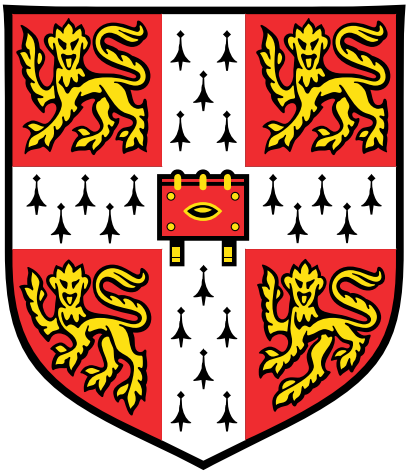


Engineering Department
University of Cambridge

Milestone Report

Low cost imaging and image processing with the Raspberry Pi

Author:
Jamie Magee

Supervisor:
Dr. Alexandre Kabla

January 16, 2014

Contents

1 Abstract

Optical flow is a critical tool in many experimental methods such as Particle Image Velocimetry [1] for flow analysis and and soil mechanics, speckle tracking echocardiography [2] in medical imaging, as well as mechanical testing [3]. During my project I have investigated various different optical flow algorithms, and selected the Farnebäck algorithm for use on the limited hardware of the Raspberry Pi. I have implemented each algorithm in C++, as well as several different output options. Continuing next term I would like to implement curl and div calculation, and aim to provide online processing as well.

2 Objectives

My main objectives for the project are to:

- Provide access to optical flow methods on cheap and readily available hardware
- Perform online and offline processing using the Raspberry Pi
- Capture data using the Raspberry Pi Camera
- Integrate with existing experiments and other OpenLabTools projects
- Provide a clear set of instructions aimed at undergraduate level

3 Tools

To aid me in implementing image processing algorithms, including optical flow methods, I made use of OpenCV [4]. A precompiled binary does not exist for the Raspberry Pi, so it was necessary to compile it myself and write instructions on how to do so. The compilation time is around 10-12 hours depending on the overclock used on the Raspberry Pi. Also included are instructions on how to allow the Raspberry Pi camera to interface with the operating system. An extra driver (UV4l-raspi [5]) is required, however a precompiled binary is available in this case. I have also written up instructions on how to install SimpleCV, a Python wrapper for OpenCV. I have not made use of SimpleCV due to the performance benefits of C++ over Python, however the instructions are included as they will be provided on the OpenLabTools website.

4 Optical Flow Methods

Optical flow methods mostly fall under one of two categories: dense optical flow and sparse optical flow. Dense optical flow methods track all the pixels in a frame, whereas sparse optical flow methods merely track a few points in an image. For my comparison I will consider three different methods: Lucas-Kanade [8], Farnebäck [9] and SimpleFlow [10].

4.1 Lucas-Kanade

The Lucas-Kanade method is a well established, sparse optical flow method. In the one dimensional case it can be thought of as finding the horizontal displacement, δ, between the curves $F(x)$ and $G(x) = F(x + \delta)$. If we assume that δ is small and that $F(x)$ is approximately linear in the area of x then we can say:

$$\begin{aligned} F(x) &\approx \frac{F(x+\delta) - F(x)}{\delta} \\ &= \frac{G(x) - F(x)}{\delta} \end{aligned}$$

Therefore,

$$\delta = \frac{G(x) - F(x)}{F(x)}$$

We can generalise this to two dimensions using linear algebra [11]

$$\begin{aligned} I_x(q_1)V_x + I_y(q_1)V_y &= -I_t(q_1) \\ I_x(q_2)V_x + I_y(q_2)V_y &= -I_t(q_2) \\ &\vdots \\ I_x(q_n)V_x + I_y(q_n)V_y &= -I_t(q_n) \end{aligned}$$

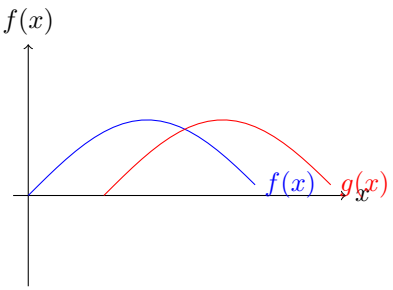


Figure 1: The 1-D case

Where $q_1, q_2, \ldots, q_n$ are pixels inside the window centred around the pixel under consideration, $I_x(q_i), I_y(q_i)$ and $I_t(q_i)$ are the partial derivatives of the image I with respect to position x, y and t respectively, and V_x and V_y are the velocity vectors in the x and y directions respectively. This can be written in Matrix form $Av = b$ where:

$$A = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \vdots & \vdots \\ I_x(q_n) & I_y(q_n) \end{bmatrix}, \qquad v = \begin{bmatrix} V_x \\ V_y \end{bmatrix}, \quad \text{and} \quad b = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \\ -I_t(q_n) \end{bmatrix}$$

