Optical Flow

Image processing

Timofte Maximillian  
AC-CIT  
Tehnical UniversityCluj-Napoca, Romania  
max.timofte@yahoo.com

*Abstract*—This electronic document is a “live” template and already defines the components of your paper [title, text, heads, etc.] in its style sheet. *\*CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract*. (*Abstract*)

Keywords—component, formatting, style, styling, insert (key words)

# Argument

I have chosen this project because I want to learn the fundamentals of image processing. I hope that this project will increase my interest towards this field, and I will be able to implement more complex projects after that.

# Introduction

Recent breakthroughs in computer vision research have allowed machines to perceive its surrounding world through techniques such as object detection for detecting instances of objects belonging to a certain class and semantic segmentation for pixel-wise classification.

Optical flow is the motion of objects between consecutive frames of sequence, caused by the relative movement between the object and camera .

However, for processing real-time video input, most implementations of these techniques only address relationships of objects within the same frame (x,y) disregarding time information (t). In other words, they re-evaluate each frame independently, as if they are completely unrelated images, for each run.

The following are five methods that I read about that implement corner detection.

# Related Work

## Lucas-Kanade: Sparse Optical Flow

Lucas and Kanade proposed an effective technique to estimate the motion of interesting features by comparing two consecutive frames in their paper . An [Iterative Image Registration Technique with an Application to Stereo Vision](https://ri.cmu.edu/pub_files/pub3/lucas_bruce_d_1981_2/lucas_bruce_d_1981_2.pdf). The Lucas-Kanade method works under the following assumptions:

1. Two consecutive frames are separated by a small time increment (dt) such that objects are not displaced significantly (in other words, the method work best with slow-moving objects).
2. A frame portrays a “natural” scene with textured objects exhibiting shades of gray that change smoothly.

First, under these assumptions, we can take a small 3x3 window (neighborhood) around the features detected by Shi-Tomasi and assume that all nine points have the same motion.

## Farneback Optical Flow

Gunnar Farneback proposed an effective technique to estimate the motion of interesting features by comparing two consecutive frames in his paper [Two-Frame Motion Estimation Based on Polynomial Expansion](http://www.diva-portal.org/smash/get/diva2:273847/FULLTEXT01.pdf).

First, the method approximates the windows (see Lucas Kanade section of sparse optical flow implementation for more details) of image frames by quadratic polynomials through [polynomial expansion transform](http://www.diva-portal.org/smash/get/diva2:302485/FULLTEXT01.pdf). Second, by observing how the polynomial transforms under translation (motion), a method to estimate displacement fields from polynomial expansion coefficients is defined. After a series of refinements, dense optical flow is computed. Farneback’s paper is fairly concise and straightforward to follow so I highly recommend going through the paper if you would like a greater understanding of its mathematical derivation.

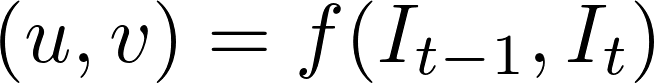


It computes the magnitude and direction of optical flow from a 2-channel array of flow vectors (dx/dt,dy/dt), the optical flow problem. It then visualizes the angle (direction) of flow by hue and the distance (magnitude) of flow by value of HSV color representation. The strength of HSV is always set to a maximum of 255 for optimal visibility.

