Homework 5: Optical Flow & Tracking

CS 639, Fall 2020

Due on December 3

Total points: 14

Please follow the homework guidelines in your submission.

`runHw5.m` will be your main interface for running your code. Parameters for the different programs or unit tests can also be set in that file. Before submission, make sure that you can run all your programs with the command `runHw5("all")` with no errors.

Challenge 1: Optical Flow (6 points)

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 correspondence between points in the

two images using template matching. For each image point in the first image, take a small window (say 7 x 7) 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 7 x 7 image window in the second image that is maximally correlated with the 7 x 7 window in the first image is assumed to be the corresponding point. The vector between two corresponding points is the optical flow *(u,v)*.

Compute Flow (5 points):

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)
```

We have already written the skeleton of the `computeFlow` function for you. In the `computeFlow` function you will see a double for loop. Within that for loop you will have to complete the remainder parts of the code that:

1. Extract the `template` and `search_window` from img1 and img2
2. Find the location in the `search_window` that best matches the template. Recall that normalized cross-correlation is what we use in template matching to measure how well the template matches.
3. Given the location of the best match, store the optical flow (motion in x and y).

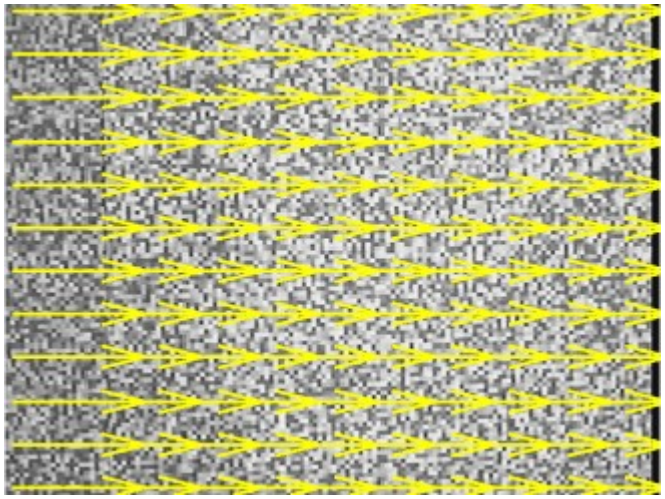
You may use the Image Processing Toolbox function `normxcorr2`. The needle map (and hence optical flow) should only be computed and drawn on an M x N grid equally sampled over the image.

At the end of the `computeFlow` function we have added code to that overlays the computed optical flow as a needle map on top of the image. We 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.

Synthetic case (1 point):
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.

If you have implemented the `computeFlow` function correctly the debug1a code should output an image that looks like this:



Challenge 2: Tracking (8 points)

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 intensity distribution of an object stays relatively constant over time. Therefore, we will track an object using its intensity histogram.

To extract plain intensity from a color image, you need to convert it to grayscale through the MATLAB function `rgb2gray`. After this conversion, you can use the function `hist`/`histc`/`histcounts` to compute the histogram of the image. Make sure that the histogram bins are consistent across all the frames!

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 sub-folder given in `data_params.out_dir`. Use `drawBox.m` (included in the homework package) to draw a box on images.

We have already written the skeleton of the `trackingTester` function for you. Please read the function and fill in the necessary code in the places indicated by the `%--- FILL IN CODE ---` message. In the `trackingTester` function we provide you will still need to complete the code that does the following:

1. Extract the rectangular region of interest from the first frame and create its histogram .

2. For each subsequent frame
    a. Extract the search window
    b. Create a histogram for all possible candidate regions in the search window
    c. Find the histogram that best matches the one you originally computed
    d. Update the location of the rectangular bounding box that tells us where the object we are tracking is at.

At the end of challenge2a, challenge2b, and challenge2c, an output video is created by calling the generateVideo function. IMPORTANT: This video should be in color! You only need the grayscale conversion for generating the histograms, so you should do that with copies of the frames.

Include all the code you have written, as well as this generated video of the annotated frames, in your submission. DO NOT include the three tracking datasets, and the individual output frames, in your submission.