# Department of Computing

**CS471: Machine Learning**

**Class: BESE-7AB**

**Lab 12: Computer Vision Basics in Python**

**C5: Use modern tools to solve practical problems.**

**Date: 10-05-2019**

**Time: 10:00 am– 12:00 pm & 2:00 pm-5:00 pm**

**Instructor: Dr. Pakeeza Akram**

**Lab 12: Computer Vision Basics in Python**

**Introduction**

Computer vision aims to artificially replicate the human visual perceptions. It deals with the science and technology of processes related to the acquisition of images, analysis of images and sequence of images to extract useful information, and to the development of artificial cognitive systems that “see.” Computer Vision is a part of the artificial intelligence field or the computer science field in general.

**Objectives**

In this lab you will be introduced to computer vision programming using Python.

**Tools/Software Requirement**

PIL and OpenCV python package

**Lab Tasks**

Python Imaging Library (PIL) provides general image handling and lots of useful basic image operations like resizing, cropping, rotating, color conversion and much more. See the documentation at <http://effbot.org/imagingbook/pil-index.htm>. It comes preinstalled in Canopy and Anaconda python distribution.

**Task 1**

Read any color image using PIL. Resize the image, rotate it 45 degrees and convert it to gray scale as shown in the following images. The following line reads and displays the image.

from PIL import Image

im = Image.open(‘image location’)

im.show()

#Your task is to resize, rotate and convert to grayscale.



NumPy (http://www.scipy.org/NumPy/) is a package popularly used for scientific computing with Python. NumPy contains a number of useful concepts such as array objects (for representing vectors, matrices, images and much more) and linear algebra functions. The array object let’s you do important operations such as matrix multiplication, transposition, solving equation systems, vector multiplication, and normalization, which are needed to do things like aligning images, warping images, modeling variations, classifying images, grouping images, and so on. This comes installed with Canopy.

**Task 2**

Use numpy to convert colored image to gray scale. You cannot use the built in function for this task. Use the following code snippet as your starting point.

Import numpy as np

a = np.array(im) # this will convert the PIL.Image object to numpy array representation

a.shape # this will print out the shape of the array.

# Now convert this to gray scale

Matplotlib is python library for visualization. It is also pre-installed in Canopy and Anaconda.

**Task 3**

1. Explain what is histogram of an image?
2. Using Matplotlib to display histogram of an image. (You will first need to convert the image to grayscale and then convert it to array representation)
3. What is histogram equalization?
4. Implement a function for histogram equalization.

**Task 4**

For many of the computer vision tasks such as object recognition, image panoramas etc., the first step is local features extraction from images. The purpose of local features is to provide a representation that allows to efficiently match local structures between images. That is, we want to obtain a sparse set of local measurements that capture the essence of the underlying input images and that encode their interesting structure. To meet this goal, the feature extractors must fulfill two important criteria:

* The feature extraction process should be repeatable and precise, so that the same features are extracted on two images showing the same object.
* At the same time, the features should be distinctive, so that different image structures can be told apart from each other.

Please go through local\_features\_synthesis and featuresDescriptors concepts on your own.

The first step of the local feature extraction pipeline is to find a set of distinctive keypoints that can be reliably localized under varying imaging conditions, viewpoint changes, and in the presence of noise. In particular, the extraction procedure should yield the same feature locations if the input image is translated or rotated. It is obvious that those criteria cannot be met for all image points. For instance, if we consider a point lying in a uniform region, we cannot determine its exact motion, since we cannot distinguish the point from its neighbors. Similarly, if we consider a point on a straight line, we can only measure its motion perpendicular to the line. This motivates us to focus on a particular subset of points, namely those exhibiting signal changes in two directions. You will use two keypoint detectors that employ different criteria for finding such regions: the Hessian detector and the Harris detector. Use OpenCV to compute Hessian and Harris detector on graf/img1.ppm graf/img4.ppm. **OpenCV is not pre-installed. You need to install it using Canopy package manager**. The code for the case of Harris detector is shown below:

**import** cv2

**import** numpy **as** np

img **=** cv2**.**imread(‘graf/img1.ppm’)

gray **=** cv2**.**cvtColor(img,cv2**.**COLOR\_BGR2GRAY)

gray **=** np**.**float32(gray)

dst **=** cv2**.**cornerHarris(gray,2,3,0.04)

img[dst**>**0.01**\***dst**.**max()]**=**[0,0,255]

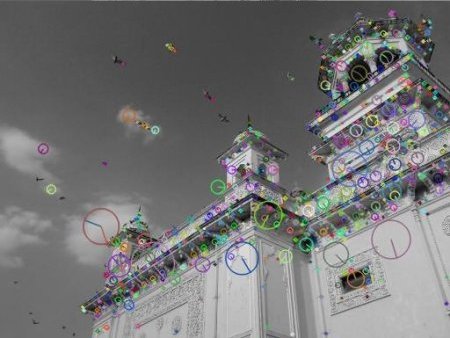
cv2**.**imshow('dst',img)

**if** cv2**.**waitKey(0) **&** 0xff **==** 27:

cv2**.**destroyAllWindows()

**Task 5**

While shown to be remarkably robust to image plane rotations, illumination changes, and noise, the locations returned by the Harris and Hessian detectors are only repeatable up to relatively small scale changes. The reason for this is that both detectors rely on Gaussian derivatives computed at a certain fixed base scale σ. If the image scale differs too much between the test images, then the extracted structures will also be different. For scale invariant feature extraction, it is thus necessary to detect structures that can be reliably extracted under scale changes. In the next task you will use Scale Invariant Feature Transform (SIFT) <http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html#sift-intro> to detect keypoints. SIFT not only detects keypoints but it also computes descriptors. Descriptor encodes the image structure in spatial neighbourhood. Use SIFT to detect and compute the descriptors on images img1.ppm and img4.ppm. You output should look like the one below but on images img1.ppm and img4.ppm.



**Task 6**

Match the keypoints detected using SIFT across two images (img1.ppm and img4.ppm). Your output should look like this but on img1.ppm and img2.ppm.

****

**Deliverables**

Upload Jupiter file along with Word file containing output (figures) and your code.