*Assignment 02*

CS412 – Computer Vision

*1351052 – Truong Minh Bao*

*Advanced Program in Computer Science*

1. **ASSIGNMENT DESCRIPTION**

* Languages & Libraries: Python, OpenCV 3
* Setup:
  + Make sure that you have python installed.
  + OpenCV dlls should be available in python library path.
* Usage:
  + To run the program:   
    E.g.:  
    python main.py harris lena.jpg
* Command line arguments:
  + harris image.jpg - detect key points using harris algorithm and show the keypoints in original image.
  + blob image.jpg - detect key points using blob algorithm and show the keypoints in original image.
  + dog image.jpg - detect key points using DoG Algorithm and show keypoints in original image.
  + m harris sift image1.jpg image2.jpg - match and show results of image1 and image2 using Harris detector and SIFT descriptor.
  + m dog sift image1.jpg image2.jpg - match and show results of image1 and image2 using DoG detector and SIFT descriptor.
  + m blob sift image1.jpg image2.jpg - match and show results of image1 and image2 using using Blob detector and SIFT descriptor.
  + m harris lbp image1.jpg image2.jpg - match and show results of image1 and image2 using Harris detector and LBP descriptor.
  + m dog lbp image1.jpg image2.jpg - match and show results of image1 and image2 using DoG detector and LBP descriptor.
  + m blob lbp image1.jpg image2.jpg - match and show results of image1 and image2 using Blob detector and LBP descriptor.
  + h - Display a short description of the program, its command line arguments, and the keys it supports.

1. **FUNCTIONS IMPLEMENTED**
2. **Source code overview:**

* descriptors.py: A function that helps the program create an instance of SIFT/LBP descriptor.
* detectors.py: A function that helps the program create an instance of Harris/BLOB/DOG detector.
* harris.py: Wrap the built-in harris corner to produce OpenCV’s Keypoint instead of a harris-response matrix.
* lbp.py: An implementation of LBP descriptor.
* main.py: The main program.
* settings.py: All necessary parameters for the above detector.

1. **Key point detection with Harris’ algorithm:**

In order to detect key point using algorithm, I make use of . All the parameters for the function are available in . Since Harris’ algorithm does not give us any information about the size of a key point, I choose to give them the radius of 1. Additionally, quality level which is the minimum ration of a key point that should be chosen and the maximum response can be adjusted with a track bar.

More specifically, Harris’ algorithm helps us compute the response matrix based on the following equation [[\*]](http://docs.opencv.org/2.4/doc/tutorials/features2d/trackingmotion/harris_detector/harris_detector.html):

Where

Where and are image derivatives in x and y directions, respectively.

After computing M, the algorithm determines if a window can contain a kernel or not using the response value computed as follow:

Where

* and are the Eigen values of

Though key points detected using Harris’ algorithm are invariant to rotation and illumination, they vary when the input image’s scale or view point is changed.

1. **Key point detection with BLOB algorithm:**

This method uses . All the parameters for creating an instance of this class can be found in . I also create a track bar to adjust the maximum area of a single blob detected by the algorithm.

BLOB detection algorithm involves the following steps [[\*]](https://www.learnopencv.com/blob-detection-using-opencv-python-c/):

1. **Thresholding**: Convert the input image into several binary images by thresholding the source image with thresholds starting at minThreshold. These thresholds are incremented by thresholdStep until maxThreshold. So the first threshold is minThreshold, the second one is minThreshold + thresholdStep, the third one is minThreshold + 2 \* thresholdStep, and so on.
2. **Grouping**: In each binary image, connected white pixels are grouped together. Let’s call these binary blobs.
3. **Merging**: The centers of binary blobs in the binary images are computed, and blobs located closer than minDistBetweenBlobs are merged.
4. **Center & Radius Calculation**: The centers and radii of the new merged blobs are computed and returned.

Key points detected by BLOB detection algorithm are invariant to rotation, illumination, rotation and scaling but they vary to view point changes.

1. **Key point detection and description with SIFT algorithm:**

As SIFT algorithm uses Difference of Gaussians to detect key points. I also added a track bar to adjust the number of octave layers.

The SIFT algorithm consists of the following steps [[\*]](http://docs.opencv.org/3.1.0/da/df5/tutorial_py_sift_intro.html):

1. **Scale-space Extrema Detection:** Uses Difference of Gaussians to approximate Laplacian of Gaussian. Once this DoG are found, image is searched for local extrema over scale and space. One pixel in an image is compared with its 8 neighbors as well as 9 pixels in next scale and 9 pixels in previous scales. If it is a local extremum, it is a potential key point.
2. **Key point Localization:** Taylor series expansion of scale space is used to get more accurate location of extrema, and if the intensity at these extrema is less than a threshold value (0.03 as per the paper), it is rejected. This threshold is called contrastThreshold in OpenCV. Also, edges are removed with edgeThreshold.
3. **Orientation Assignment (for invariance to rotation):** A neighborhood is taken around the keypoint location depending on the scale, and the gradient magnitude and direction is calculated in that region. An orientation histogram with 36 bins covering 360 degrees is created. The highest peak in the histogram is taken and any peak above 80% of it is also considered to calculate the orientation.
4. **Key point Descriptor:** A 16x16 neighborhood around the key point is taken. It is divided into 16 sub-blocs of 4x4 size. For each sub-block, 8 bin orientation histogram is created. So a total of 128 bin values are available. It is represented as a vector to form key point descriptor.

