Introduction to Computer Vision (ECSE 415) Assignment 2

Due: February 25th, 11:59PM

Please submit your assignment solutions electronically via the myCourses assignment dropbox. The submission should include a single jupyter notebook. More details on the format of the submission can be found below. Submissions that do not follow the format will be penalized 10%. Attempt all parts of this assignment. The assignment will be graded out of total of **65 points**. You can use OpenCV and Numpy library functions for all parts of the assignment except stated otherwise. Students are expected to write their own code. (Academic integrity guidelines can be found at `https://www.mcgill.ca/students/srr/academicrights/integrity`). Assignments received up to 24 hours late will be penalized by 30%. Assignments received more than 24 hours late will not be graded.

1 Submission Instructions

1. Submit a single jupyter notebook consisting solution of the entire assignment.
2. Comment your code appropriately.
3. Do not forget to run Markdown cells.
4. Submit images collected for the first question.
5. Do not submit train/test images for the second question.
6. Do not submit output images. Output images should be displayed in the jupyter notebook itself. Assume input images are kept in a same directory as the codes.
7. Make sure that the submitted code is running without error. Add a README file if required.
8. If external libraries were used in your code please specify its name and version in the README file.

9. Answers to reasoning questions should be comprehensive but concise.
10. Submissions that do not follow the format will be penalized 10%.

Note that you can use OpenCV functions for this assignment, unless stated otherwise.

2 Image Stitching

In this question, you will learn to stitch images to create a panoramic view. Use your cellphone to collect three overlapping pictures of a same scene as shown in Figure 1. Next, follow these steps in order to stitch captured images:

- Compute **SIFT** keypoints and corresponding descriptors for left and central images. **(3 points)**
- Find matching keypoints in two images and display the 20 best pairs. **(4 points)**
- Find homography using RANSAC method. Implement RANSAC from scratch. Do not use OpenCV implementation. **(6 points)**
  - Steps to compute homography using RANSAC are given in Lecture 9 - slide 52.
  - Note on step 3: Compute H using opencv function `findHomography` with method=0.
  - Note on step 5: Before computing distance, convert projected keypoints from homogeneous to euclidean space using opencv function `convertPointsFromHomogeneous`.
- Apply transformation to left image. Central image should not be transformed. You are given a function `warpPerspectivePadded`. Use this function instead of OpenCV function `warpPerspective` for image transformation. **(2 points)**
- Stitch transformed left image and original central image together using **pyramid** image blending. **(6 points)**
- Display stitched image. **(1 point)**

Now let's stitch above resultant image with a right image.

- Compute **SURF** keypoints and corresponding descriptors for above stitched image and right image. **(3 points)**
- Find matching keypoints in two images and display the 20 best pairs. **(4 points)**

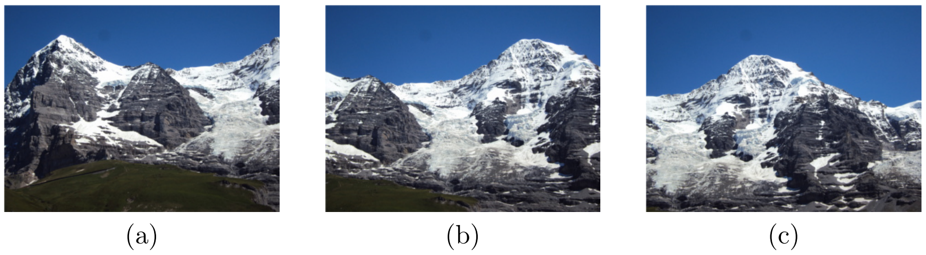

Figure 1: Three overlapping images of a same scene. (a) left image (b) central image (c) right image

- Find homography using RANSAC method. Use above implemented RANSAC function. Do not use OpenCV implementation. **(2 points)**
- Apply transformation to right image. The stitched image should not be transformed. Use OpenCV function `warpPerspective` for image transformation. **(2 points)**
- Stitch transformed right image and already stitched left and central images together using **linear** image blending. **(3 points)**
- Display final panoramic image. You can crop panoramic image to remove excess zero-padding; however it is not compulsory. **(1 point)**
- Compare and comment on effectiveness of pyramid and linear blending. Discuss the pros and cons of both methods. **(2 points)**

3 Object Detection

In this question you will learn to detect pedestrians in a given scene using Histogram of Gradient (HoG) features. You are given a small training set of 30 images of pedestrian and a test image containing 4 pedestrian (see Figure 2). Use following instructions to perform the detection task.

- Training
    1. Resize the training images to 128 $\times$ 128. **(1 points)**
    2. Compute HoG features using cell size of 4 $\times$ 4 pixels, block size of 2 $\times$ 2 cells and 9 orientation bins **(5 points)**
       (Suggestion: Make a function which takes list of images as argument and delivers list of HoG features as output. The same function can be used during testing.)
    3. Calculate and store the mean feature map across training images. **(1 points)**

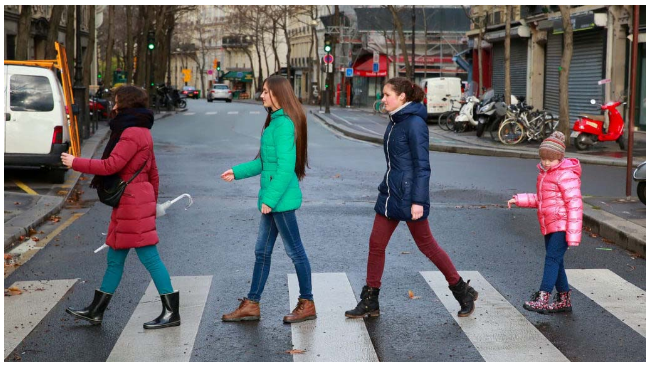

Figure 2: Test image for pedestrian detection

4. Display 9 orientation channels of the mean feature maps for the first block. **(3 points)**

- Testing

1. Extract overlapping windows from the test image. Experiment with stride and size of the window to achieve good performance. **(3 points)**
2. Resize windows to 128×128 and Compute HoG features similar to what was done during training. **(5 points)**
3. Compute the Euclidean distance between the feature map of each window and the mean feature map of training images. **(2 points)**
4. Threshold the Euclidean distances to detect pedestrians in the test image. Display the detected window. Experiment with threshold values in order to achieve good performance. **(3 points)**
5. Discuss the performance of above method. Did you detect all four pedestrians in the test image? Suggest methods to improve results. **(3 points)**