Implementing Lucas-Kanade's Sparse Optical Flow, Template Tracking and Gunnar-Farneback's Dense Optical Flow

Sreekanth Putta

I. Introduction

Tracking can be defined simply as follows: given a current frame of a video and the location of an object in the previous frame, find its location in the current frame. The Lucas-Kanade sparse optical flow algorithm is a simple technique which can provide an estimate of the movement of interesting features in successive images of a scene. The algorithm does not scan the second image looking for a match for a given pixel. It works by trying to guess in which direction an object has moved so that local changes in intensity can be explained. It assumes that the flow is essentially constant in a local neighbourhood of the pixel under consideration, and solves the basic optical flow equations for all the pixels in that neighbourhood, by the least squares criterion. In this paper, we try to implement Lucas-Kanade's sparse optical flow, extend it to dense optical flow like Gunnar-Farneback's Dense optical flow and Lucas-Kanade's Template Tracking..

II. Related Work

Tracking has been an active area of research since the Lucas Kanade's algorithm was came into light in 1981. David Beymer[2] has developed a feature based tracking approach to avoid partial occlusion for congested traffic. Instead of tracking the entire vehicle, vehicle sub features are tracked to improve the tracking. S.M.Smith[3] has described a sysem for detecting and tracking moving objects in a moving world. The feature-based optic flow field is segmented into clusters with affine internal motion which are tracked over time. David Schreiber[4] introduced a novel optimization-based algorithm for histogram-based tracking which is a generalization of the Lucas–Kanade algorithm. It establishes a closer link between template matching and histogram-based tracking methods

III. Theory

A. Lucas Kanade Algorithm

Lucas-Kanade method computes optical flow for a sparse feature set. The Lucas–Kanade method assumes that the displacement of the image contents between two nearby instants (frames) is small and approximately constant within a neighborhood of the point p under consideration.
The brightness constancy equation[5] is:

$$I_x(p)u + I_y(p)v = I_t(p)$$

where $q_1, q_2, .., q_n$ are the pixels inside the window, and $I_x(p), I_y(p), I_t(p)$ are the partial derivatives of the image I evaluated at point p.
To solve for u and v, we need atleast 2 equations. The optical flow equation can be assumed to hold for all pixels within a window with p as center. For each pixel, we have the following equations,

$$I_x(p_1)u + I_y(p_1)v = -I_t(p_1)$$

$$I_x(p_2)u + I_y(p_2)v = -I_t(p_2)$$

$$\vdots$$

$$I_x(p_n)u + I_y(p_n)v = -I_t(p_n)$$

The equations can be written in matrix form $Ax = b$, where

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_n) & I_y(p_n) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_n) \end{bmatrix}$$

To solve for u and v from the above matrix, Lucas Kanade method used Least squares principle.

$$A^T Ax = A^T b$$

$$x = (A^T A)^{-1} A^T b$$

Solving above yields,

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum I_x(p_i)^2 & \sum I_x(p_i) I_y(p_i) \\ \sum I_y(p_i) I_x(p_i) & I_y(p_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum I_x(p_i) I_t(p_i) \\ -\sum I_y(p_i) I_t(p_i) \end{bmatrix}$$

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \frac{-\sum I_y^2 \sum I_x I_t + \sum I_x I_y \sum I_y I_t}{\sum I_x^2 \sum I_y^2 - (\sum I_x I_y)^2} \\ \frac{-\sum I_x^2 \sum I_y I_t + \sum I_x I_y \sum I_x I_t}{\sum I_x^2 \sum I_y^2 - (\sum I_x I_y)^2} \end{bmatrix}$$

For each feature point, we computer the displace vector (u,v) and then we consider the new point to be the base point to find in the next frame and so on. The (u,v) obtained from above is a vector which defines the movement of the feature in 2 frames. There are some limitations for this algorithm. The Flow over a small patch should be the same. It works for moderate object speeds. It assume that the flow over a small image patch is constant i.e., the constant flow.

B. Dense Optical Flow

In the previous section, we have seen the Lucas-Kanade's sparse optical flow for given set points. The dense optical flow computes the optical flow for all the points in the frame. It is based on Gunnar-Farneback's algorithm[6]. This algorithm uses polynomial expansion to estimate displacement.

