

Cairo University

Faculty of Engineering

Department of Computer Engineering

**Feature Extraction Module 2**

**Horn & Schunck**

Report to explain the approach we took to make a feature extraction to group of frames

**Introduction**

We need a way to calculate the optical flow of a frame – a vheclie – to be able to describe its flow in a fast and good accuracy so we can later process it in the vif descriptor module.

**Different Algorithms**

We searched in the most known algorithms in making optical flow and what we found appropriate to our case are two algorithms: Horn & Schunck and lucas kanade method,

By comparing these two methods together we determined that we use Horn & Schunck method over what we implemented earlier lucas kanade, that is because lucas kanade as we found is slow and it depends on feature extraction of corners in the image, and that’s it’s local optical flow but Horn & Schunck is global optical flow.

**Horn & Schunck Algorithm**

The proposal of Horn and Schunck consists in formulating the problem of optical flow estimation as a variational problem, where the desired vector field h is defined as the minimizer of a certain energy functional J(h). This functional has two terms: a data attachment term, given by the optical flow constraint, and a regularity term that is based on the gradient of the flow:



where α is a parameter to control the weight of the smoothness term compared to the optical flow constraint. The parameter α is squared so that its units are units of grey-level, and it can be regarded as the intensity of an additive Gaussian noise present in the input images.

This energy model uses quadratic functionals in both terms. This assumes that the image noise and the flow derivatives are expected to follow a Gaussian distribution. A direct consequence of this kind of functionals is that the method is very sensitive to the presence of noise and the computed flow fields are very smooth.

These shortcomings have led to the appearance of many research works that try to deal with these limitations.

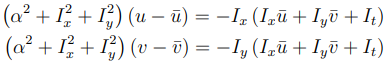
The minimization of the above functional yields the following Euler-Lagrange equations:



The Laplacian can be approximated with the following expressions, which will be useful for the discretization below:



where u and v bar are local averages of (u, v). This approximation is analogous to the commonly used “difference of gaussians”, where the Laplacian operator is approximated by the difference of blurred versions of the image, in this case the smallest blur is zero. Solving the equations above for (u, v) and re-arranging the terms, we obtain the following system of equations:



Writing these equations for each pixel of the input images, we obtain a sparse system of linear equations. This system can be solved efficiently with an iterative scheme.

The partial derivatives, Ix, Iy and It, are approximated using forward differences and averaging between two consecutive frames:







The local averages (¯u, v¯) are estimated from the eight neighbors of (u, v) as:





The coefficients of this discretization are the same as in the original paper by Horn and Schunck. In order to have a correct discretization of the Laplacian, these should be chosen so that the sum of coefficients are equal to the coefficient associated with

(u, v). The solution of the above sparse system of linear equations can be obtained by means of the following iterative scheme:





**Implementation**

Here are the steps that we implemented in the program for feature extraction module “Horn & Schunck”:-

1. Pass these inputs to the function

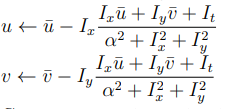
- frame(t-1)

- frame(t)

- alpha: regularization constant

- NumOfIter: number of iteration to increase accuracy

1. Compute the derivatives in fx, fy, ft
2. Initialize horizontal change matrix, vertical change matrix
3. Average flow vector to get h bar and v bar
4. Compute the new u and v



1. Go back to step 4 until numOfIter ends
2. Calculate the magnitude of the horizontal and vertical change
3. Return horizontal, vertical and magnitude

**Processing Time Takes**

As we don’t take the whole frame to process then the processing time differes but on averge it takes a [7 to 18] ms and we hope this speed don’t affect on us when we attached later to the optical flow.

**Comparison to Opencv Method**

We compared our implementation to opencv method and the accuracy was very close you can’t even determine which is better, but on the other side the speed of opencv method is much faster as results showed it’s faster than our implementation - even after optimizing it - with [60 – 80]% faster than us and that is because their function is implemented by c/c++ which is already compiled.