Homework 4
M1522.001000 Computer Vision (2020 Fall)
Due: Nov 19 Thursday 11:59PM.

The goal of this homework is to explore Stereo Camera Models, Optical Flow and Lucas-Kanade Method. There are 3 theory questions and 2 **Python** programming questions on this assignment, and 100 points in total.

Put your **code and writeup** into a directory called "`(studentid)-(yourname)-HW4`" and pack it into a `zip` named "`(studentid)-(yourname)-HW4.zip`".

For example, `202012345-gildonghong-HW4.zip`.

Your writeup should be **typed** with **English**. Please do **not** attach your code to writeup. Upload your `zip` file to **ETL** until due date. Refer to the **ETL** page for the policies regarding collaboration, due dates, extensions, and late days.

Your homework should be formatted as following:

```
(studentid)-(yourname)-HW4
├─ writeup.pdf
├─ run_motion.py
├─ data
│  ├─ 0.jpg
│  ├─ ...
│  └─ 149.jpg
└─ motion.mp4 (you don't need to upload this)
```

Your `zip` file should be sent in before the due. Later than that, only one late day is allowed. Finally, note that we will use a code similarity checker to detect plagiarism. You are expected to work on the assignment individually. I firmly believe that every student can do his or her own work. For your sake, please do not copy and paste others code. Good Luck!

1 Theory Questions

1.1 Camera Models (10 points)

Suppose we want to solve for the intrinsic parameter matrix $\mathbf{K}$ of a camera and that we know that the world coordinate system is the same as the camera coordinate system. Assume the matrix $\mathbf{K}$ has the structure outlined below. Assume that we are given 100 correspondences. Each correspondence consists of a world point (x_i, y_i, z_i) and its

projection (u_i, v_i) for $i \in \{1, \cdots, 100\}$.

$$\mathbf{K} = \begin{bmatrix} K_{11} & K_{12} & K_{13} \\ 0 & K_{22} & K_{23} \\ 0 & 0 & K_{33} \end{bmatrix} \tag{1}$$

(a) [5 points] Assume that all intrinsic parameters in $\mathbf{K}$ are unknown. Set up an equation of the form $\mathbf{Ax} = \mathbf{0}$ to solve for the unknowns in $\mathbf{K}$ (where $\mathbf{A}$ is a matrix, and $\mathbf{x}$ and $\mathbf{0}$ are vectors). Be specific about what the matrix $\mathbf{A}$ and vector $\mathbf{x}$ are.

(b) [5 points] Explain how to solve for the unknowns in the intrinsic parameter matrix $\mathbf{K}$. Make sure $K_{33} = 1$.

1.2 Epipolar Geometry (15 points)

Consider the point $\mathbf{X}$ expressed with respect to the world coordinate system. Suppose that the transformation between two cameras is described by a rotation matrix $\mathbf{R}$ and a translation vector $\mathbf{t}$ so that $\mathbf{X}' = \mathbf{RX} + \mathbf{t}$. Assume that the first camera coordinate system is equivalent to the world coordinate system and that its camera matrix is $\mathbf{K}$. The intrinsic parameter matrix of the second camera is $\mathbf{K}'$.

(a) [3 points] Write the camera projection matrices $\mathbf{M}$ and $\mathbf{M}'$ in terms of $\mathbf{K}$, $\mathbf{K}'$, $\mathbf{R}$, and $\mathbf{t}$.

(b) [3 points] What are the expressions for the epipoles, $\mathbf{e}$ and $\mathbf{e}'$ in terms of one or more of the following: $\mathbf{M}$, $\mathbf{M}'$, $\mathbf{R}$, and $\mathbf{t}$.

(c) [6 points] Using the results from the earlier parts, show that $\mathbf{e}' \times \mathbf{x}' = [\mathbf{e}']_\times \mathbf{K}'\mathbf{R}\mathbf{K}^{-1}\mathbf{x}$ where $\mathbf{x}'$ is the image of $\mathbf{X}$ in the second camera coordinate system.

(d) [3 points] What is the geometric interpretation of $\mathbf{e}' \times \mathbf{x}'$?

1.3 Optical Flow (25 points)

(a) [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.

(b) [5 points] Give an example of a sequence of images from which you could always recover horizontal or vertical components of motion. As in (a), explain with regards to knowns and unknowns.

(c) [7 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.

(d) [8 points] Consider a camera observing a 3D scene in which each point has the same translation motion. This means the velocity 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.