We can therefore solve this using the least squares method to find the velocity vector (V_x, V_y)

$$\begin{aligned} Av &= b \\ A^T Av &= A^T b \\ v &= (A^T A)^{-1} A^T b \end{aligned}$$

4.2 Farnebäck

Farnebäck optical is classified as a dense optical flow method. Using this method each neighbourhood of pixels is approximated by a quadratic polynomial of the form:

$$f(x) \approx x^T Ax + b^t x + c$$

In The one dimensional case is shown in figure 2.

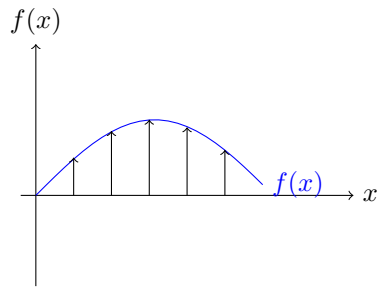


Figure 2: The 1-D case

For the general case, if we consider the polynomial $f_1(x) \approx x^T A_1 x + b_1^T x + c_1$ which is shifted by a displacement δ

$$\begin{aligned} f_2(x) &= f_1(x - \delta) \\ &= (x - \delta)^T A_1 (x - \delta) + b_1^T (x - \delta) + c_1 \\ &= x^T Ax + (b_1 - 2A_1\delta)^T x + \delta^T A_1 \delta - b_1^T \delta + c_1 \\ &= x^T A_2 x + b_2^T x + c_2 \end{aligned}$$

We can therefore equate the coefficients of $f_1(x)$ and $f_2(x)$ which yields

$$
\begin{aligned}
A_2 &= A_1 \\
b_2 &= b_1 - 2A_1\delta \\
c_2 &= \delta^T A_1 \delta - b_1^T \delta + c_1
\end{aligned}
$$

Thus if A_1 is non-singular we can solve for the translation δ

$$
\begin{aligned}
b_2 &= b_1 - 2A_1\delta \\
2A_1\delta &= -(b_1 - b_2) \\
\delta &= \frac{1}{2}A_1^{-1}(b_2 - b_1)
\end{aligned}
$$

4.3 SimpleFlow

SimpleFlow is a combination of both dense and sparse optical flow, it only runs a full flow estimation at a few key pixels, and linearly interpolates the rest of the flow, thus it is classified as *semi-dense* optical flow. At the key pixels a likelihood model is used, that is, we seek flow vectors (u, v) such that $F_t(x, y)$ and $F_{t+1}(x + u, y + v)$ are similar. This is modelled by the energy term:

$$
e(x, y, u, v) = \|F_t(x, y) + F_{t+1}(x + u, y + v)\|^2
$$

5 Comparison

For the comparison of the three optical flow methods listed previously, I implemented them using OpenCV in C++.

5.1 Artificial Data

In order to test the methods I generated sample sequences for affine transformations. The transformations include: scale, shear, translate, rotate and fracture/disjoint. Each lasts for ten frames, and uses the MATLAB `checkerboard` pattern.

