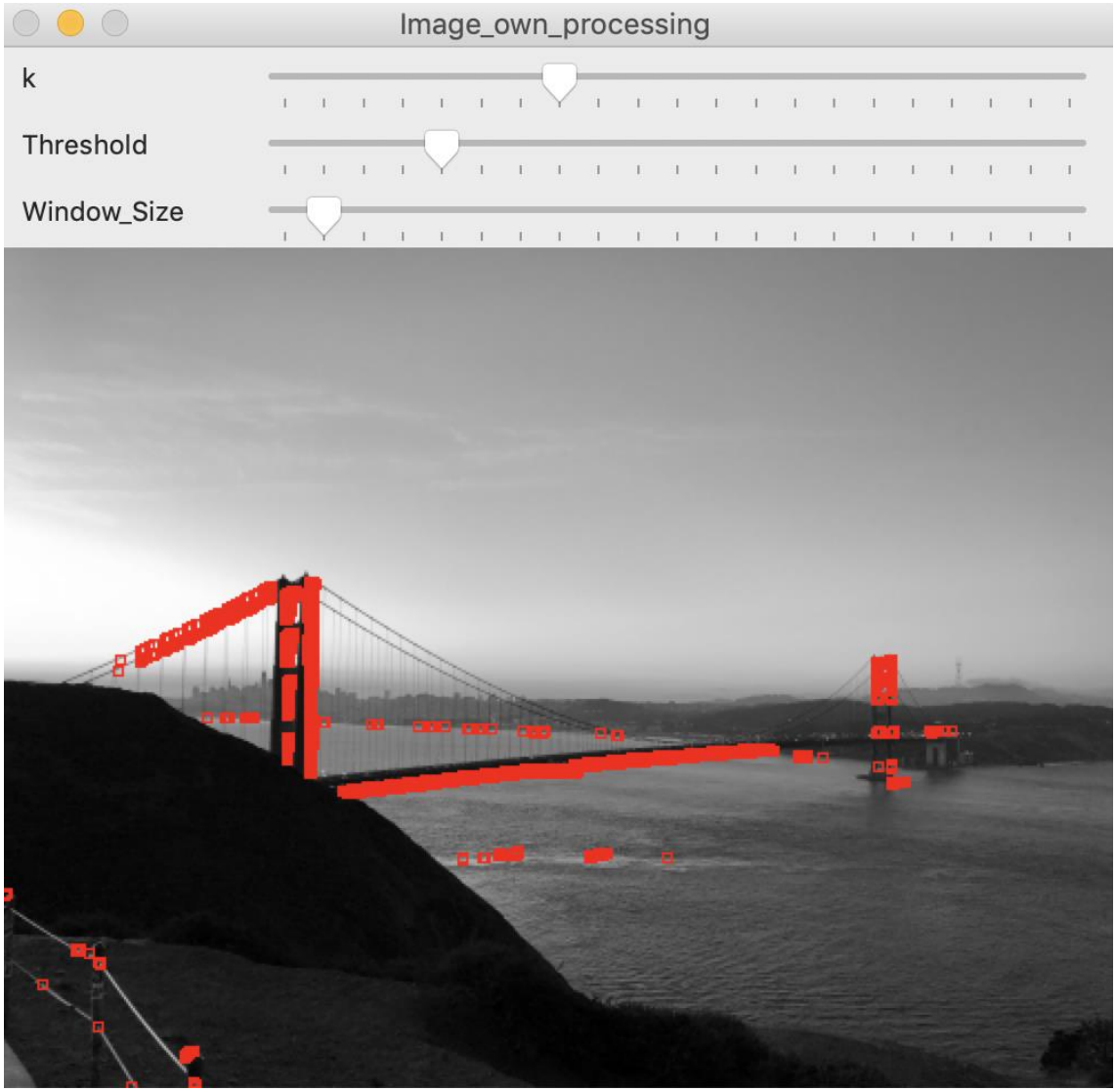
USING OWN IMPLEMENTATION


Image_own_processing
k
Threshold
Window_Size

USING OPEN_CV



USING OWN IMPLEMENTATION



As it can be inferred from the previous images my algorithm does perform quite similar to the OpenCV implementation. However, there is to say that it takes much more time to make the calculations. This is due to the double “for” loops from my algorithm. OpenCV is much more optimized in this aspect.

On the other hand, I notice that the values from k, Threshold and Window_size form my implementation do not correspond to the same ones used by OpenCV. For instance, the threshold used in OpenCV is much smaller than my implementation.

After checking that my algorithm (own implementation of corner detection) worked I went on and compute the corners and the feature vectors for two similar images and displayed them in the same window.

The following figures shows the result:

FEATURES USING OWN IMPLEMENTATION

As the feature vectors are computed using OpenCV and the cornet detection using my own implementation, not all the corners correspond to all the features vectors. This is because what I mention before, that the OpenCV implantation computes the corners quite different from my implementation.

Overall my algorithm implementation does perform quite similar to OpenCV functions. However, as it takes more time to compute that OpenCV functions. I would recommend to use the OpenCV implementation.

5. REFERENCES

https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_features_harris/py_features_harris.html

https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.concatenate.html

https://docs.opencv.org/3.0-beta/modules/imgproc/doc/feature_detection.html

https://docs.opencv.org/2.4/modules/highgui/doc/user_interface.html

https://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html

https://pythonprogramming.net/feature-matching-homography-python-opencv-tutorial/

http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html

https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_matcher/py_matcher.html