CS412 - Homework 2 - Object Detection

Khac Le Duy CAO - 1351008

December 11, 2016

1 Note

This should be attached to the homework 2 project of CS412 course. This report will list out some main features and the techniques used in the project

2 Features and Usage

2.0.1 Prerequisites

- Python 2.7 installed, together with Numpy library.
- OpenCV 3.1.0 + OpenCV Contribute (this one is needed for xfeatures2d package), these can be cloned from github of OpenCV
- You may want to use your own data images which must be added to folder data in this project root directory

2.0.2 Descriptions

This small program can be used by using this command in the root directory of the project:

$ python main.py -i [options]

The options are the commands for each functionalities:

- Detect key points using harris algorithm and show the keypoints in original image.
  harris [image_path]
- Detect key points using blob algorithm and show the keypoints in original image.
  blob [image_path]
- Detect key points using DoG Algorithm and show keypoints in original image.
  dog [image_path]

- Match and show results of image1 and image2 using Harris detector and SIFT descriptor.
  harris sift [image_path1] [image_path2]
- Match and show results of image1 and image2 using DoG detector and SIFT descriptor.
  dog sift [image_path1] [image_path2]
- Match and show results of image1 and image2 using using Blob detector and SIFT descriptor.
  blob sift [image_path1] [image_path2]
- Match and show results of image1 and image2 using Harris detector and LBP descriptor.
  harris lbp [image_path1] [image_path2]
- Match and show results of image1 and image2 using DoG detector and LBP descriptor.
  dog lbp [image_path1] [image_path2]
- Match and show results of image1 and image2 using Blob detector and LBP descriptor.
  blob lbp [image_path1] [image_path2]

These descriptions are also displayed when the users press 'h' when the program windows are active.

3 Theory and Implementation

3.1 Harris Detector

This algorithm is based on the change of intensity when shifting the examined window with a specific size to another position having the relative coordinates [u,v] to the previous position:

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u,y+v) - I(x,y)]^2 \tag{1}$$

The step by step algorithm is described as follows:
First, compute x and y derivatives of the image

$$I_x = G_\sigma^x * I; I_y = G_\sigma^y * I \tag{2}$$

Second, at each pixel, find the products of derivatives:

$$I_{x2} = I_x.I_x; I_{y2} = I_y.I_y; I_{xy} = I_x.I_y; \tag{3}$$

Third, Get the sums of products of derivatives at each pixel:

$$S_{x2} = G_{\sigma'} * I_{x2}; S_{y2} = G_{\sigma'}.I_{y2}; S_{xy} = G_{\sigma'} * I_{xy}; \tag{4}$$

Fourth, for each pixel, get a matrix:

$$H(x,y) = \begin{bmatrix} S_{x2}(x,y) & S_{xy}(x,y) \\ S_{xy}(x,y) & S_{y2}(x,y) \end{bmatrix} \tag{5}$$

Finally, compute the response of detector at each pixel, which is used for examining the threshold that determine if a point is a key point or not.

$$R = Det(H) - k(Trace(H))^2 \tag{6}$$

Due to the complication in implementing this algorithm, the author used the built-in function of OpenCV called *cornerHarris* to get the keypoints.

3.2 DoG Detector

This method is based on the difference among the results of convolution between the image and the multiple-size Gaussian kernels. In this project we consider 8 Gaussian kernel sizes (3, 5, 7, 9, 11, 13, 15). After executing convolution, the author get 7 matrices from the differences of result matrices. Then the author make comparisions of each pixel to its local 8 neighbors and itself in the above and beneath matrix layers. The pixels that has value greater than 8 neighbors and their above and beneath values are the keypoints.

This is a self-implemented algorithm

3.3 BLOB Detector

The basic concept of blob detection:

First, the image under consideration is convolved with the scale-normalized Laplacian at several scales

Second, the maxima of squared Laplacian response in scale-space is computed. This value indicate if a blob has been detected and what is its intrinsic scale.

Laplacian is the second Gaussian derivative:

$$\nabla^2 g = \frac{\delta^2 g}{\delta x^2} + \frac{\delta^2 g}{\delta y^2} \tag{7}$$

Scale-normalized:

$$\nabla^2_{norm} g = \sigma^2 (\frac{\delta^2 g}{\delta x^2} + \frac{\delta^2 g}{\delta y^2}) \tag{8}$$

In this project, the author used the instance of *SimpleBlobDetector* from OpenCV for simple implementation.

3.4 SIFT Descriptors Computing

For explanation, descriptors provides invariance to changes in illumination and 3D camera viewpoint. It is calculated for a region around a key point as opposed

to directly to the key point. Magnitudes and orientations are also calculated for points in that region using L of nearest scale. to ensure orientation invariance the gradient orientations and coordinates of descriptor are rotated relatively to orientation of key point.

Magnitude of each point is weighted with σ of $\frac{1}{2}$ the width of the descriptor window.

The result descriptor is a 128-dimensional feature vector built from considering each 4x4 cell with 8 orientations around each key point.

In this project, the author used the SIFT instance from xfeatures2d in opencv_contrib repository for simpler implementation.

3.5 LBP Descriptors Computing

This method is somehow similar to SIFT in object it considers, but the later manipulation and the properties of these objects are different.