(a) Original (b) Translate (c) Rotate (d) Scale (e) Skew (f) Disjoint

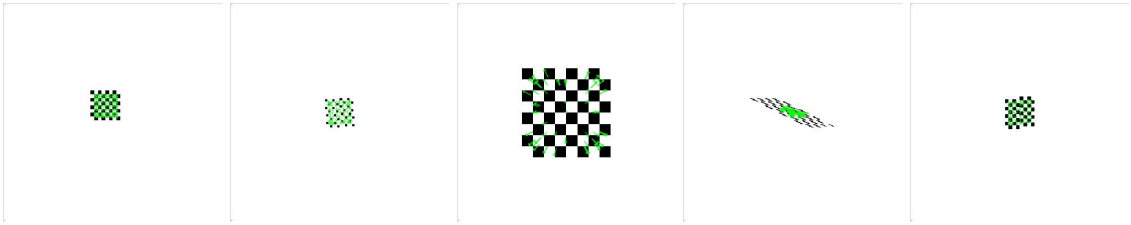
Figure 3: Artifical dataset

I was making a comparison between the different optical flow algorithms - Lucas-Kanade, Farnebäck, and Simpleflow - in order to be able to make an informed decision as to which one I would choose to continue to develop and work with for the remainder of the project. I compared the algorithms in terms of accuracy and speed. Speed is easily classified by measuring the time taken to process a frame of each of the videos. In order to ensure the reliability of the results, each test sequence was run three times, and the processing time was measured as an average across the three runs over the ten frames of each test video. Accuracy was measured qualitatively by comparing the output from the algorithms with prior knowledge of the motion in the test sequences. The test sequences were chosen as they each contain only one affine transformation, therefore making measuring the accuracy easier.

Figure 4 shows the output of the optical flow algorithm for each of the test sequences. As the algorithm is a sparse optical flow algorithm it relies on a feature detector to select the points to track. In some cases, such as the skew in Figure 4d, the feature detector selects points which cluster around the centre of the frame and therefore provides suboptimal results.

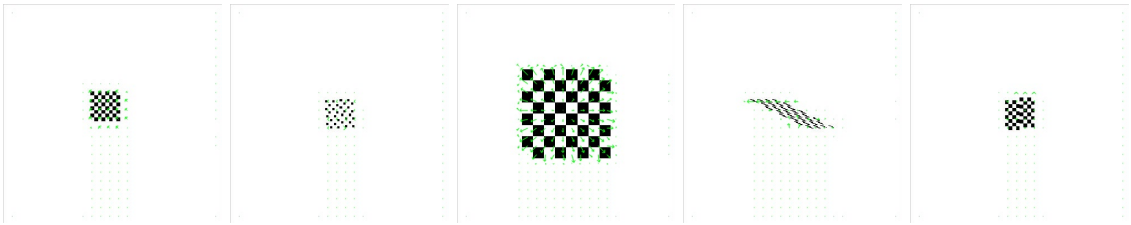
Figure 5 shows the output of the optical flow algorithm for each of the test sequences. Farnebäck optical flow is a dense optical flow algorithm, and therefore tracks all points from frame to frame. As a result it more accurately tracks the motion of the test sequences, even for a more difficult sequence such as Figure 5d.

The SimpleFlow optical algorithm has a runtime two orders of magnitude slower than that of Farnebäck optical flow, and three orders of magnitude slower than Lucas-Kanade optical flow. This is most likely due to the algorithm being designed to be parallelised across many GPUs which are optimised to perform floating point operations. The speed of the algorithm therefore makes it practically unfeasible for use on the Raspberry Pi



(a) Translate (b) Rotate (c) Scale (d) Skew (e) Disjoint

Figure 4: Lucas-Kanade on Artificial Data



(a) Translate (b) Rotate (c) Scale (d) Skew (e) Disjoint

Figure 5: Farnebäck on Artificial Data

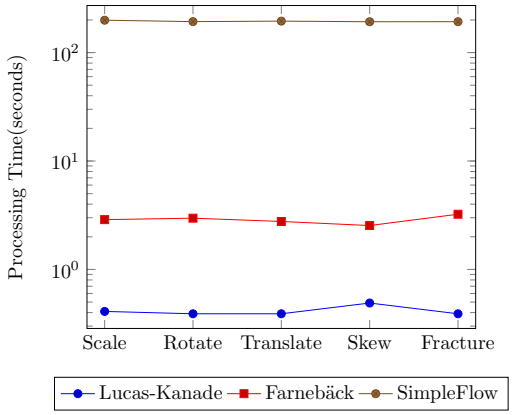


Figure 6: Processing Time Comparison

Figure 6 shows the processing time for each algorithm for each data set. Lucas-Kanade is consistently faster at processing than both Farnebäck and SimpleFlow. The only point of note is the result for the skew data set. the Lucas-Kanade algorithm processed the data set notably slower than the other data sets, whereas the Farnebäck algorithm processed the data set notably faster. There was no discernible difference for the SimpleFlow algorithm.