SIFT algorithm has robustness against changes in both illumination, rotation, scaling, and minor changes in view point.

1. **Key point description with LBP:**

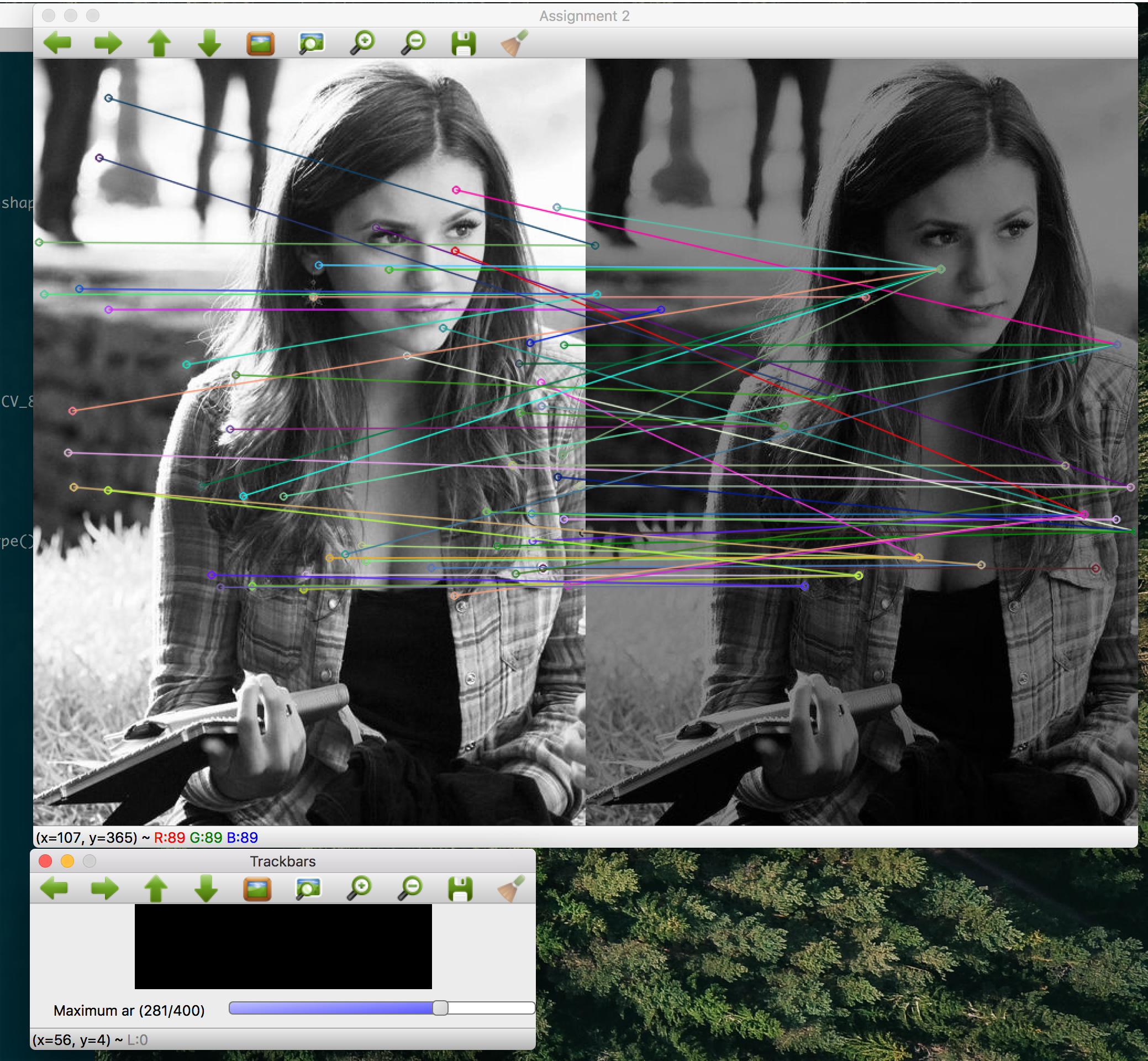
Local binary patterns (LBP) is a very simple and efficient feature which labels pixels of an image by thresholding their neighborhoods.

LBP descriptor is computed as follows [[\*]](http://hanzratech.in/2015/05/30/local-binary-patterns.html):

1. Divided the examined window into cells (e.g. 16x16 pixels for each cell). Here, we look at the 16x16 neighborhood around each key point.
2. For each pixel in a cell, compare the pixel to each of its 8 neighbors (on its left-top, left-middle, left-bottom, right-top, etc.). Follow the pixels along a circle, i.e. clockwise or counter-clockwise.
3. Where the center pixel’s value is greater than the neighbor’s one, write “0”. Otherwise, write “1”. This gives an 8-bit binary number.
4. Computer the histogram, over the cell, of the frequency of each “number” occurring. This histogram can be seen as a 256-dimensional feature vector.
5. **Key point matching:**

In order to carry out matching, I use K-Nearest Neighbor Matching implementation of on feature descriptors extracted from 2 input images.

1. **SOME PROGRAM SCREENSHOTS**

****