Laboratory Session 6: Optical flow.

In this laboratory session we will implement an optical flow estimator based on the Lucas-Kanade algorithm identifying its uses and limitations. Finally, we will test a set of approaches for dense optical flow estimation.

Goals of the assignment:

1. Implement a Kanade-Lucas-Tomasi tracker.
2. Understanding dense optical-flow.

Evaluation of the assignment:

The resulting work will be shown by presenting the obtained results and the code will be submitted through the ADD (Moodle).

1. Ground truth data

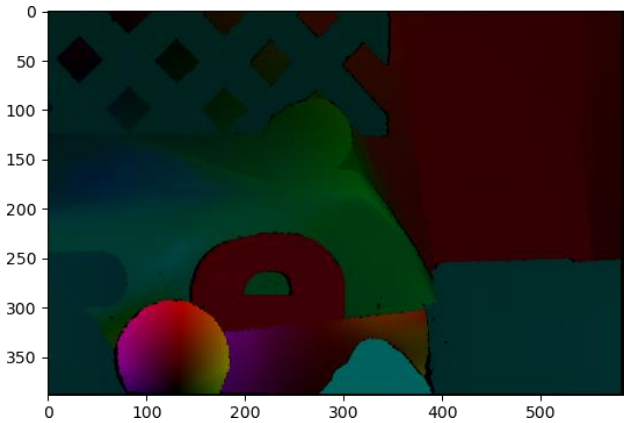
We provide a pair of images `frame10.png`, `frame11.png` from the Middlebury College dataset that includes ground truth of the optical flow in all the pixels (`flow10.flo` file).

`https://vision.middlebury.edu/flow/data/`

We also provide a function for reading this ground truth and several plotting functions in `plotGroundTruthOpticalFlow.py`. Notice that in this sequence there exist a set of objects moving in different directions and the motion is small.

frame 10

frame 11



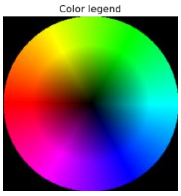


Fig.1 Input data and optical flow ground truth

The ground truth for the optical flow can be loaded using the function `read_flo_file`. **Notice that in some points the optical flow is unknown**. You can identify these points when the values are greater than 1e9.

2. Sparse optical flow estimation

2.1 Compute the motion with pixel accuracy by using Normalized Cross Correlation (NCC) brute-force search. For that, you can complete the template included in `NCCTemplate.py` considering an 11x11 centered window. Notice that you are going to obtain a discrete motion in pixels (integers).

2.2 Starting from the solution obtained in previous section, implement the sparse optical flow estimation based on Lucas Kanade approach that refines your NCC brute-force search. Use an 11x11 centered window. For efficiency reasons consider the following approximation $J_1(\mathbf{x}_i+\mathbf{u}) \approx J_0(\mathbf{x}_i)$ resulting in the following algorithm:

Compute $\begin{pmatrix} I_x & I_y \end{pmatrix} = J_0(\mathbf{x}_i)$

Compute $\mathbf{A}$ matrix

Check that $\mathbf{A}$ is invertible by computing the determinant

while $\|\Delta\mathbf{u}\| \geq \varepsilon$

1. From current motion $\mathbf{u}$, compute $I_1(\mathbf{x}_i+\mathbf{u})$, i.e. the warped patch 0 on image1. Use bi-linear interpolation for this warping.
2. Compute $e_i = I_1(\mathbf{x}_i+\mathbf{u}) - I_0(\mathbf{x}_i) = I_t$, the error between the patches.
3. Compute vector $\mathbf{b}$ from the error between patches and the gradients.
4. Solve $\mathbf{A}\Delta\mathbf{u} = \mathbf{b}$.
5. Update $\mathbf{u} = \mathbf{u} + \Delta\mathbf{u}$.

end

2.3 Check your complete algorithm (first 2.1 and then 2.2) at the following points

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 257 \\ 117 \end{bmatrix}, \begin{bmatrix} 17 \\ 209 \end{bmatrix}, \begin{bmatrix} 301 \\ 310 \end{bmatrix}, \begin{bmatrix} 192 \\ 337 \end{bmatrix}, \begin{bmatrix} 199 \\ 286 \end{bmatrix}, \begin{bmatrix} 272 \\ 78 \end{bmatrix}$$

These points are also included in the provided file `points_selected.txt`, please check `plotgroundTruthOpticalFlow.py`.

2.4 Represent the optical flow and the error with respect to the ground truth (norm L2 of the difference) and discuss each particular case.

