Assign 2: The Secrets of Optical Flow

CS7.505 Computer Vision Spring 2023
International Institute of Information Technology Hyderabad

February 4, 2023

Start early PLEASE!!! Please stick to the provided function signatures, variable names, and file names. This assignment cannot be completed within two hours. Submit typed solutions (inserting handwritten images wherever necessary) the theory questions and the report. The report carries 50% of the assignment credits and should clearly show the experimental settings, results/observations and a discussion/conclusions based on the results. **There will be strict plagiarism checking and any breach of the rule could result in an F grade.**

1 Basics of Optical Flow

Estimating the motion of every pixel in a sequence of images is a problem with many applications, such as image segmentation, object classification, and visual odometry. Optical flow generally describes a sparse or dense vector field, where a displacement vector is assigned to a specific pixel position, that points to where that pixel can be found in another image.

1.1 Thought Experiments

1. The number-one use of optical flow in visual effects is for retiming a sequence — speeding it up or slowing it down. **Describe how optical flow could be used to create a slow-motion video**. You can find the answer in Amazing Slow Motion Videos With Optical Flow video on YouTube.
   It's movie time now! Let's watch an epic movie clip from one of the Academy Award-winning movies for Best Visual Effects - The Matrix (1999).

2. In The Matrix, one of the most remembered, iconic use of the tactic comes during the rooftop scene where Neo effortlessly dodges one bullet after another. (Re-)watch "Bullet Time" here and **explain briefly how optical flow is used here**. You may also find this video on bullet-time interesting.

3. Consider a Lambertian ball that is: (i) rotating about its axis in 3D under constant illumination and (ii) stationary and a moving light source. What do the 2D motion field and optical flow look like in both cases?

1.2 Concept Review

1. What does the objective function imply about the noise distribution?

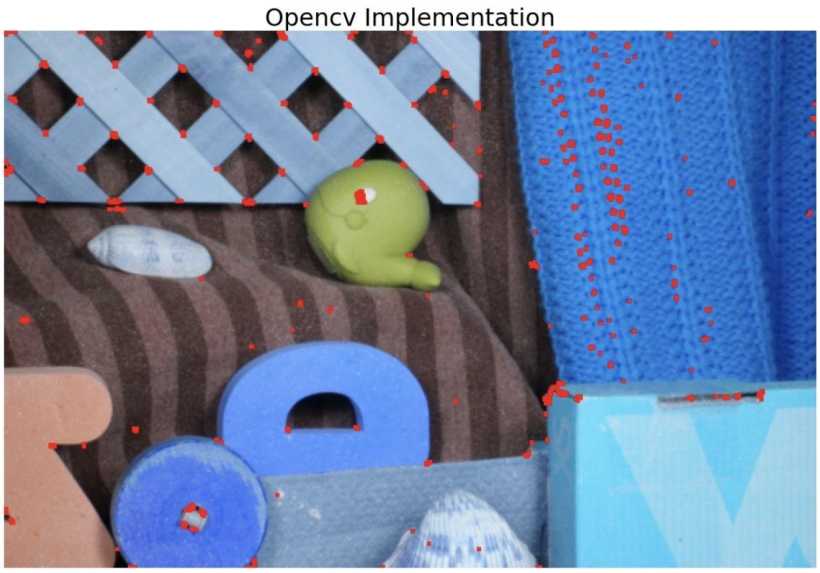
2. In optimization, why is the first-order Taylor series approximation done?

3. Geometrically show how the optical flow constraint equation is ill-posed. Also, draw the normal flow clearly.

2 Single-Scale Lucas-Kanade Optical Flow

2.1 Keypoint Selection: Selecting Pixels to Track

Use OpenCV's Harris Corner Detector implementation and visualize the feature points obtained by the algorithm. Visualize the detected pixels superimposed on the images for at least one image from each of the given sequences. This part of the assignment is based on the 1988 work, A Combined Corner and Edge Detector, by Chris Harris and Mike Stephens.



Reference image

Note: This part of the assignment will be used in the downstream tasks by the sparse LK method.

2.2 Forward-Additive Sparse Optical Flow

The main task is to implement the single-scale Lucas-Kanade (LK) algorithm based on the work described in the classic 1981 IJCAI paper An iterative image registration technique with an application to stereo vision by Bruce D. Lucas and Takeo Kanade.
This involves finding the motion (u, v) that minimizes the sum-squared error of the brightness constancy equations for each pixel in a window.
Your algorithm will be implemented as a function with the following inputs

```
LucasKanadeForwardAdditive(Img1, Img2, windowSize, tau)
```

Here, *u* and *v* are the *x* and *y* components of the optical flow, `Img1` and `Img2` are two images taken at times t = 1 and t = 2 respectively, and `windowSize` is a 1 × 2 vector storing the width and height of the window used during flow computation. `tau`(τ) is an additional threshold parameter such that if τ is larger than the smallest eigenvalue of A^TA, then the optical flow at that position should not be computed. A typical value for τ is 0.01.