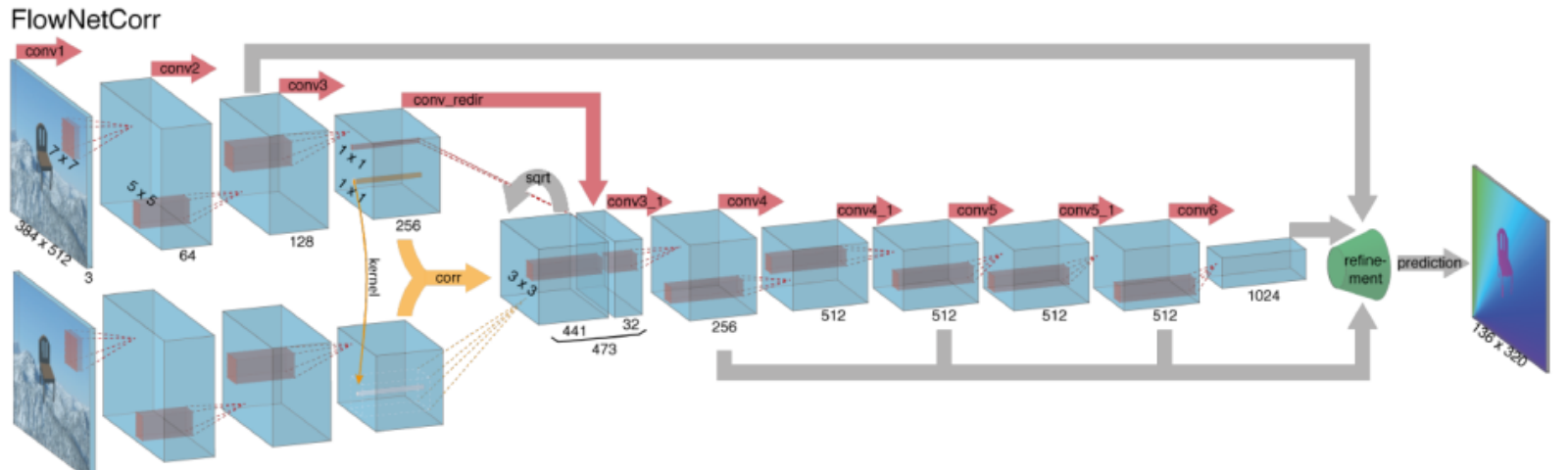
## Optical Flow using Deep Learning

In some cases, one may wish to compute the location of a

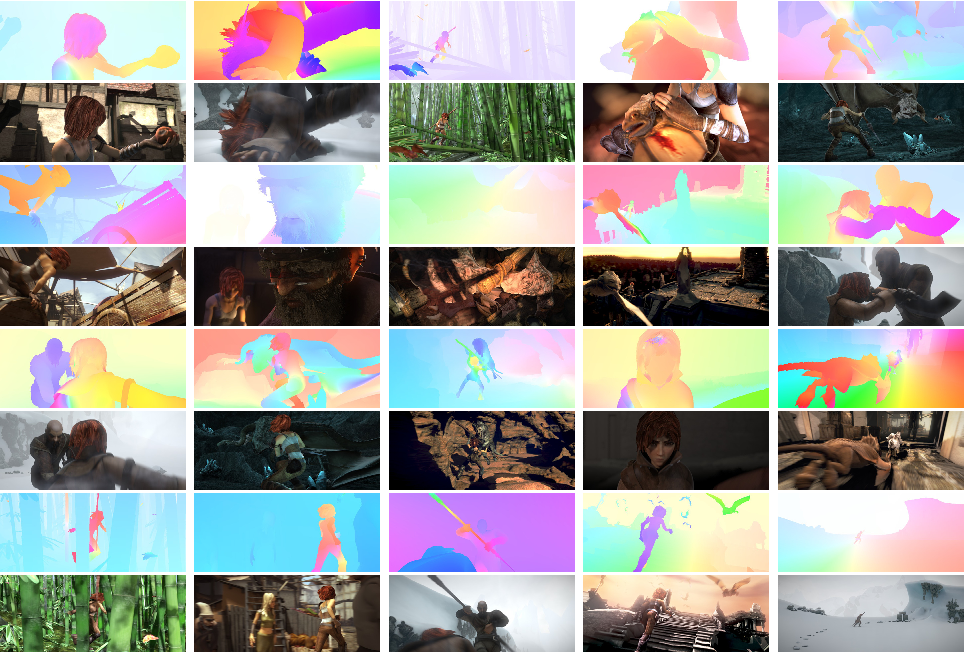
While the problem of optical flow has historically been an optimization problem, recent approaches by applying deep learning have shown impressive results. Generally, such approaches take two video frames as input to output the optical flow (colour-coded image), which may be expressed as:

