Homework 7
Computer Vision, Spring 2019
Due Date: April 29, 2019
Total Points: 24

This homework contains two written assignments and two programming challenges. All submissions are due at the beginning of class on **April 29, 2019**. The written assignments should be turned in as a hard copy at the beginning of class, and the challenges should be submitted according to the instructions in the document "**Guidelines for Programming Assignments.pdf**" before the beginning of class.

Written Assignments

Problem 1: We discussed the optical flow constraint equation. This constraint states that the optical flow estimates *(u, v)* at a point *(x, y)* in image space lie on a straight line whose coefficients are the derivatives of image brightness in the *x, y* (spatial), and *t* (temporal) dimensions. Clearly, this constraint does not yield a unique *(u, v)* solution for each point *(x, y).* Now consider a structured environment (say, industrial) where illumination can be controlled (changed) at a speed much faster than the motion of objects in the scene. Using the optical flow constraint equation show how two (perhaps unknown) different illuminations can be used to obtain a unique solution for *(u, v)* at each image point. Hint: Two illuminations provide two constraints rather than one. **(4 points)**

Programming Assignments

This programming assignment has two challenges (each with its own subset of milestones or unit tests). Instructions and summary corresponding to these are given below. **runHw7.m** will be your main interface for executing and testing your code. Parameters for the different programs or unit tests can also be set in that file.

Before submission, make sure you can run all your programs with the command `runHw7('all')` with no errors.

MATLAB is optimized for operations involving matrices and vectors. Avoid using loops (e.g., for, while) in MATLAB whenever possible—looping can result in long running code. Instead, you should "vectorize [1]**" loops to optimize your code for performance. In many cases, vectorization also results in more compact code (fewer lines to write!). If you are new to MATLAB, refer to these articles** [1] [2] **on techniques to optimize MATLAB code.**

Challenge 1: In this challenge you are asked to develop an optical flow system. You are given a sequence of 6 images (**flow1.png – flow6.png**) of a dynamic scene. Your task is to develop an algorithm that computers optical flow estimates at each image point using the 5 pairs (1&2, 2&3, 3&4, 4&5, 5&6) of consecutive images.

Optical flow estimates can be computed using the optical flow constraint equation and Lucas-Kanade solution presented in class. For smooth motions, this algorithm should produce robust flow estimates. However, given that the six images were taken with fairly large time intervals in between consecutive images, the brightness and temporal derivatives used by the algorithm are expected to be unreliable.

Therefore, you are advised to implement a different (and simpler) optical flow algorithm. Given two consecutive images (say 1 and 2), establish correspondences between points in the two images using template matching. For each image point in the first image, take a small window (say 7x7) around the point and use it as the template to find the same point in the second image. While searching for the corresponding point in the second image, you can confine the search to a small window around the pixel in the second image that has the same coordinates as the one in the first image. The center of the 7x7 image window in the second image that is maximally correlated with the 7x7 window in the first image is assumed to be the corresponding point. The vector between two corresponding points is the optical flow *(u,v)*.

Write a program `computeFlow` that computes optical flow between two gray-level images, and produces the optical flow vector field as a "needle map" of a given resolution, overlaid on the first of the two images.

```
result = computeFlow(img1, img2, search_half_window_size,
template_half_window_size, grid_MN)
```

You may use the Image Processing Toolbux function `normxcorr2`. The needle map (and hence optical flow) should only be computed and drawn on an MxN grid equally sampled over the image. You can use the MATLAB function `quiver` to annotate the image with the computed optical flow.

You need to choose a value for the grid spacing that gives good results without taking excessively long to compute. **(6 points)**

For debugging purposes use the test case in `debug1a`. In this synthetic case, the flow field consists of horizontal vectors of the same magnitude (translational motion parallel to the image plane). Note that in the real case, foreshortening effects, occlusions, and reflectance variations (as well as noise) complicate the result. **(2 point)**

Challenge 2: Your task is to develop a vision system that tracks the location of an object across video frames. Object tracking is a challenging problem since an object's appearance, pose and scale tend to change as time progresses. In class we have discussed three popular tracking methods: template based tracking, histogram based tracking and detection based tracking. In this challenge, we will assume the color distribution of an object stays relatively constant over time. Therefore, we will track an object using its color histogram.

A color histogram describes the color distribution of a color image. The color histogram that you will need to compute is defined as follows. Each bin of the color histogram represents a range of colors, and the number of votes in each bin indicates the number of pixels that have the colors within the corresponding color range.

Use the MATLAB function `rgb2ind` to divide the color range of an image into several bins (the ranges of bins are defined by a matrix called color map in MATLAB). You can specify the number of bins as an input to `rgb2ind`. `rgb2ind` then returns a color map along with an index image, in which the color of each pixel is represented as an index to the color map. After computing an index image, you can use the function `hist/histc` to compute the color histogram of the image. You may use the Image Processing Toolbux function `im2col`.

Be careful, in the initialization of your program, you should generate a color map from the object of interest, and compute all subsequent color histograms based on **the same** color map. It is only meaningful to compare two histograms computed based on the same color map.

Write a program named `trackingTester` that estimates the location of an object in video frames.

```
trackingTester(data_params, tracking_params)
```

`trackingTester` should draw a box around the target in each video frame, and save all the annotated video frames as PNGs into a subfolder given in `data_params.out_dir`. Use `drawBox.m` (included in the homework package) to draw a box on images.
(12 points)

Include all the code you have written, as well as the generated results, but DO NOT include the three tracking datasets in your submission.

References

[1] MathWorks. Vectorization. [Online]. http://www.mathworks.com/help/matlab/matlab_prog/vectorization.html

[2] MathWorks. Technique for Improving Performance. [Online]. http://www.mathworks.com/help/matlab/matlab_prog/techniques-for-improving-performance.html