Note: Before running your code on the images, you should first convert your images to grayscale and map intensity values to the range [0,1].

The main task can be broken into the following sub-parts:

1. Visualizing the dense optical flow of the `.flo` files given (Use the helper code of this)
2. Implement sparse LK Method
3. Plotting and comparing your implementation and OpenCV implementation for all consecutive frames for the three sequences. You have to add quiver plots superimpose on the images. (OpenCV implementation uses multi-scale LK so It is alright if the results are not the same, but the general flow will still be the same)

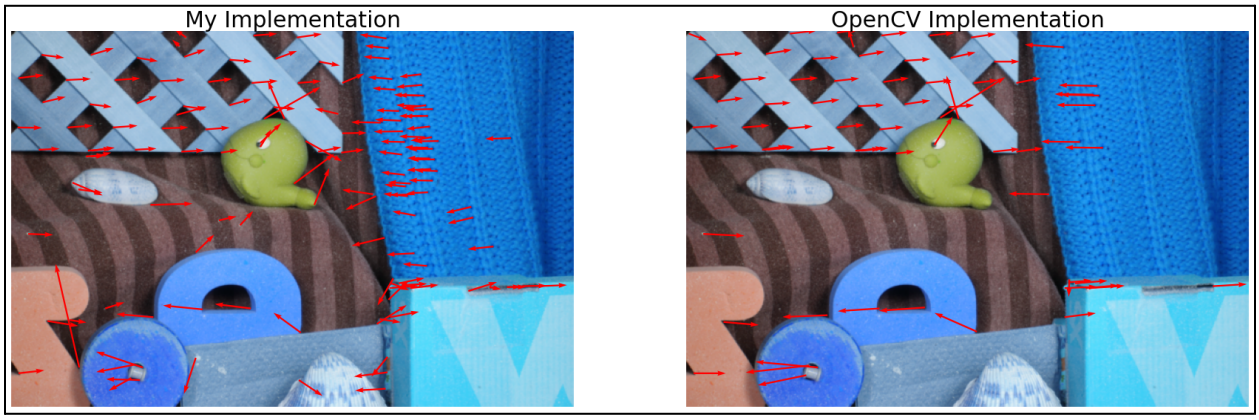


Image for reference (your output might be different)

4. Create a video out of the images to show the optical flow (This is an optional task, it helps you to see if the optical flow vectors are in line with the actual motion of the images)
5. Compute the Average End Point Error (EPE) between the ground truth optical flow in `.flo` files and your predicted optical flow (only for the feature points detect.

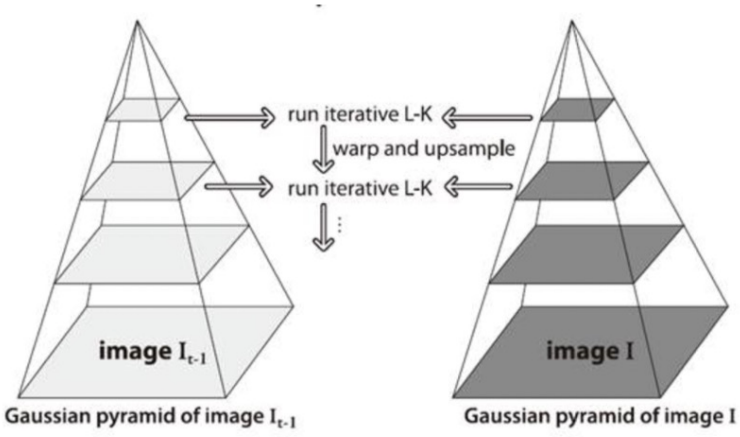
You will use all the given image sequences from the Middlebury Optical Flow dataset throughout part two to test your algorithm.

2.3 Analyzing Lucas-Kanade Method

1. Why is optical flow only valid in the regions where local structure-tensor A^TA has a rank of 2? What role does the threshold τ play here?
2. In the experiments, did you use any modifications and/or thresholds? Does this algorithm work well for these test images? If not, why?
3. Try experimenting with different window sizes. What are the trade-offs associated with using a small versus a large window size? Can you explain what's happening?
4. Did you observe that the ground truth visualizations are in HSV colour space? Try to reason it.