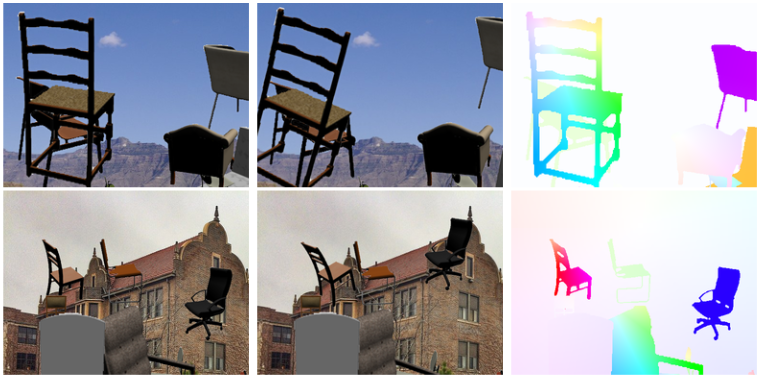


where u is the motion in the x direction, v is the motion in the y direction, and f is a neural network that takes in two consecutive frames It−1 (frame at time = t−1) and It (frame at time = t) as input.



Computing optical flow with deep neural networks requires large amounts of training data which is particularly hard to obtain. This is because labeling video footage for optical flow requires accurately figuring out the exact motion of each and every point of an image to subpixel accuracy. To address the issue of labeling training data, researchers used computer graphics to simulate massive realistic worlds. Since the worlds are generated by instruction, the motion of each and every point of an image in a video sequence is known. Some examples of such include MPI-Sintel, an open-source CGI movie with optical flow labeling rendered for various sequences, and Flying Chairs, a dataset of many chairs flying across random backgrounds also with optical flow labeling.





Solving optical flow problems with deep learning is an extremely hot topic at the moment, with variants of Flow Net, SPY Net, PWC-Net, and more each outperforming one another on various benchmarks.