Based on the above results, the Lucas-Kanade algorithm is consistently the fastest, and the Farnebäck algorithm is consistently the most accurate. The SimpleFlow algorithm provides no advantages on the Raspberry Pi hardware. Given that the speed difference between Lucas-Kanade and Farnebäck is small, I would suggest the Farnebäck algorithm

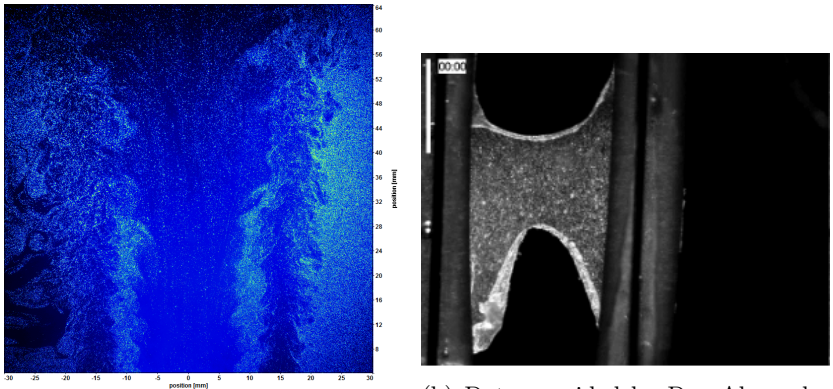
5.2 Real World Data

In order to examine the algorithms in a real world scenario I was graciously provided with data sets from Dr Alexandre Kabla [3], my supervisor, and Mustafa Kamal, A PhD student in the Hopkinson Lab. Samples from both data sets can be seen in figure 7

I measured the average time to process a frame from each set using each algorithm. The results are plotted on a semi-log scale in figure 8.

With regards to the usefulness of the output, while the Lucas-Kanade algorithm is consistently faster than the others, it is a sparse optical flow algorithm and therefore depends heavily on a separate feature detection algorithm to provide useful features to track. Some of the more popular algorithms include: the Harris edge detector [12], the Minimum Eigenvalues Algorithm [13] or the Canny Edge Detector [14]. Obtaining information for key points may be useful for initial analysis of data, most use cases will require a full analysis of each frame. For this purpose only the Farnebäck algorithm is suitable.

As can be seen in figure 8 the SimpleFlow algorithm is approximately two orders of magnitude slower than the others. This is most likely due to the algorithm being designed for parallel processing, and the linear interpolation. The Raspberry Pi only has a single core ARM CPU, compared with high end GPUs which may have several thousand cores which can operate at several TFLOPs. The Lucas-Kanade algorithm is clearly the fastest algorithm, however



(a) Data provided by Mustafa Kamal Kabla

(b) Data provided by Dr. Alexandre

Figure 7: Real world data sets used

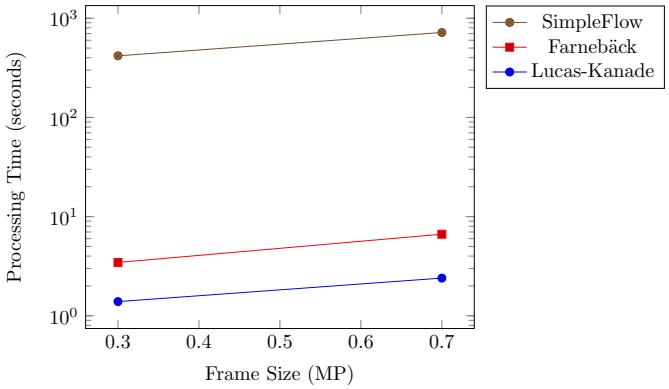


Figure 8: Processing time comparison

it does not describe the entire flow. Despite the Farnebäck algorithm being slower than the Lucas-Kanade, it does provide us with a complete description of the flow, and for the reasons listed above I will recommend the Farnebäck optical flow algorithm.

6 Output

6.1 Vector Arrows

In order to aid the visualisation of the vector field, I draw the vectors calculated from the optical flow methods. The method I wrote is similar to the MATLAB function `quiver` in that it automatically scales the arrows depending on the magnitude of the vector. An example of the output can be seen in figure 9

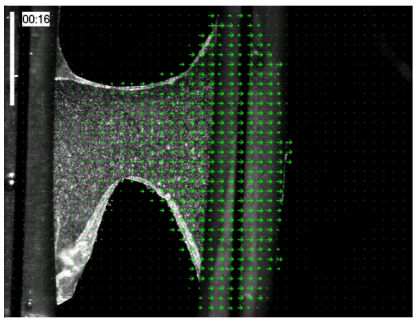
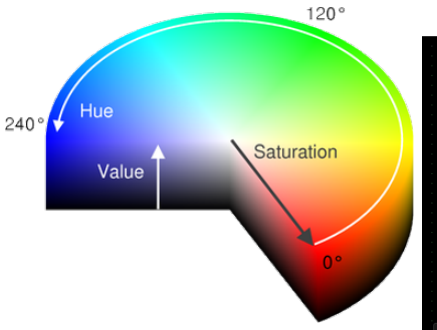