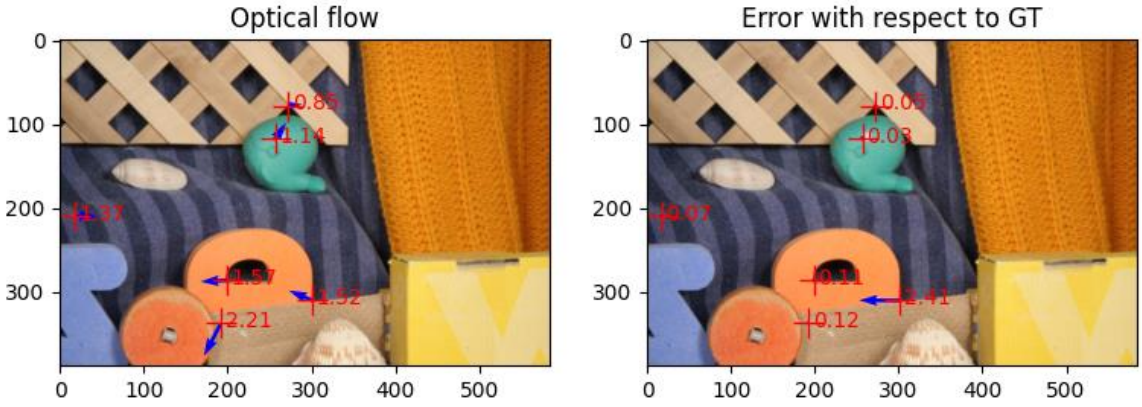


Fig 2. Optical flow at the given points and error in optical flow estimation.

3. Dense optical-flow estimation (Optional)

3.1 Apply the Lucas-Kanade optical flow estimator to a sub-selection of the image. Notice that in some pixels the computing of the inverse of matrix A is going to be ill-conditioned. You can detect these cases by using a threshold for the determinant of the matrix and ignore them, keeping the initial guess obtained from the cross correlation search. The size of these sub-images should be considerably greater than the patch size.

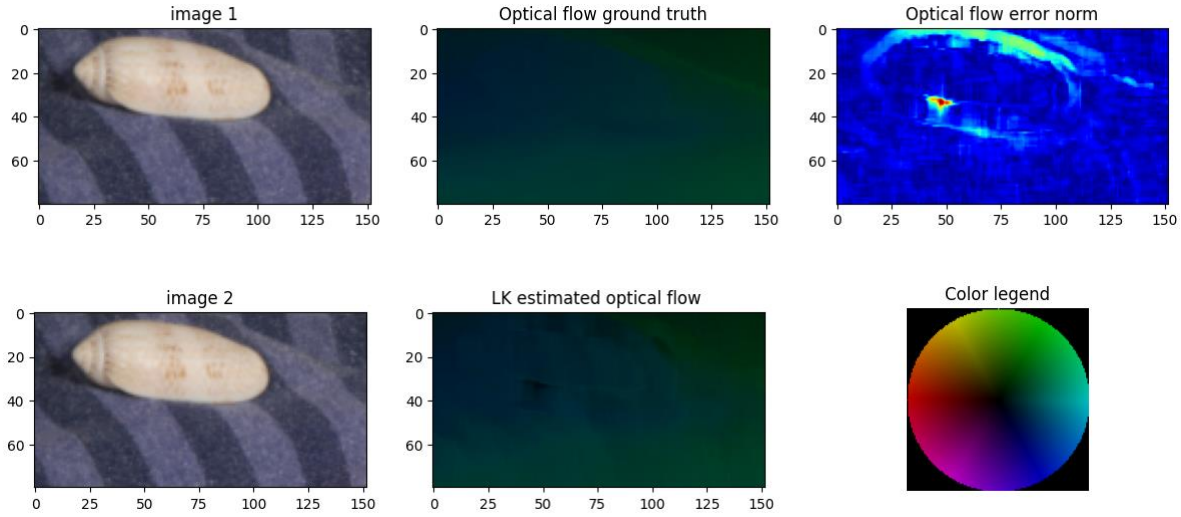


Fig 3. Lucas Kanade dense optical flow obtained in a sub-image.

3.2 Apply a variational based dense optical flow approach to the image and represent the norm of the error.

You can use opencv function `createOptFlow_DualTVL1` for testing a total variation algorithm with L1 norm regularization [1] ,`calcOpticalFlowFarneback` in order to estimate another dense optical flow approach or any other dense optical flow provided by openCV.