# Proposed Solution

LUCAS-KANADE: SPARSE OPTICAL FLOW

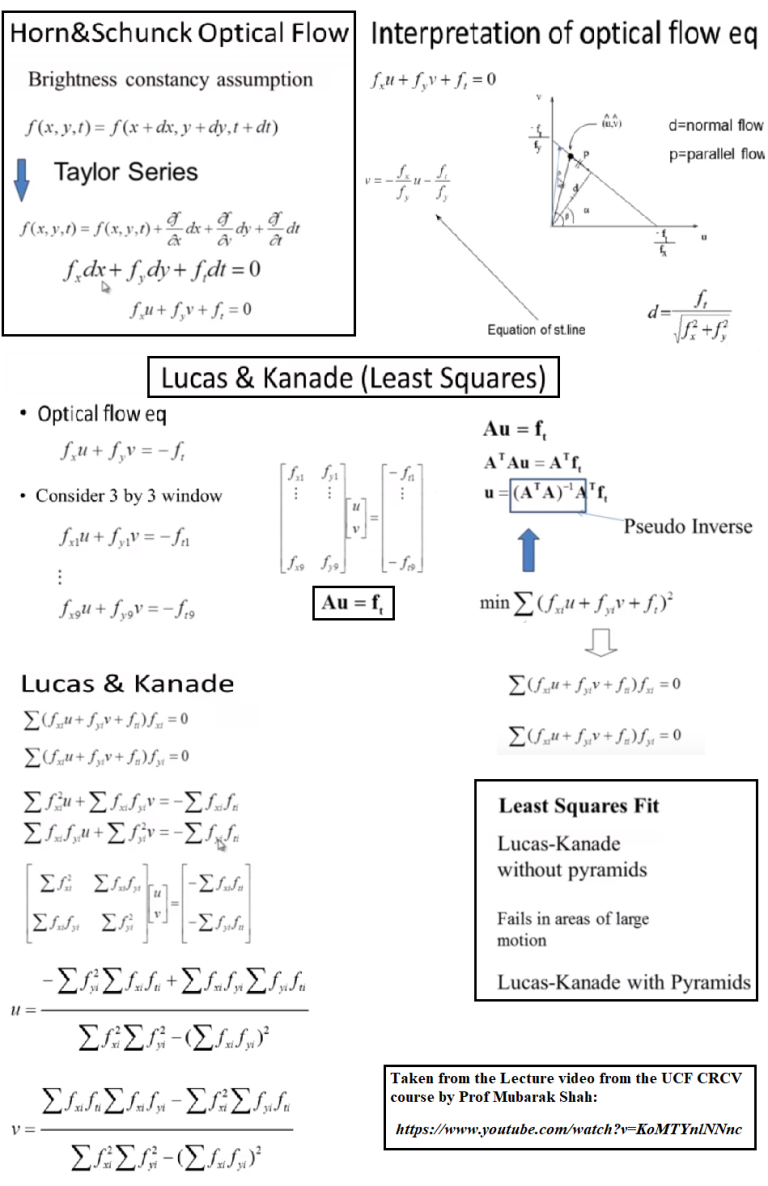
Sparse optical flow selects a sparse feature set of pixels (e.g. interesting features such as edges and corners) to track its velocity vectors (motion). The extracted features are passed in the optical flow function from frame to frame to ensure that the same points are being tracked. There are various implementations of sparse optical flow, including the Lucas-Kanade method, the Horn-Schunck method, the Buxton–Buxton method, and more. We will be using the Lucas-Kanade method with OpenCV, an open source library of computer vision algorithms, for implementation.

We have seen an assumption before, that all the neighbouring pixels will have similar motion. Lucas-Kanade method takes a 3x3 patch around the point. So all the 9 points have the same motion. We can find  for these 9 points. So now our problem becomes solving 9 equations with two unknown variables which is over-determined. A better solution is obtained with least square fit method. Below is the final solution which is two equation-two unknown problem and solve to get the solution.

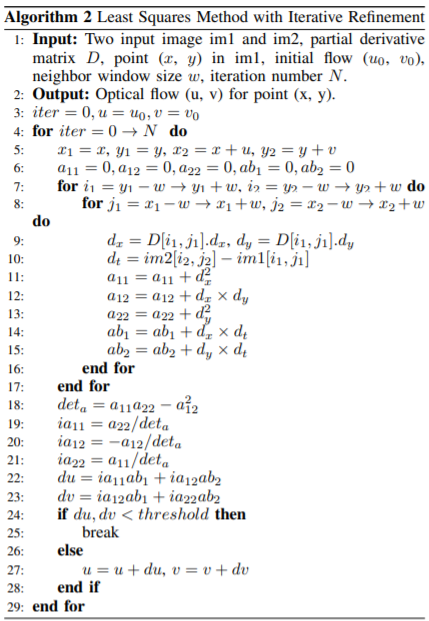
* 1. **Gaussian Pyramid Generation** After applying kernel separation, the procedure of 2D Gaussian blur can be separated to a horizontal 1x7 Gaussian followed by a vertical 1x7 Gaussian. As shown in Figure 1, a 1x7 Gaussian blur involves 7 multiplications and 7 additions for each pixel that can be organized into the SIMD instructions QMPYSP (4-way SIMD multiplication) and DADDSP (2-way SIMD addition) so that the utilization of the computation units on the DSP can be maximized. The last operand of the Gaussian blur is set to be NULL since only 7 computations are needed. In the vertical pass of the Gaussian blur the input pixels are strided across different rows. For this we manually unrolled the outer loop (column loop) by a factor of 2 so that the data from two consecutive iterations can be integrated into a 2-way SIMD load. In each loop iteration, a 7x2 (7 rows, 2 columns) sub-block is loaded into 7 register pairs.

