Version: February 11, 2019	Name of student _____
© Computer Vision Laboratory	Personal number _____
Linköping University	Date _____
	Teaching assistant _____

Motion estimation

Lab Exercise 2

1 Introduction

In this exercise you will implement methods for estimation of motion in image sequences (or rather optical flow which is the *apparent* motion). The methods will be implemented in Python.

1.1 Preparations

Before the exercise you should have read through this exercise guide and completed the home exercises. They are all clearly marked with a pointing finger. To be able to do this you should have read, and understood, the simplified paper on LK tracking that can be found at the course homepage and also completed lab exercise 1.

1.2 Initialization

To use Python in the ISY computer rooms, issue the following command at the shell prompt:
`$ module add courses/TSBB15`
After that you can use `python3 script.py` to run a Python script. For interactive work you can optionally use `ipython3`, which allows e.g. tab-completion and syntax highlighting (you should still use `python3` to run any scripts!).

1.2.1 Help code and notes

The helper functions referenced in assignments can be found by importing the `lab2` and `lab1` modules. Familiarize yourself with the available functions:

```
>>> import lab2
>>> help(lab2)
```

We suggest using `scipy.signal.fftconvolve` for convolutions, instead of `convolve2d`, as the latter can be slow for larger arrays.

2 Algorithm outline

Present a description of how the tasks outlined in Section 3 are to be solved, including

- A block scheme showing the needed steps
- The filters that are needed
- The pixel(point)wise operations that are needed
- The parameters that are set by the user

- Other possible parameters that are constant
- Suggestion of simple synthetic test data for debugging
- An idea of how to test each step of the algorithm

3 Single-scale LK

You will implement the Lucas-Kanade equation (Lucas-Kanade method with only one iteration). You can choose either the symmetric version[1] or the original non-symmetric version[2].

Implement a function `LK_equation(I, J, param1, param2,...)` → `V, C` that solves the Lucas-Kanade equation for all pixels. The input is a pair of images $\mathbf{I}, \mathbf{J}$ with size $M_1 \times M_2$, and a set of user parameters. Then the output is a dense motion vector field $\mathbf{V}$ of size $M_1 \times M_2 \times 2$, i.e. a motion vector for every pixel/region.

Hint: This is similar to lab 1, but you may not use this function to loop over pixels. Instead try to avoid for-loops over pixels as far as possible.

Extra

Some estimates will be more reliable than others depending on local image content, try to design a confidence measure $\mathbf{C}$ for every estimate as well (this is optional).

3.1 Demonstration

Create a script file that contains the following:

- A demonstration of the single-scale method on a simple synthetic motion between two frames of your own choice (e.g. global translation), where the true optical flow is known.
- A demonstration of the single-scale method on two frames from the real sequence 'forwardL.zip'. This sequence contains a zooming towards the center of the image. There is no ground truth, but you can at least show that the method gives a reasonable result by comparing the error $\|\mathbf{J}(\mathbf{x}) - \mathbf{I}(\mathbf{x})\|$ with $\|\mathbf{J}(\mathbf{x}+\mathbf{v}) - \mathbf{I}(\mathbf{x})\|$ (possibly ignoring pixels near the image border). The latter should be smaller. You can also make a plot function that toggles between the images $\mathbf{J}(\mathbf{x}+\mathbf{v})$ and $\mathbf{I}(\mathbf{x})$.

  You may downsample the images once before the motion estimation if your implementation seems too slow for the full resolution.

Hint: To visualize your estimated motion field $\mathbf{V}$, use the `gopimage` help function, or plot with `quiver`.

4 Multi-scale LK

Implement a function `LK_equation_multiscale(I,J,param1,param2,...)` → `V, C` that solves the Lucas-Kanade equation using the previous function in multiple scales. The basic idea is that you first estimate the motion field in a coarse scale. Then you use this estimate as prediction for the next finer scale, and so on.

Demonstrate the results as in the previous exercise. As another sequence you may also choose 'SC-car4.zip' that contains aerial images of a moving car.

---

[1] S. Birchfield, *Deriviation of Kanade-Lucas-Tomasi Tracking Equation*, http://www.ces.clemson.edu/~stb/klt/birchfield-klt-derivation.pdf

[2] B.D. Lucas and T. Kanade, *An Iterative Image Registration Technique with an Application to Stereo Vision*, In Proceedings of Imaging Understanding Workshop, 1981. The original article for KLT, http://www-cse.ucsd.edu/classes/sp02/cse252/lucaskanade81.pdf

Hint: The use of prediction could be implemented in a similar manner as in lab 1, where the template was re-interpolated in each iteration. But doing this for every pixel region in the image will be too slow. Instead, use the dense motion field on the coarse scale to re-interpolate the entire image all at once (i.e. create the image $\mathbf{J}(\mathbf{x}+\mathbf{v})$). Then use this image instead of the original one for the next finer scale.

Hint: You can use the same image size in all the scales and have different parameters when calling `LK_equation`, (standard deviation of the gradient filters, standard deviation of the region size and the size of the filters) which depend on the current scale `n`, see `sc` in the Python example below.

```
V_tot = ...
Jn = J
for n in range(number_of_scales, 0, -1):
    sc = 2 ** (n - 1)
    Vn, Cn = LK_equation(I, Jn, sc * param1, sc * param2, ...)
    V_tot += Vn
    Jn = ...
```

4.1 Outliers

The estimated dense motion field (at any scale) may contain outliers. Try to remove the outliers, using e.g. median filtering (see `scipy.signal`), before re-interpolating the image and computing the next finer scale.