University of Michigan
EECS 504: Foundations of Computer Vision
Winter 2020. Instructor: Andrew Owens.

Problem Set 10: Optical Flow

Posted: Tuesday, March 31, 2020 **Due:** Tuesday, April 21, 2020

For Problem 10.1, please submit your solution to Canvas as a notebook file (`.ipynb`), containing the visualizations that we requested. Your `.ipynb` notebook should be named as <unique_name>_<umid>.ipynb. Example: `adam_01100001.ipynb`. Also, please remember to put your name and unique name in the first text block of the notebook. **Credit:** This problem was originally developed by the course staff for MIT 6.869.

The starter code can be found at:
https://drive.google.com/open?id=1EXGOAtOMTRLwTyhtyu46RgfiItvmssFl

We recommend editing and running your code in Google Colab, although you are welcome to use your local machine instead.

Problem 10.1 *Optical flow*

In this problem, you will implement the Lucas-Kanade (LK) optical flow algorithm for estimating dense motion between a pair of images. Since some of the motions might be too large for the Taylor approximation of the LK step, we will apply the algorithm in a coarse-to-fine manner.

You should refer to the comments in the provided notebook for further implementation details.

(a) Complete the implementation of the Lucas-Kanade algorithm (7 points). First, finish the implementation of the function:

`(u, v, warpI2) = lucas_kanade(I1,I2,u0,v0,winsize,medfiltSize,nIterations).`

This function receives as input two images I1 and I2 and an initial flow estimate (u0,v0), and computes the optical flow field (u,v) from image I2 to I1 using the Lucas-Kanade optical flow algorithm. The function receives the following parameters:

`winsize`: half the patch size,
`medfiltSize`: the size of the window for the spatial median filter,
`nIterations`: the number of flow refinement iterations.
`warpI2`: the image I2 warped according to (u,v). (3 points)

Next, use `lucas_kanade` as part of a coarse-to-fine optimization scheme. Implement the function:

`(u, v, warpI2) = coarse2fine_lk(I1, I2, nlevels)` that receives as input two images I1 and I2, and computes the optical flow (u,v) from I2 to I1 using the coarse-to-fine scheme of Lucas-Kanade (Algorithm 1), using your function `lucas_kanade`.

Algorithm 1: Coarse-to-Fine-LK(I_1, I_2, k)

1. Build k-level Gaussian pyramids G_1, G_2 for I_1, I_2
2. Find the optical flow field (u_k, v_k) from G_2^k to G_1^k at the coarsest pyramid level k using the Lucas-Kanade algorithm
3. Upsample the flow field for level k-1, and transform G_2^{k-1} toward G_1^{k-1} using (u_{k-1}, v_{k-1})
4. Update the optical flow estimation (u_{k-1}, v_{k-1}) at level $k-1$
5. Repeat 3-4 for levels $k-2$, $k-3$, ..., 1

(b) Run the algorithm on the car image pair supplied with the code. You can use the code in results section, which calls the `coarse2fine_lk` function from part (b) and displays the computed flow field and corresponding warp. The parameters in that script are set to values which we found to produce good results. Report the resulting optical flow image and the warped car2 image. Here is a sample output of the resulting optical flow image.

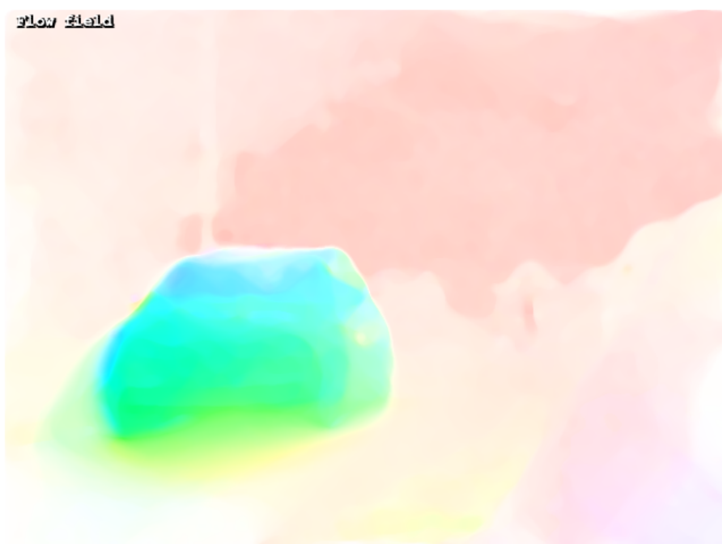

Figure 1: Sample output of the resulting optical flow image, color-coded using the visualization method described in [1].

(c) Your Lucas-Kanade implementation performs warping using a helper function we provided, `warp_flow_fast`. Here, you'll implement your own version of this function. Implement a function `warp_flow(I2, u, v)` that warps an image I2 using the flow field (u, v). For example, a use case for such a function inside Lucas-Kanade optical flow is to warp I2 using flow (u, v) to make it more closely resemble I1. Your solution should perform inverse warping and bilinear interpolation. You may implement it using nested `for` loops or with `numpy` primitives; you should *not* use built-in warping or interpolation functions (such as those in scipy or OpenCV). Handle out-of-bounds values by zero padding.

Since your solution may not be particularly fast `warp_flow_fast`, you do not need to use it in your Lucas-Kanade implementation. (3 points)

References

[1] Simon Baker, Daniel Scharstein, JP Lewis, Stefan Roth, Michael J Black, and Richard Szeliski. A database and evaluation methodology for optical flow. *International journal of computer vision*, 92(1):1–31, 2011.