The 7 lower halves and higher halves of the register pairs are processed in the same way as we perform SIMD multiplication and addition on the horizontal Gaussian blur.

* 1. **X and Y Direction Derivative Computation** Two of the derivative values dx and dy can be generated for each pyramid level of the first input image and reused during



the entire procedure. Instead of storing dx and dy value into two matrices, we interleave them into a float2 vector type. This allows dx and dy values to be stored and loaded together and aligns the starting address of each pair of values with 64 bits so that aligned SIMD load and store can be utilized and achieve better memory performance. The optimization of derivative computation is implemented by applying SIMD addition and subtraction. An example of SIMD derivative computation is shown in Figure 2.

* 1. **Corner Least Squares Method** After the generation of the Gaussian pyramid and derivative matrices, the next step is to compute optical flow by least squares method on each level of the Gaussian pyramid. This part of the optimization consumes over 95% of the total execution time. Algorithm 2 shows the most expensive component of the least squares method. It is a 2D loop that requires the computation of the summation of Pd 2 x , Pdxdy, Pd 2 y , P P dxdt, dydt in the neighbor window. This computation is performed on each pixel on all the pyramid levels multiple times (iterative refinement). The dx and dy values are read from the partial derivative matrix, and dt value is computed from subtraction of the corresponding pixel intensity im2[i2, j2] and im1[i1, j1]. Without optimization, four floating point loads (D[i1, j1].dx, D[i1, j1].dy, im2[i2, j2] and im1[i1, j1]) and five floating point multiplications and additions are required for processing each pixel. Algorithm 2 Least Squares Method with Iterative Refinement   
       
       
       
       
     Notice that the pixel intensity im2[i2, j2] and im1[i1, j1] are stored separately in two matrices, and im2[i2, j2] is not a regular data access (the indices are computed from the previous flow update). In order to maximize the memory performance, we manually unroll the loop by a factor of two to enable SIMD load on the two input images. However since the starting address of im2[i2, j2] is not predictable, we must use the unaligned SIMD loads for them. Correspondingly, two pair of dx and dy are read by aligned SIMD loads. On the C66, two aligned SIMD loads can be packed into a parallelized VLIW instruction and issued in one cycle, but when unaligned cannot be parallelized. Three cycles are required for each memory operation.
  2. Multicore Utilization  
     The master core (core 0) initializes data structure, loads the images, and generates Gaussian pyramids and derivative matrices. Then, eight cores will begin performing the least squares method for the first level of the Gaussian pyramid. The workload is uniformly distributed along the rows. A synchronization is performed after each pyramid level is finished, then core 0 will interpolate the Optical Flow field onto the next level of the pyramid and then all the cores begin the least squares computation on the next level.

##### References

1. https://en.wikipedia.org/wiki/Lucas%E2%80%93Kanade\_method
2. https://www.youtube.com/watch?v=1r8E9uAcn4E
3. https://www.youtube.com/watch?v=TcnVeK6YjUc
4. http://www.cse.psu.edu/~rtc12/CSE486/lecture30.pdf
5. http://www.inf.fu-berlin.de/inst/ag-ki/rojas\_home/documents/tutorials/Lucas-Kanade2.pdf
6. https://www.semanticscholar.org/paper/Optical-Flow-Measurement-using-Lucas-kanade-Method-Patel-Upadhyay/48310d8abf7f3d1f3ac9989edb07aa67e949b0cf
7. https://homes.cs.washington.edu/~shapiro/EE596/notes/Optical\_Flow.pdf
8. https://opencv-python-tutroals.readthedocs.io/en/latest/py\_tutorials/py\_video/py\_lucas\_kanade/py\_lucas\_kanade.html
9. http://www.ieee-hpec.org/2014/CD/index\_htm\_files/FinalPapers/98.pdf