Assign #02: **The Secrets of Optical Flow**

CS7.505 Computer Vision Spring 2022
International Institute of Information Technology Hyderabad

March 8, 2022

Please stick to the provided function signatures, variable names, and file names. **Start early!** This assignment cannot be completed within two hours. Part four is for extra-credits and hence, optional. You have to submit handwritten solutions for the theory questions, however, the report could be typed. 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 a F grade.**

1 Basics of Optical Flow

Estimating the motion of every pixel in a sequence of images is a problem with many applications in computer vision, such as image segmentation, object classification, and visual odometry. In general, optical flow describes a sparse or dense vector field, where a displacement vector is assigned to certain 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 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 two epic movie clips from two of the Academy Award winning movies for the Best Visual Effects - What Dreams May Come (1998), and 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. You may also find this amazing video on bullet-time interesting.

3. So breathtaking, heartbreaking and brimming with emotion, WDMC is a journey into the afterlife and deals with a dead man's attempt to reunite with his wife. Catch a glimpse of the "Painted World" here! You'll now describe briefly on how optical flow is used to create this "painterly effect".

4. 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 does the 2D motion field and optical flow look like in both cases.

1.2 Concept Review

1. List down the important assumptions made in optical flow estimation. Describe each one of them in one-two lines.

2. Formalize the objective function of the classical optical flow problem. Clearly mark the data term and the spatial term. What does the objective function imply about the noise distribution?

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

4. 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

Implement (i) Harris Corner Detector and (ii) Shi-Tomasi Corner Detector and visualize the feature set of pixels obtained by both algorithms that will be tracked by the sparse LK method. 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, and 1994 CVPR paper, Good Features to Track, by Jianbo Shi and Tomasi.

2.2 Forward-Additive Sparse Optical Flow

Implement the single-scale Lukas-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,

```
LukasKanadeForwardAdditive(Img1, Img2, windowSize)
```

Figure 1: Sample ground truth flow fields and corresponding images from the **MPI-Sintel Flow dataset**.

Figure 2: From left to right: flow representation, horizontal optical flow and vertical optical flow from the MPI-Sintel Flow dataset.

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. In addition to these inputs, you should add a threshold τ such that if τ is larger than the smallest eigenvalue of $A^T A$, then the the optical flow at that position should not be computed. A typical value for τ is 0.01.

You will use all the given image sequences from Middlebery Optical Flow dataset throughout part two to test your algorithm. First, compile the ground truth optical flow into a video to see it. Extract the vertical and horizontal flow and repeat the same. Now, visually compare the output optical flow with the ground truth and compute the Average End Point Error (EPE). Ensure you add the quiver plots superimposed on the images and also, compile it as a video.

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

2.3 Analyzing Lucas-Kanade Method

1. Why is optical flow only valid in the regions where local structure-tensor $A^T A$ has a rank 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. There are locations in an image where Lucas–Kanade optical flow will fail, regardless of choice of window size and sigma for Gaussian smoothing. Describe two such situations.You are not required to demonstrate them.

5. 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.

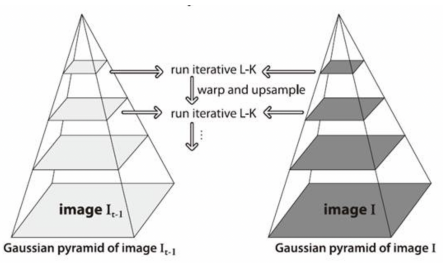

Figure 3: Multi-scale coarse-to-fine optical flow estimation.

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 Middlebery dataset. Compare against part two and analyze the results. 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.

4 Detection and Tracking with Lucas-Kanade

4.1 Inverse Composition: Baker-Matthews Method

Implement the Inverse Compositional Lucas-Kanade tracker with one single template. In the scenario of two-dimensional tracking with pure translation, the problem can be described as follows: starting with a rectangle R_t on frame I_t, the Lucas-Kanade tracker aims to move it by an offset $(\Delta u, \Delta v)$ to obtain another rectangle R_{t+1} on frame I_{t+1}, so that the pixel squared difference in the two rectangles is minimized.

Starting with an initial guess of (u, v), for instance, $(0, 0)$, we can compute the optimal (u^*, v^*) iteratively. In each iteration, the objective function is locally linearized by first-order Taylor expansion and optimized by solving a linear system that has the form $A\Delta p = b$, where $\Delta p = (u, v)^T$, the template offset.

Your algorithm will be implemented as a function with the following signature,

```
LukasKanadeInverseCompositional(Img1, Img2, rect)
```

that computes the optimal local motion from frame I_t to frame I_{t+1} that minimizes the objective function. Here `Img1` is the image frame I_t, `Img2` is the image frame I_{t+1}, and `rect` is the 4×1 vector that represents a rectangle on the image frame I_t. The four components of the rectangle are $[x1, y1, x2, y2]$, where $(x1, y1)$ is the top-left corner and $(x2, y2)$ is the bottom-right corner. The rectangle is inclusive, i.e., it includes all the four corners. To deal with fractional movement of the template, you will need to interpolate the image using OpenCV functions. You will also need to iterate the estimation until the change in (u, v) is below a threshold. Read more about it in the IJCV 2004 paper Lucas-Kanade 20 Years On: A Unifying Framework Part II by S. Baker and I. Matthews.