2 Programming

First of all, make sure you have **Python** 3.6+ installed. You can check your version with command `python3 -V`. To run the program, you will also need Numpy, SciPy , Scikit-Image and OpenCV packages. To install them with the package manager for Python, run `pip3 install numpy scikit-image scipy opencv-python`. Then, you should complete the code. Specifically, you have to fill in two blocks starting with `### START CODE HERE ###` and ending with `### END CODE HERE ###`. For this, usage of third-party image processing library other than `numpy` & `RectBivariateSpline` is **prohibited**. Never use **OpenCV**.

In this section, 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 directory `data`, which consists aerial views of moving vehicles from a non-stationary camera.

2.1 Dominant Motion Estimation

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 $p = [p_1\ p_2\ p_3\ p_4\ p_5\ p_6]^T$ of affine flow parameters. They relate to the equivalent affine transformation matrix as:

$$M = \begin{bmatrix} 1+p_1 & p_3 & p_5 \\ p_2 & 1+p_4 & p_6 \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

relating the homogenous image coordinates of $I(t)$ to $I(t+1)$ according to $\mathbf{x}_{t+1} = M\mathbf{x}_t$ (where $\mathbf{x} = [x\ y\ 1]^T$). 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 and the temporal derivatives 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.

For better results, Sobel opterators may be used for computing image gradients. So, we have provided a code for them.

Part 1: Lucas–Kanade Method (25 points)

Complete the following function and explain your code in the writeup:

```
dp = lukas_kanade_affine(img1, img2, p, Gx, Gy)
```

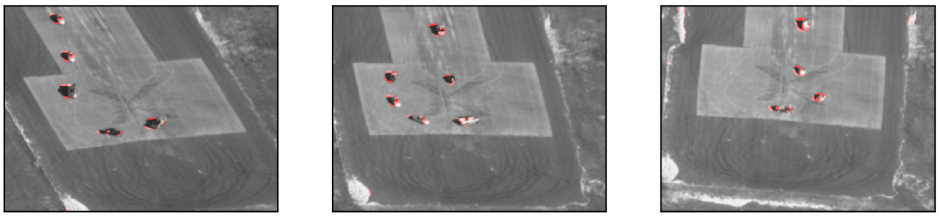

Figure 1: Sample results of affine motion subtraction at frame 0, 60 and 120

where `dp` is Δp, and `img1`, `img2`, `p`, `Gx` and `Gy` are $I(t)$, $I(t+1)$, p, I_x and I_y, respectively. Do not attach your code to the writeup. The explanation in the writeup should include high level explanations of your algorithm design and an itemized list of your assumptions and parameter choices.

A naive implementation may not work well. Then, refer to the following tips:

- Scaling image coordinates can be crucial for this type of problem. You should scale coordinates of image gradients ∇I to be between 0 and 1. But be sure to undo that scaling whenever you need to use coordinates to actually reference pixels in an image.
- This function will probably take too long. Then, try to subsample pixels when computing the Hessian matrix H and Δp.

2.2 Moving Object Detection

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 regarded as moving objects.

For better results, hysteresis thresholding may be used. So we have provided a code for hysteresis thresholding.

Part 2: Affine Motion Subtraction (25 points)

Complete the following function and explain your code in the writeup:

```
mask = subtract_dominant_motion(img1, img2)
```

where `img1` and `img2` form the input image pair, and `mask` is a binary image of the same size that dictates which pixels are considered to be corresponding to moving objects. You should invoke `lukas_kanade_affine` in this function to derive Δp, and produce the aforementioned binary mask accordingly. Do not attach your code to the writeup. The explanation in the writeup should include high level explanations of your algorithm design and an itemized list of your assumptions and parameter choices. Any experimental try will be gracefully accepted.

Your implementation has to yield similar or better results in Figure 1. In addition, its total running time should be less than 30 minutes. TA's implementation runs within 13 minutes in Google Colab environment.

Revision History

Revision	Date	Author(s)	Description
1.0	2020/11/5	TA	HW4 out
1.1	2020/11/8	TA	A typo is fixed in 1.2.(c). $\mathbf{e}' \times \mathbf{x}' = [\mathbf{e}']_\times \mathbf{K}'\mathbf{R}\mathbf{K}^{-1} \rightarrow \mathbf{e}' \times \mathbf{x}' = [\mathbf{e}']_\times \mathbf{K}'\mathbf{R}\mathbf{K}^{-1}\mathbf{x}$