Homework 4
M1522.001000 Computer Vision (2016 Spring)
Due: Wednesday May 11, 11:59PM

In this assignment, you will implement a tracker for estimating dominant affine motion in a sequence of images and subsequently identify pixels corresponding to moving objects in the scene. You will be using the images in the file `Sequence1.zip` consisting of aerial views of moving vehicles from a non-stationary camera.

1 Preliminaries

Remember that there are a lot of files in the directory that we give you. This should not translate into code that runs for a long time, since you should be able to vectorize your code since the vectorization is almost exactly the same as in Assignment 1. Our code takes a few seconds per image pair.
All the images are 640x480 in size. However, your implementation should run on only two images at a time, and since you will never have to load all of the images into memory at once, you should have no memory problems.
Finally, note that even with a few seconds running per pair of image, processing over a thousand frames takes a fair amount of time. We have provided all the frames, since it is fun to see how long things are tracked.

1.1 Dominant Motion estimation

You will start by implementing a tracker for affine motion using the equations for affine flow described in class and in the Motion-I slides from class. The tracker will only work on two frames at a time.
To estimate dominant motion, the entire image $I(t)$ will serve as the template to be tracked in image $I(t+1)$, i.e. $I(t+1)$ is assumed to be approximately an affine warped version of $I(t)$. This approach is reasonable under the assumption that a majority of the pixels correspond to stationary objects in the scene whose depth variation is small relative to their distance from the camera. Using the affine flow equations, you can recover the 6-vector $p = [p_1\ p_2\ p_3\ p_4\ p_5\ p_6]'$. of affine flow parameters. They relate to the equivalent affine transformation matrix as:

$$M = \begin{bmatrix} 1+p_1 & p_3 & p_5 \\ p_2 & 1+p_4 & p_6 \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

relating the homogenous image coordinates of $I(t)$ to $I(t+1)$ according to $x_{t+1} = Mx_t$ (where $x = [x\ y\ 1]^T$). For the next pair of consecutive images in the sequence, image $I(t+1)$ will serve as the template to be tracked in image $I(t+2)$, and so on through the rest of the sequence. Note that M will differ between successive image pairs.

Each update of the affine parameter vector, Δp is computed via a least squares method using the pseudo-inverse as described in class. `Note`: Image $I(t)$ will almost always not be contained fully in the warped version of I(t + 1). Hence the matrix of image derivatives,

A, and the temporal derivatives, I_t, must be computed only on the pixels lying in the region common to $I(t)$ and the warped version of $I(t+1)$ to be meaningful.

2 Submitting Your Assignment

Your submission for this assignment consists of answers to a few theory questions, The answers to the theory questions should be in a plaintext file or a pdf.
the code for your MATLAB implementation and a short writeup describing any thresholds and parameters used, interesting observations you made or things you did differently while implementing the assignment. You need to zip the following items, name it as `M1522.001000_HW4_(your student ID)#.zip`, and send it to TA's email. (`ta.cv@vision.snu.ac.kr`)

Your submitted zip file should include the files arranged in this layout:

- `theory.txt` or `theory.pdf`
- `experiments.txt` (or `.pdf`): answers to the theory questions
- Folder `data`
  - Folder `frame`
- Folder `matlab`
  - `hysthresh.m` , `test_motion.m` (*already given*).
  - `SubtractDominantMotion.m` `trackTemplate.m`
  - `mainScript.m`
  - You can also include any extra helper functions.
- Folder `matlab`
  - Folder `output` (*your Car tracking output*).
  - `movie.avi`

Refer to Section 4 and Section 5 for the details of the above files.

Your zip file should be sent before the beginning of the class of the due date. Later than that, you will use one late day.

3 Theory Questions

Question 1: Recovering component of motion (5 points)

Give an example of a sequence of images from which you could not recover either horizontal or vertical components of motion. Explain in terms of number of knowns and unknowns of the optical flow equation.

Question 2: Recovering component of motion (5 points)

Give an example of a sequence of images from which you could always recover horizontal or vertical components of motion. As in Question 1, explain with regards to knowns and unknowns.

Question 3: Moving object in moving camera (5 points)

Consider a situation where the motion of the camera is in the forward direction. Does the optical flow equation still hold? Find a simple way to figure out which scene point the object is moving towards.

Question 4: Observing translation motion (5 points)

Consider a camera observing a 3D scene in which each point has the same translation motion. This means the velocity V at each point has the same components. Show that all the optical flow vectors originate from the same point on the image plane. This point is called the focus of expansion.

4 Programming

Once you are able to compute the transformation matrix M relating an image pair $I(t)$ and $I(t+1)$, a naive way of determining pixels lying on moving objects is as follows. Warp the image $I(t)$ using M so that it is registered to $I(t+1)$, and subtract it from $I(t+1)$. The locations where the absolute difference exceeds a threshold can then be declared as corresponding to locations of moving objects. For better results, hysteresis thresholding may be used. So we have provided MATLAB code for hysteresis thresholding (`hysthresh`).