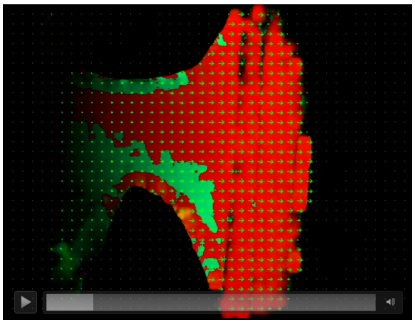
Figure 9: Vector arrows overlaid onto the data set

6.2 Colour Map

The vector field can also be represented using a colour map. This is achieved by converting the cartesian vector coordinates into polar coordinates which are then mapped onto the HSV colour space as: Hue = Angle, Saturation = 255, Value = Magnitude. This can be seen in figure 10



(a) Mapping of vectors to HSV colour space



(b) An example mapping

Figure 10: Mapping the vector field to the HSV colour space

7 Improvements

During lent term I would like to implement the curl and the divergence of the calculated vector field. If the vector field represents the flow velocity of a moving fluid, then the curl is the circulation density of the fluid and the divergence measures the magnitude of a vector field's source or sink at a given point. I will also optimise the implementation of my chosen optical flow algorithm in order to achieve as great performance as possible. This will lead onto integration with experiments, such as the 1st year materials testing lab. Finally, I will improve the user experience by providing a GUI for immediate user feedback.

8 References

[1] Georges M Quénot, Jaroslaw Pakleza, and Tomasz A Kowalewski. "Particle image velocimetry with optical flow". In: *Experiments in fluids* 25.3 (1998), pp. 177–189.

[2] M Bertrand et al. "Ultrasonic biomechanical strain gauge based on speckle tracking". In: *Ultrasonics Symposium, 1989. Proceedings., IEEE 1989*. IEEE. 1989, pp. 859–863.

[3] Andrew R Harris et al. "Characterizing the mechanics of cultured cell monolayers". In: *Proceedings of the National Academy of Sciences* 109.41 (2012), pp. 16449–16454.

[4] *OpenCV | OpenCV*. [Online; accessed 15-December-2013]. URL: `http://opencv.org/`.

[5] *Linux Projects - Documentation*. [Online; accessed 15-December-2013]. URL: `http://www.linux-projects.org/modules/sections/index.php?op=viewarticle&artid=14`.

[6] *SimpleCV*. [Online; accessed 15-December-2013]. URL: `http://simple.org/`.

[7] *OpenLabTools*. [Online; accessed 15-December-2013]. URL: `http://openlabtools.org/`.

[8] Bruce D. Lucas and Takeo Kanade. "An Iterative Image Registration Technique with an Application to Stereo Vision". In: *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*. IJCAI'81. Vancouver, BC, Canada: Morgan Kaufmann Publishers Inc., 1981, pp. 674–679.

[9] Gunnar Farnebäck. "Two-Frame Motion Estimation Based on Polynomial Expansion". In: *Proceedings of the 13th Scandinavian Conference on Image Analysis*. LNCS 2749. Gothenburg, Sweden, 2003, pp. 363–370.

[10] Michael W. Tao et al. "SimpleFlow: A Non-iterative, Sublinear Optical Flow Algorithm". In: *Computer Graphics Forum (Eurographics 2012)* 31.2 (May 2012). URL: `http://graphics.berkeley.edu/papers/Tao-SAN-2012-05/`.

[11] Bruce D. Lucas. "Generalized Image Matching by the Method of Differences". PhD thesis. Robotics Institute, Carnegie Mellon University, 1984.

[12] Chris Harris and Mike Stephens. "A combined corner and edge detector." In: *Alvey vision conference*. Vol. 15. Manchester, UK. 1988, p. 50.

[13] Jianbo Shi and Carlo Tomasi. "Good features to track". In: *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*. IEEE. 1994, pp. 593–600.

[14] John Canny. "A computational approach to edge detection". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 6 (1986), pp. 679–698.