EL-GY 6123 Image and Video Processing, Fall 2017

Programming Assignment 3 (Due 3/20/2017)

Note that you can google each of the opencv functions suggested below to find how to use them properly and what parameters you can set and what are their meainings.

1) Harris detector
   a) Write your own program for Harris corner point detection at a fixed scale. Your program can contain the following steps:

      i) Generate gradient images of I_x and I_y using filters corresponding to derivative of Gaussian functions of a chosen scale σ and window size w (let us use w=4σ+1). You can use the conv2( ) function.
      ii) Compute three images $I_x * I_x$,$I_y * I_y$, $I_x * I_y$.
      iii) Smooth each image using a Gaussian filter with scale 2σ, and window size 6σ+1.
      iv) Generate an image of Harris cornerness value by forming the moment matrix A at every pixel based on the images from (c).
      v) Detect local maxima in the above image (for simplicity you could just check whether a center pixel is larger than its 8 neighbors). Pick the first N feature points with largest Harris values.
      vi) Mark each detected point using a small circle. You can use **cv2.circle()** function to draw circle. Display the image with the detected points.
      vii) You should write your own functions to generate Guassian and derivative of Gaussian filters from the analytical forms.

   b) Apply your Harris detector to a test image (you can just work on gray scale image). Using σ=1, N=50. Do the features detected make sense?
   c) Starting with the previous image, generate another two images with different rotations and another two images with different scalings (half and double) (**using cv2.resize( )**).

      For image rotation, you can first generate a rotation matrix by using OpenCV function **cv2.getRotationMatrix2D(center, angle, scale)**, and then apply this rotation matrix on the image by using OpenCV function **cv2.warpAffine(img, transform_matrix, output_size)**

   d) (d) Apply your Harris detector to these images. Do you detect almost the same set of feature points in each case? Why?

2) Write a program that can generate SIFT descriptor for each detected feature point using your program in Prob. 1. You may follow the following steps:
   a) Generate gradient images I_x and I_y as before. Furthermore, determine the gradient magnitude and orientation from I_x and I_y at every pixel.
   b) Quantize the orientation of each pixel to one of the N=8 bins. Suppose your original orientation is x. To quantize the entire range of 360 degree to 8 bins, the bin size is q=360/N=45 degree. Assuming your orientation is determined with a range of [0,360]. You can perform quantization using

(a) x_q = floor((x+q/2)/q); if x_q=N, x_q=0
(b) x_q will range from 0 to 7, with 0 corresponding to degree in [-22.5, 22.5), or [0, 22.5) and (360-22.5, 360).

c) Then for each detected feature point, you may follow the following steps to generate the SIFT descriptor:
   i) Generate a patch of size 16x16 centered at the detected feature point;
   ii) Multiply the gradient magnitude with a Guassian window with scale= patch width/2.
   iii) Generate a HoG for the entire patch using the weighted gradient magnitude.
   iv) Determine the dominant orientation of the patch by detecting the peak in the Hog determined in (e).
   v) Generate a HoG for each of the 4x4 cell in the 16x16 patch.
   vi) Shift each HoG so that the dominant orientation becomes the first bin.
   vii) Concatenate the HoG for all 16 cells into a single vector.
   viii) Normalize the vector. That is, divide each entry by L2 norm of the vector.
   ix) Clip the normalized vector so that entries >0.2 is set to 0.2.
   x) Renormalize the vector resulting from (k).

3) Finding corresponding points in two images based on SIFT descriptors.
   a) Using your program in Prob. 1 and Prob. 2 to detect feature points and generate their descriptors for two images you created in Prob. 1 (an original image and a rotated image).
   b) Write a program that can find matching points between the two images. For each point p in the first image, you compute its distance to each point in the second image (using Euclidean distance between two SIFT descriptors) to find two closest points q1 and q2 in the second image. Let us call the distance of p to q1 and q2 by d1 and d2, you will take point q1 as the matching point for p if d1/d2 <r. Otherwise, you assume there is ambiguity between q1 and q2 and skip the feature point p in image 1. You can experiment with different threshold r. Obviously r should be <1. At the end of this process, you should have a set of matching pairs.
   c) Create an image that shows the matching results. For example you can create a large image that has the left and right images side by side, and draw lines between matching pairs in these two images. Do the matched points look reasonable? You can use **cv2.line()** to draw line between each matching pair. You need to save and display the image after you add lines into the image array using the cv2.line() function.

4) Stitch two images into a panorama using SIFT feature detector and descriptor. (you can download sample image pairs from here)
   a) Read in an image pair (left and right)
   b) Detect SIFT points and extract SIFT features from each image by using the following OpenCV sample code.

```
detector = cv2.FeatureDetector_create("SIFT")
descriptor = cv2.DescriptorExtractor_create("SIFT")


skp = detector.detect(img)
skp, sd = descriptor.compute(img, skp)
```

Where skp is a list of all the key points found from img and sd is the descriptor for the image. Each element in skp is an OpenCV '**key points class**' object, and you can check the corresponding coordinate by **skp[element_index].pt** .

c) Mark the detected points in each image by a circle with radius equal to the scale of the feature. You can check the scale of a detected point using **skp[element_index].size**

d) Find the corresponding point pairs between left and right images based on their SIFT descriptors. You can reuse your program for Prob. 3.

e) Apply RANSAC method to these matching pairs to find the largest subset of matching pairs that are related by the same homography. You can use the function **cv2.findHomography(srcPoints, dstPoints, cv2.RANSAC)**

f) Create an image that shows the matching results by drawing lines between corresponding points. You can reuse the plotting function you developed for Prob. 3.

g) Apply the homography determined in (e) to the right image. You can use **cv2.warpPerspective()** to apply the homography transformation to the image.

h) Stitch the transformed image from (g) and the right image together to generate the panorama.

Hint: you should first create am array for the stitched image whose width is larger of the width of transformed left image and the right image. Similarly, the height is the larger of the heights of the transformed left image and the right image. Then, you put transformed left image into this array, and then the original right image into this array. With this simple approach, the part where the two images overlap will be represented by the right image.

In your report, show the left and right images, the left and right images with SIFT points indicated, the image that illustrates the matching line between corresponding points, the transformed left image, and finally the stitched image.