We provide following frame data.

- Folder `frame (data/frame/)` : It includes short video sequence: The video sequence is the data for 261 car driving images, and the size of each image is (640×480).

Part 1 Moving Object Detection (20 points)

Write the function

```
[moving_image] = SubtractDominantMotion(image1, image2)
```

where `image1` and `image2` (in uint8 format) form the input image pair and `moving_image` is a binary image of the same size as the input images, in which the nonzero pixels correspond to locations of moving objects.
The script `test_motion.m` that we will use for grading this section has been provided to you for testing your function. This script simply makes repeated calls to `SubtractDominantMotion` on every consecutive image pair in the file `Sequence1.zip` , makes an AVI movie out of the `moving_image` returned for every image pair processed, and saves it as a file `motion.avi` for your offline viewing pleasure. Note that as mentioned before, creating this movie can take a long time for a large number of files.

`Note`: VideoWrite function can generate compressed AVI files. do not submit uncompressed avi file. submit your video file in output folder. It cannot must not be over 50 Mega bytes.

Estimate Dominant Motion (10 points)

You will notice that the estimates of moving pixels are a bit noisy. There are several reasons for this, ranging from poor image quality, changes in lighting with viewpoint and mpeg-related artifacts in the original sequence, to inaccurate affine registration, the choice of thresholds, and the obvious limitations of our overly simplified poor-mans-motion-detector! We do not expect perfect results this is, after all, real world data! You are free to implement any algorithm of your choice for estimating the `moving_image` following an affine registration step. But please follow the above function format. To denoise your output, you may also want to look at the imerode and imdilate morphological operators in MATLAB. and the provided hysteresis code should make substantial improvement over simple thresholding. Again, this is optional, Any experimental try will be gracefully accepted. But explain your way to remove noise your output.

- Scaling image coordinates can be crucial for this type of problem. You should scale coordinates to be between 0 and 1 when computing the A matrix. But be sure to undo that scaling whenever you need to use coordinates to actually reference pixels in an image. Also note that I_x and I_y will vary depending on your scaling.
- Note that transforming by an affine matrix M_1 and then by M_2 is the same as transforming by $M_{12} = M_2 M_1$. It may be easier to keep track of the total accumulated warp (e.g. M_{12} here) as you iterate towards convergence, and warp the original $I(t+1)$ using that cumulative warp at each iteration.
- Note that the spatial derivatives remain the same between iterations when converging to the correct affine parameters, p. The only thing that changes will be I_t as you warp $I(t+1)$ more and more to better match $I(t)$.
- Related to the previous point, the matrix A contains one row for every pixel in the image. But as discussed above, when we compute the temporal derivative, I_t , we should only consider those pixels of the warped version of $I(t+1)$ that overlap with $I(t)$. Thus, at a given iteration, we need to ignore some of the rows of A specifically those that correspond to pixels that do not overlap with the current warped version of $I(t+1)$. Note that this still does not require completely rebuilding A. Instead, you only need to keep track of which of its rows are currently active and index those accordingly when computing Δp.

Part 2 Template tracking (20 points)

Write the function `trackTemplate` that track a specific object in an image sequence. Using the template tracking method in the Motion notes and the translation-only model

```
trackTemplate(path_to_car_sequence,sigma)
```

Before tracking, your function should prompt the user to select a template from the first image (use `ginput` and let the user click on the top-left and bottom-right corner that will designate the template.) We suggest you try tracking the pick-up truck. Your function should create a new directory called `output` for the output. Display each image in the sequence with a box around the tracked object, and save the image (with the box) to your output directory. If your tracker loses the object before the end of the sequence, display a message saying so and exit gracefully. You will probably notice that your tracker will lose the template before the end of the sequence. One way to fix this is to smooth the template and the window in each image using a Gaussian. Try this with sigmas ranging from 1 to 10 (this is the `sigma` input to the `trackTemplate` function). What do you notice? Why?

- Please note that we have covered all the math that you need for the above part in class. Please read the sections on Motion if you have problems.
- Remember that, similar to the first section, you will need to iterate your changes to u and v for every image pair.
- You will need to use the interp2 command in Matlab to interpolate non-integer row-column positions of your shifted window.

5 Experiments

(15 points)

Descriptions of your motion detector, and template tracker. These descriptions should be high level explanations of your algorithm design and how you tackled each problem. Make an itemized list of your assumptions and parameter choices. This may include the template size, and termination criteria. Please give us all the pertinent details, but be concise. Include short discussion about the effect of different smoothing sigmas and a possible cause. Which sigma worked best?