3 Multi-Scale Coarse-to-fine Optical Flow

Modify your algorithm to iteratively refine the optical flow estimate from multiple image resolutions. The basic idea is to initially compute the optical flow at a coarse image resolution. The obtained optical flow can then be upsampled and refined at successively higher resolutions, up to the resolution of the original input images. By computing the optical flow in this manner, we can circumvent the aperture problem to some degree – because the window size remains fixed across all resolutions, the algorithm effectively computes the optical flow using successively smaller apertures. This approach also works well on images with large displacements, something the single-scale algorithm is unable to handle.



Multi-scale coarse-to-fine optical flow estimation

In terms of implementation, first modify your code from part two so that it accepts and refines parameters u_0 and v_0, which corresponds to the initial estimates of the optical flow,

```
OpticalFlowRefine(Img1, Img2, windowSize, u0, v0)
```

The function should refine the optical flow according to,

$$u = u_0 + \Delta u \text{ and } v = v_0 + \Delta v$$

where Δu and Δv represent the offset between the initial estimate and the refined estimate. To compute Δu and Δv, we simply shift the window in Img2 by (u_0,v_0) and compute the optical flow between the shifted windows.

Once `OpticalFlowRefine` is working, you should write another function,

```
MultiScaleLucasKanade(Img1, Img2, windowSize, numLevels)
```

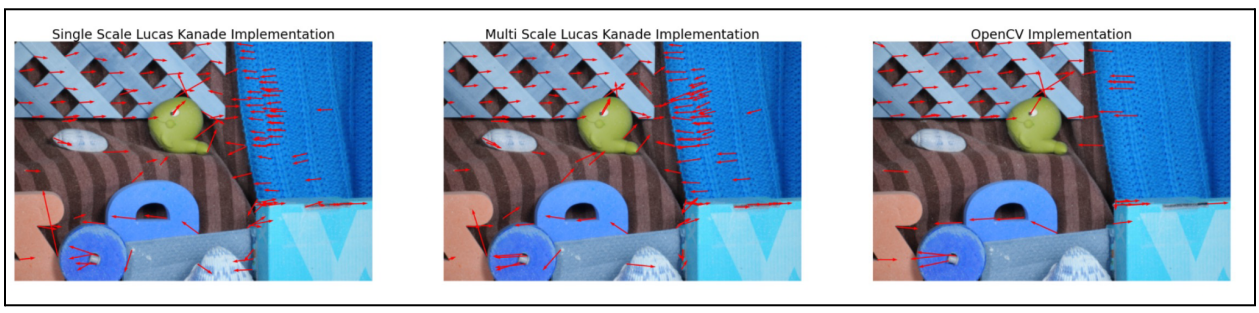
which calls the refinement function. MultiScaleLucasKanade should implement the following pseudo-code,

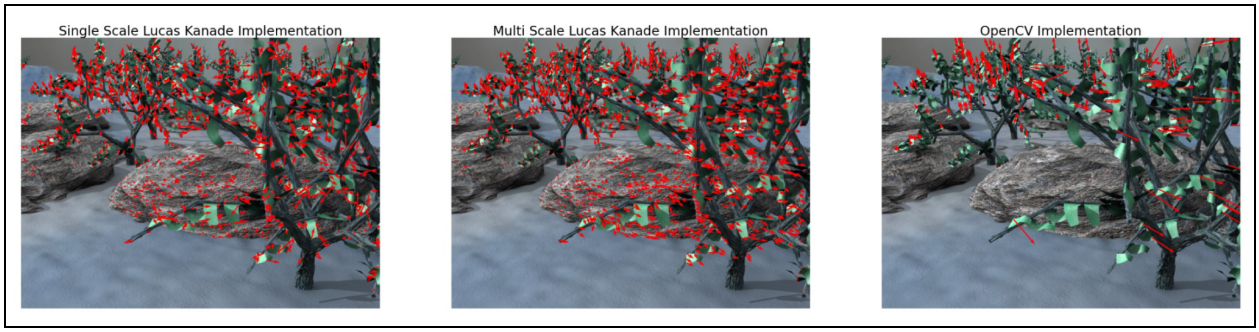
```
Step 01. Gaussian smooth and scale Img1 and Img2 by a factor of 2^(1
- numLevels).

Step 02. Compute the optical flow at this resolution.

Step 03. For each level,

 a. Scale Img1 and Img2 by a factor of 2^(1 - level)

 b. Upscale the previous layer's optical flow by a factor of 2

 c. Compute u and v by calling OpticalFlowRefine with the

     previous level's optical flow
```

Run the algorithm on the image sequences from the Middlebury dataset.

Compare against part two using:

1. EPE Error Values (only on feature points detected by you)
2. Images with the optical flow (from the three image sequences) for single scale LK, Multiscale LK and OpenCV implementation and analyze the results.





Using this analysis, show a case where multi-scale performs better than single-scale. This method is based on the 2001 article, Pyramidal Implementation of Affine Lucas Kanade Feature Tracker, by J. Bouguet.