The displacement estimation is:

$$d = -\frac{1}{2}A_1^{-1}(b_2 - b_1)$$

We get a 2-channel array with optical flow vectors, (u,v). We find their magnitude and direction. We color code the result for better visualization using flow_vis library. Direction corresponds to Hue value of the image. Magnitude corresponds to Value plane.

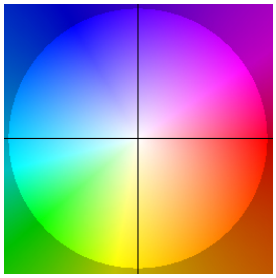

Fig. 1. The optical flow field color-coding. Color represents the direction

C. Template Tracking

Template Tracking is to follow a template image in a video sequence by estimating the warp. The goal of Lucas-Kanade is to align a template image T(x) to an input image I(x), where x = $[x\ y]^T$ is a column vector containing the pixel coordinates.

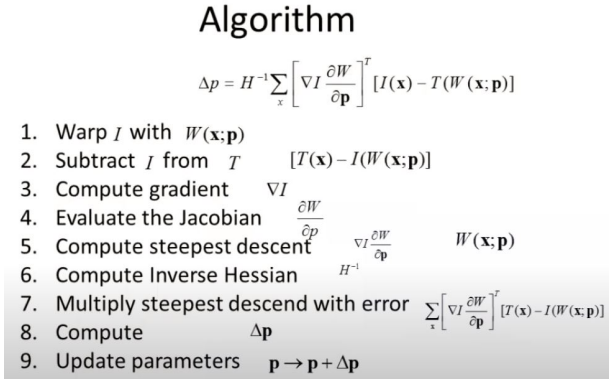

Fig. 2. Lucas Kanade Algorithm

The assumptions of the algorithm are as follows: The entire template is visible in the input image, the intensity of the object appearance is always the same, temporal consistency and spacial coherency.

This method only works for small changes. If the initial estimate is too far, then the linear approximation does not longer hold. Pyramidal implementation would improve the prediction.

IV. Implementation

A. Lucas Kanade Algorithm feature tracking

We take 2 image frames which are captured at a very small difference of time in gray scale. We now use opencv library to find good features in the selected area in the image. I have set the opencv's image view to listen to the selection of a patch in the image using callback function and then we will find the features in the selected patch. These features are sent to the algorithm to track.

With the images obtained, compute I_x, I_y and I_t as follows. I_x and I_y are obtained by convolving the image with vertical and horizontal Sobel filters respectively. For I_t, we will smooth the 2 images with a box filter and subtract one from the other. Using a spatial smoother when computing the time derivative mitigates the effect of noise.

We will now find the displacement vector for every pixel using the following formula:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \frac{-\sum I_y^2 \sum I_x I_t + \sum I_x I_y \sum I_y I_t}{\sum I_x^2 \sum I_y^2 - (\sum I_x I_y)^2} \\ \frac{-\sum I_x^2 \sum I_y I_t + \sum I_x I_y \sum I_x I_t}{\sum I_x^2 \sum I_y^2 - (\sum I_x I_y)^2} \end{bmatrix}$$

We can do the summation of the image gradient of each pixel in an efficient way by using OpenCV library to convolve with a box filter of the desired window size. In this implementation the window size is (15,15). After obtaining the u and v matrices, we can use values of each pixel in them to create a displacement vector and depict the displacement of each pixel on the image.

I have surrounded the features with a minimum possible size rectangle which can fit all the features, to experiment if this can be used to track a patch continuously. This rectangle moves with the features and it changes its size depending on the positions of the features. Please find the video attached to see how it works.

B. Dense Optical Flow

In the previous section, we have implemented a sparse optical flow. This section implements a dense optical flow which means that, all the points in the image will be computed for optical flow. This makes the algorithm run slow. I have implemented this by extending the Lucas Kanade algorithm to compute for all the points in the frame instead of doing it for selected points. This gives me matrices of u and v (horizontal and vertical displacement) which are of same size as the image. We can compute the flow matrix using these values. The flow matrix is a 3 dimensional array which contains direction and angle for every point. This will be of the shape (image.shape[0], image.shape[1], 2). We calculate the magnitude by using hypot function from numpy and the angle by applying tan^{-1} to the vector. We then create a 3d array called flow and append the magnitude and angle data. The magnitude data is depicted as intensity of the pixel and the angle as color from BRG colormap from Fig. 1. Plese find the video attached to see how it works.