Visualize, compare and analyze the results of the inverse compositional LK method against the original LK algorithm on the sample images from the Middlebury Optical Flow dataset. It will help you to appreciate the performance improvement of the inverse compositional algorithm.

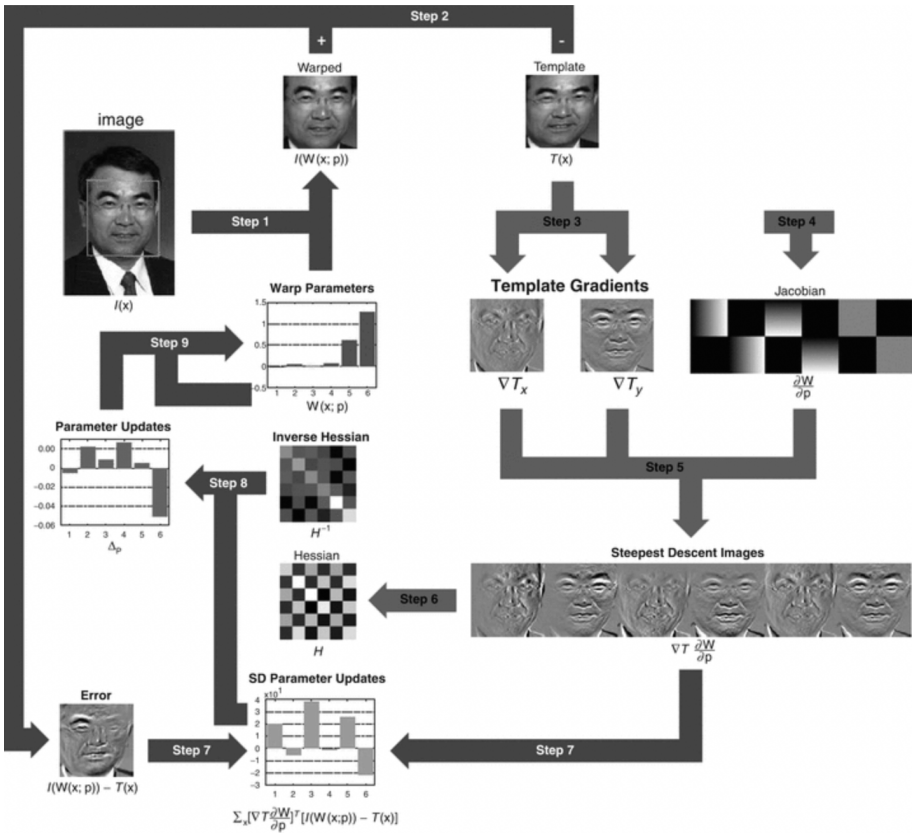

Figure 4: Baker-Matthews alignment *inverse compositional* LK algorithm.

4.2 Tracking Beating Vessel in Ultrasound Volume

Let's apply the your algorithm on a challenging problem: tracking a beating vessel in an ultrasound volume. Unlike regular scenes in the world, medical images acquired by an ultrasound transducer undergo significant non-rigid motion.

Write a function `TrackUsSequence` that loads the video frames from `usseq.mat` , and tracks the beating vessel using the Lucas-Kanade tracker that you have implemented in the previous question. `usseq.mat` contains one single three-dimensional matrix: the first two dimensions correspond to the height and width of the frames respectively, and the third dimension contain the indices of the frames. The rectangle in the first frame is $[x1, y1, x2, y2] = [255, 105, 310, 170]$. Report your tracking performance at frames 5, 25, 50, 75 and 100 with image overlaid with the corresponding bounding rectangle. Use SciPy to load `mat` files.

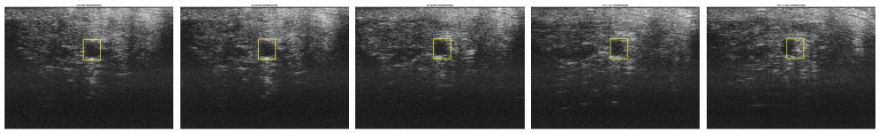

Figure 5: Lucas-Kanade Tracking with one single template for the US sequence.

4.3 Template Drifting and Correction

The image content we are tracking in the first frame differs from the one in the last frame. This tracker lags and loses some tracking capability. This is understandable since we are updating the template after processing each frame and the error can be accumulating. This problem is known as template drifting. There are several ways to mitigate this problem. Implement the template update with drift correction strategy described in the 2003 BMVC paper, The Template Update Problem.

Write a function `TrackUsSequenceWithTemplateCorrection` with similar functionality as `4.2` but with a template correction routine incorporated. Report the performance at the same frames described above. Draw two rectangles in each frame to compare `4.2` and `4.3` visually and compile it as a video.

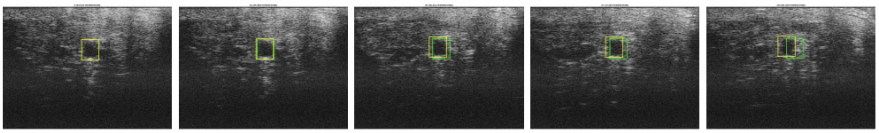

Figure 6: Lucas-Kanade Tracking with template correction for the US sequence.