

6. Optic flow estimation

Abstract

Optic flow estimation is an important building block for applications where visual information change dynamically over time. Mobile robot navigation, traffic monitoring, and human action recognition are three examples, among many others, which require or can benefit from optic flow information. In this lab, we will implement one simple, but foundational method for optic flow computation, the classical Lucas-Kanade algorithm.

Keywords

Optic flow • Lucas-Kanade algorithm

Contents

1	Lucas-Kanade algorithm	1
1.1	Computing the gradients	1
1.2	Computing $A^T A$ and $A^T b$	1
1.3	Solving for the flow	1
2	Synthetic motion	1
3	Real sequences	2
4	Additional and optional activities	2

1. Lucas-Kanade algorithm

Our first task is to complete the function

```
u,v = optical_flow(I1, I2, window_size, tau)
```

to compute the optic flow (u, v) between the image pair (I_1, I_2) with Lucas-Kanade (LK) algorithm, using local square windows of side length w (`window_size`), and τ (`tau`) as the minimum value of the eigenvalues of the matrix $A^T A$. Let's proceed step by step.

Computing the gradients

We first smooth the input images with a Gaussian filter, and then we have to compute the spatio-temporal gradients, I_x , I_y and I_t on the smoothed images. To that end, we define the corresponding three kernels for the x , y and t directions, and then apply 2D convolutions. The kernel and convolution for x is given. **1 ► Now, write the code to define the two other kernels and apply their respective convolutions.** For the temporal gradient, follow this: $I_t = (I_2 - I_1) * M_t = I_2 * M_t - I_1 * M_t$, where M_t is the 4×4 mask for a mean filter. Notation reminder: we use $*$ to denote the convolution operation (not a matrix product or an elementwise product), and I_h represents the image derivative with respect to the axis $h \in \{x, y, t\}$.

Computing $A^T A$ and $A^T b$

Remember that in LK method, for each local window of n pixels, we have that $Au = -b$, where A is the $n \times 2$ matrix of

the n spatial gradients, and b is the $n \times 1$ vector of temporal derivatives. To find out the optic flow u for that window we have that $u = (A^T A)^{-1} A^T b$.

In the provided code, you are given the arrays I_x , I_y , and I_t that contain the corresponding derivatives just for the local window around a particular pixel at (i, j) . From these, **2 ► you only have to compute the variables $A^T A$ (for $A^T A$) and $A^T b$ (for $A^T b$).**

Solving for the flow

Finally, **3 ► we can use `linalg.lstsq(C,d)`** to compute the least-squares solution x to $Cx = d$. Just pay attention to what matrix C , vector d , and x are in our case.

Remember that we can only compute the optic flow in those image locations where $A^T A$ is well-conditioned. So, before solving for u we must check for this. **4 ► We can verify** that either the smallest eigenvalue for $A^T A$ is larger than τ , or that $A^T A$ has rank 2. For this, we can rely on `np.linalg.eigvals()` and `np.linalg.matrix_rank()`, respectively.

2. Synthetic motion

We now have to test how `optical_flow()` works. Since the basic LK algorithm assumes small motions, it is good that we can generate the motions we want, under controlled conditions. It is harder to find real sequences that have the motion we are interested to check.

As you can see in the code, we use `SimilarityTransform()` to define the transformation with the desired amounts of translation, scale and rotation, which is then applied to the actual image with `warp()`.

To visualize the computed flow, you are provided with the function `display_optic_flow()`. Good visualisations of outputs of algorithms of optic flow (and others) is always critically important during the development and analysis of these algorithms.

In the following, you can use the threshold $\tau = 0.01$. **5 ► Experiment with different amounts of translation** and find out up to which magnitudes LK fares well. **6 ► Regarding**

what the function `plot_optic_flow()` displays, make sure you understand, besides the arrows, what the colors in the background represents. Check the results look qualitatively good (when translation is small enough). 7 ► *Complement this visual inspection* of the flow map with the histogram of the magnitude of the optical flow. 8 ► *Vary also the window size w* (from the smallest $w = 3$ up to a few tens) and notice the impact on the estimated flow (both the map and the magnitude histogram). Does w have an influence on the sparsity/density or smoothness of the map? Try to find an explanation.

9 ► *Repeat the above experimentation* with the scale factor (both “zoom in” and “zoom out”), and then with rotations. Notice that, unlike the translations, the magnitude of the flow at different image positions should be different. Make sure the output make sense, qualitatively.

3. Real sequences

We can now study how optic flow performs in some image sequences from real scenes. Although ground-truth optic flow is hardly available in these cases and, therefore, estimation error cannot be easily computed, we can anyway evaluate the result qualitatively in these more challenging (and interesting) scenarios, where one or more objects may move independently, in contrast to the global movement of the previous synthetic examples.

10 ► *Experiment on some of consecutive frames* of the Hamburg taxi sequence, which is one example of the classical sequences which optic flow methods used to be tested on. Although local motion in this sequence can be considered to be “large” and, therefore, the vanilla LK method might not perform very well, find for which window sizes w we can get reasonable results. What potential benefit can we get by increasing τ one or two orders of magnitude?