C. Lucas-Kanade Template Tracking

I have tried implementing this algorithm but couldn't succeed in doing it entirely by myself. I found an online repository implementing this algorithm and tested it and compared with a learned model.

V. Data

I have downloaded a video of traffic at Shibuya City, Tokyo and have tested my algorithms on it. Some of the algorithms also have a video of me moving around in the frame.

VI. Results

A. Lucas Kanade Algorithm Feature Matching

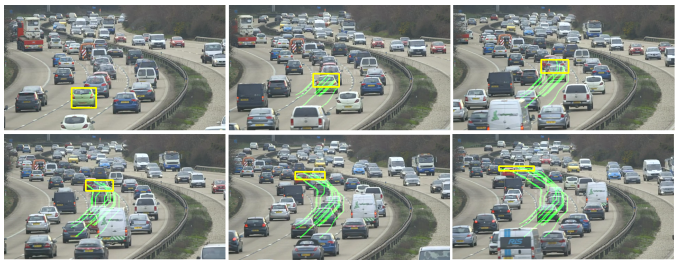

Fig. 3. My implementaion on Shibuya Traffic

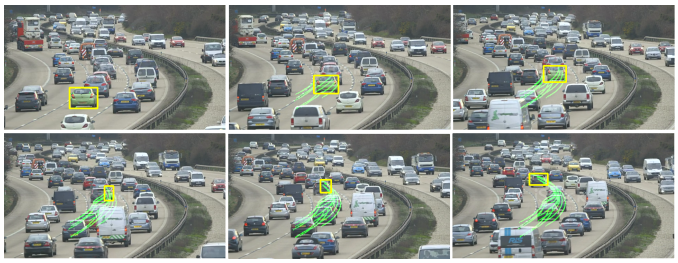

Fig. 4. OpenCV's LK algorithm on Shibuya Traffic

Here I have selected a patch where there is a green car. My algorithm misses track of the car in the middle and starts tracking another car. But the OpenCV's algorithm seems to be working better than my algorithm but it missed track when another car blocked the view and started tracking the new car. Please find the whole video attached. The computing speed of my implementation is 8fps and of the openCV's algorithm is 30fps.

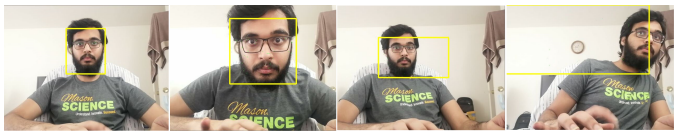

Fig. 5. My implementation to track my face

In this test, I have used my webcamera and moved in the frame to check if the algorithm can track me. The rectangle is just a minimum bounding rectangle containing all the features. This makes the rectangle increase in size abnormally, if a feature moves away from the remaining points due to any incorrect tracking. The features can be visible in green small dots if the image is zoomed in. My implementation could track my face till some extent and couple of features missed track by the last frame. OpenCV's implementation is not accurate and it lost track of some features as well but better then mine and faster.

Fig. 6. OpenCV's LK algorithm to track my face

B. Dense Optical Flow

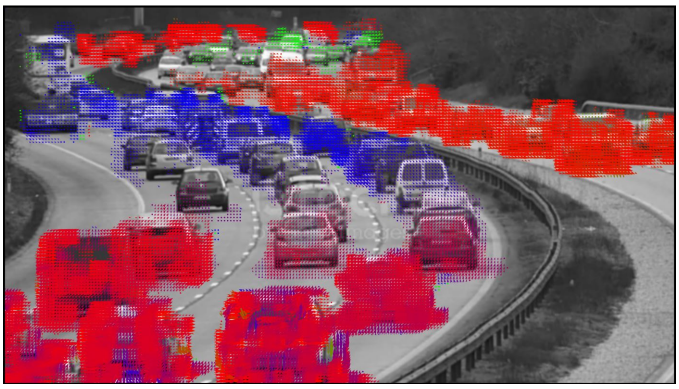

Fig. 7. My implementation of Dense Optical flow with arrows pointing towards the direction of displacement