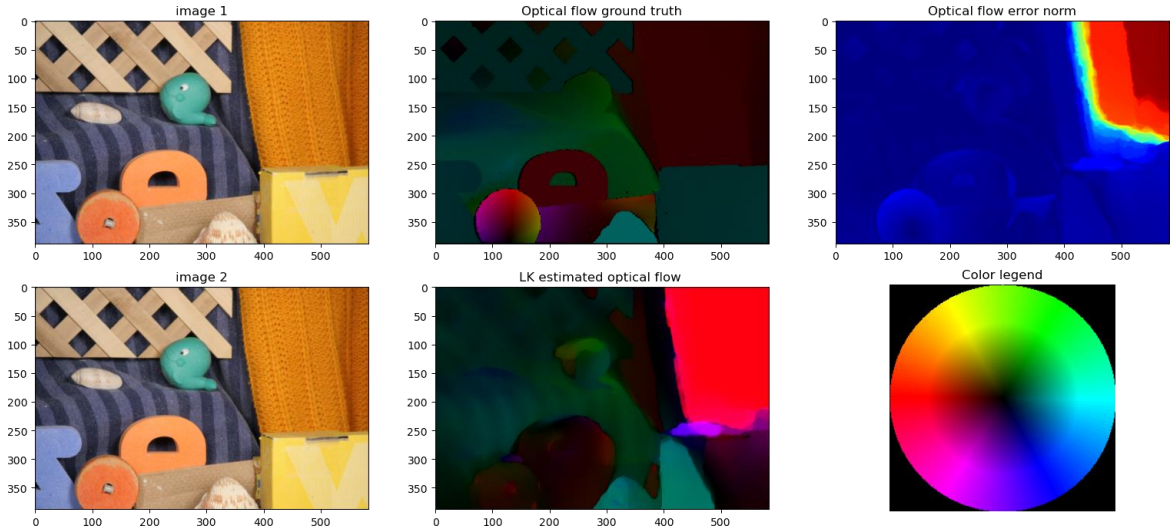


Fig 5. Total Variation L1 dense optical flow obtained in the whole image.

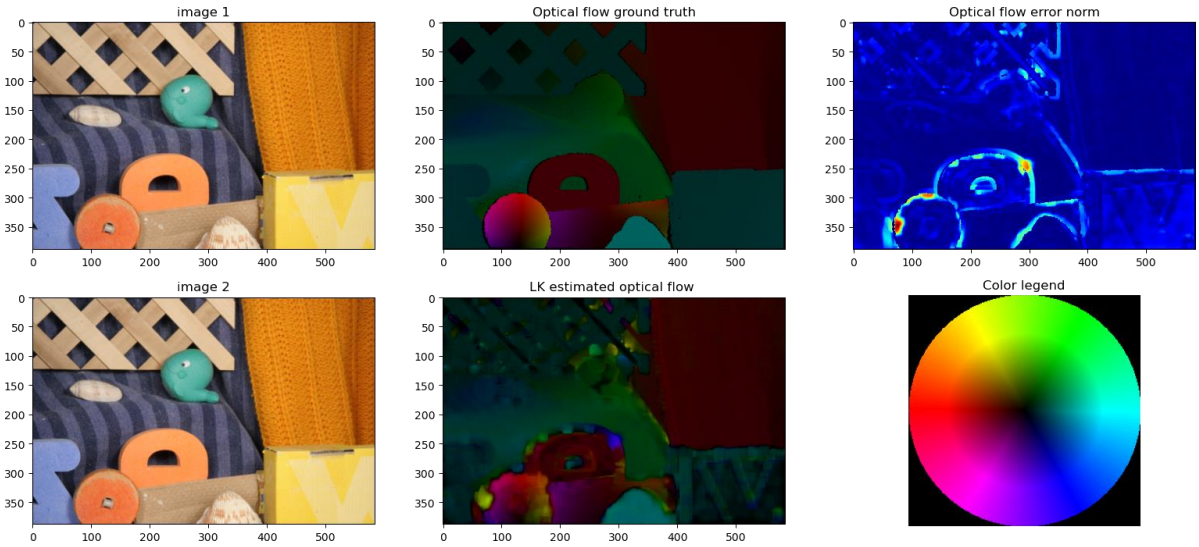


Fig 6. Farneback dense optical flow obtained in the whole image.

Appendix A

In current openCV versions (i.e. openCV 4.6) some optical flow objects have been moved to contributed packages (e.g. `createOptFlow_DualTVL1` is addressed by `cv.optflow.createOptFlow_DualTVL1()`) and you probably must install these packages by using the command:

```
pip install opencv-contrib-python
```

[1] Christopher Zach, Thomas Pock, and Horst Bischof. A duality based approach for realtime tv-l 1 optical flow. In *Pattern Recognition*, pages 214–223. Springer, 2007.

[2] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In Image Analysis, pages 363–370. Springer, 2003.