In this method, a cell of size 16x16 around each key point is put into calculating. Each cell will produce a feature vector of 256 dimensions. In each cell, each pixel and its 8 neighbors (a 3x3 area of pixels) are put in comparisons. The pixel at the center is compared with its neighbors according to the clock-wise (or anti clock-wise) direction and a bit of 1 or 0 will be written down from left to write, respectively. 1 is marked when the center is less intensive than the corresponding neighbor, and 0 for the greater intensity of the center to the corresponding neighbor.

From the above process, an 8-bit integer is extracted. After extract an 8-bit integer for each pixel in the cell. An 256-dimensional array is created. There is 256 integers from 0 to 255 can be extracted, so each element in this array will hold the frequency of the integers in a cell. This array is officially the descriptor of the corresponding key point.

This is a self-implemented algorithm.

4 Project Structure

4.1 Main

This is where the make-up functions are placed. There is a parser called mapping_function to map the commands to the correct manipulation.

4.2 Detector

There three main detectors inherited from a parent named Detector:

- Harris Detector
- Blob Detector
- DoG Detector

4.2.1 Matcher

As Detectors, there are aslo 2 matchers inherited from the parent named Matcher, the matcher is held in the detector for convenience when computing keypoints with different detecting method:

- SIFT Matcher (using SIFT to get descriptor)
- LBP Matcher (using LBP to get descriptor)

There is a supportive class called LBPDescriptorExtractor help the descriptor extraction following LBP Algorithm.

5 Statistics

Time measurement in seconds for detecting key points:

Harris	Blob	DoG
0.0813	0.18	0.77

Time measurement in seconds for extracting descriptors with Harris Keypoint Detector $k_{freevalue} = 0.24$

SIFT	LBP
2.01	4.77

6 Samples

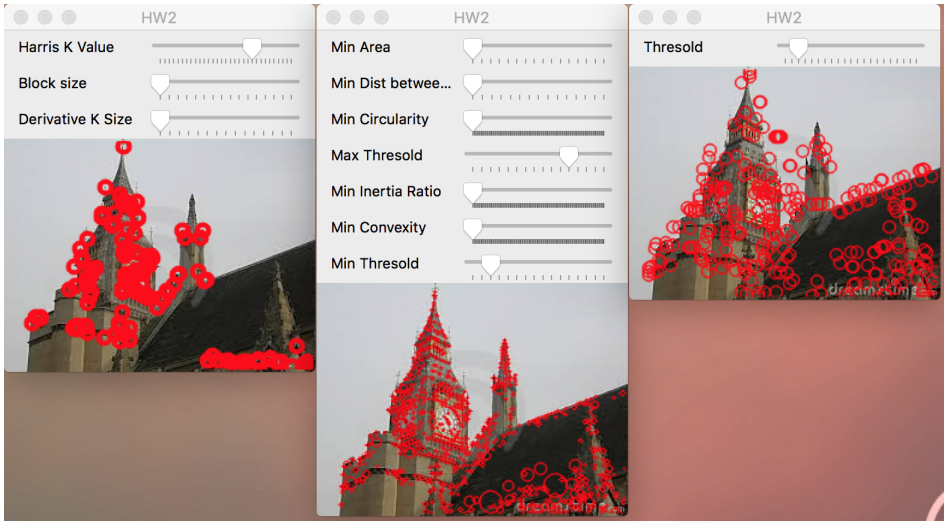

Figure 1: Detecting Keypoints by Harris - Blob - DoG

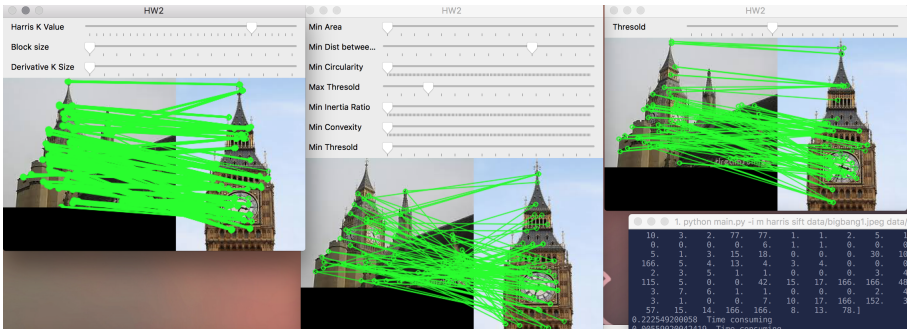

Figure 2: SIFT Extract Descriptors from keypoints from Harris - Blob - DoG

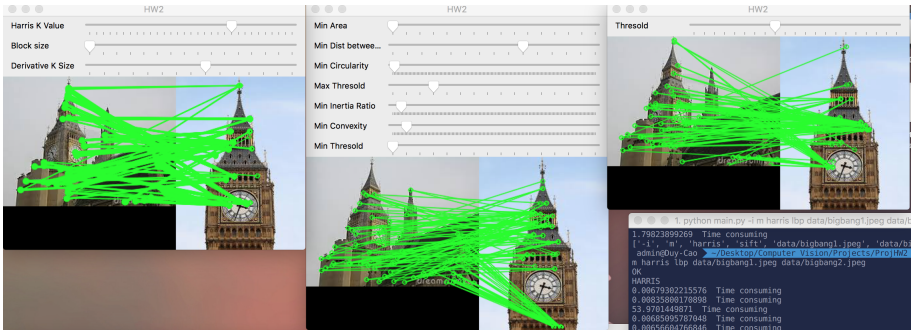

Figure 3: LBP Extract Descriptors from keypoints from Harris - Blob - DoG