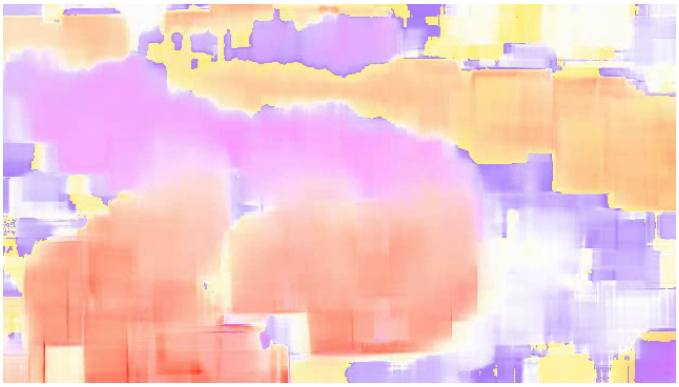

Fig. 8. My implementation of Dense Optical flow

My implementation looks noisy but the movement of the cars can be understood from the video. Computing speed is 8fps but inaccurate.

The OpenCV's implementation looks better and clearly shows the cars' movement. Computing speed is 4fps

I have implemented RAFT[7](A learned optical flow algorithm). Model used is *raft-things.pth*. This took me lot of time to install because all the algorithms I tried to install were giving me errors because I don't have Nvidia graphic card.

Fig. 9. My implementation of Dense Optical flow

Fig. 10. My implementation of Dense Optical flow

Finally I could change the configuration in RAFT to work with just with the CPU. This algorithm much slower than than my implementation and the CV implementation as well. It computes each frame in 10 seconds which is equivalent to 0.1 fps. This algoithm clearly shows the outline of the cars as well with almost accurate direction and speed. Please find the whole video attached.

C. Lucas Kanade Template Tracking

Fig. 11. My implementaion on Shibuya Traffic

Lucas Kanade's algorithm implementation works great. It is expected to loose track when the target is blocked. Computing speed is 15fps. Please find the video attached.

I have used THOR[8](learned model) as my comparison algorithm. I have used *SiamRPNBIG.model* as the model file. This algorithm almost perfectly tracks the car as long is the car is visible. When it got blocked by another car,

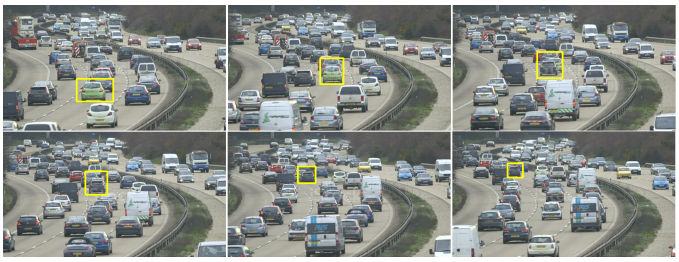

Fig. 12. THOR learned model on Shibuya Traffic

it quickly its focus to the new car ans started tracking it. This algorithm is much slower when compared to the Lucas-Kanade's algorithm. Computing speed is 2.5fps. Please find the video attached.

REFERENCES

[1] B.D. Lucas, T. Kanade, "An Image Registration Technique with an Application to Stereo Vision", in Proceedings of Image Understanding Workshop, 1981, pp. 121-130.

[2] Beymer, D., McLauchlan, P.F., Coifman, B., Malik, J., 1997. A real-time computer vision system for measuring traffic parameters. In: Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR).

[3] S.M. Smith, J.M. Brady Asset-2: Real-time motion segmentation and shape tracking IEEE Trans. Pattern Anal. Machine Intell., 17 (8) (1995), pp. 814-820

[4] David Schreiber, Generalizing the Lucas–Kanade algorithm for histogram-based tracking, Pattern Recognition Letters, Volume 29, Issue 7, 2008 Pages 852-861,

[5] Beauchemin, S. S.; Barron, J. L. (1995). The computation of optical flow. ACM New York, USA.

[6] Farnebäck, G.: Two-Frame Motion Estimation Based on Polynomial Expansion. In: Bigun, J., Gustavsson, T. (eds.) SCIA 2003. LNCS, vol. 2749, pp. 363–370. Springer, Heidelberg (2003)

[7] Teed, Zachary and Deng, Jia: RAFT: Recurrent All-Pairs Field Transforms for Optical Flow. In: Computer Vision – ECCV 2020, pp 402-419. Springer International Publishing

[8] Zhu, Zheng and Wang: Distractor-aware Siamese Networks for Visual Object Tracking. European Conference on Computer Vision (2018)