CONTROL AND MONITORING OF A PATIENT IN A PROTON THERAPY SESSION

Alex Vallejo Miracle

A Thesis Submitted in Partial Fulfilment of the Requirements for the Degree of

Audiovisual Systems Engineering

Universitat Politècnica de Catalunya. Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona (UPC. ETSETB)

Université catholique de Louvain. Institute of Information & Communication Technologies, Electronics and Applied Mathematics (UCL. ICTEAM)

February 2014

Abstract

The main objective of that thesis is the implementation of a motion detection system within software *Medusa,* from the *Centre de Protonthérapie d'Orsay (CPO, Paris),* used for controlling a proton therapy session. The solution is able to detect when motion occurs due to the movement of the patient and to decide properly when to stop the treatment if necessary.

In order to do this, different motion detection and object tracking techniques based on optical flow, motion history images, background subtraction, filtering and finding objects have been selected, exhaustively studied and finally implemented existing the chance to choose the most convenient one.

Per a la meva família, amics, i en especial
per a una persona que ens va deixar fa uns mesos,
però que ha sigut i és el meu motor
en aquest projecte.

Para mi familia, amigos, y en especial
para una persona que nos dejó hace unos meses
pero que ha sido y es mi motor
en este proyecto.

For my family, friends, and especially
for a person who left us several months ago
but has been my strongest engine
in this project.

ACKNOWLEDGEMENTS

I would like to thank my supervisor Jonathan Orban for guiding the thesis and helping me, always with a point of humor and confidence. Thank you to Michel Auger for involving CPO in the project and giving me the opportunity to work in Paris and to improve Medusa. Thank you both for trusting me. Thank you to Benoit Macq too for representing my legal part at UCL.

Thank you to all the iMagX staff and especially office 1.13 and surroundings for making the work lighter, for the daily laughs and stories and for teaching me French although you have not noticed. I will never forget you.

Thank you to my parents and the rest of my family. Thank you for always supporting me, advising me and being close. Absolutely nothing would be possible without you. You make everything so much easier.

Thank you to my people in my hometown for telling me every day what is going on and making me feel close even at 1400 km away from home.

To mention, of course, all new friends I met during these months in Louvain-la-Neuve. You all are awesome and you have been the key for disconnecting from work and spending some wonderful time. Thank you guys for all the circle nights, for making funnier the first crazy adaptation days and for all the trips and stories. I will always remember LLN time. It has been a big pleasure and I hope we will see each other soon.

And last but not least, I want to give a big thank you to my aunt and godmother Charo. You have really been my strongest motivation in this project and you have given me all the strength needed to do this. Though probably you do not know it, you have helped me a lot in all ways. Wherever you are, I will always remember you. Thank you so much.

CONTENTS

CHAPTER 1

MOTIVATION

We live in a world in constant changes and evolution. Lots of questions exist and we try to get answers for each one. Researching is an important field which leads us to create technology for different needs that we have. An important sector in which lots of industries, corporations, governments and institutions are focused and which reflects perfectly this necessity of improvement is the medical and health sector.

Many of new systems are emerging in medical applications due to digitalization and the new working techniques that exist at present and the ones that will appear at future. In order to reach some objectives medicine appeals to engineering. Specifically, the electrical engineering and his signal processing field are closely linked to this sector. The fact of being able to analyse some data by an easier way or doing some treatment or another kind of task more exactly and precise is a big advantage for the therapists as well as produces a general improvement of the results.

The proton therapy is a type of particle therapy which uses a beam of protons to irradiate a desired zone. It is used in the treatment of cancer. The main advantage is the possibility to more specifically and precisely localize the radiation dosage in comparison to other radiotherapy techniques.

The treatment is done in a special proton therapy centres which have special treatment rooms. Specialists and therapists control every moment of the session doing the previous adjustments needed and the later monitoring during the beam irradiation. Depending on where is the tumour located a room or another and different stuff is used. Cameras focus the part of the patient's body that it is going to be treated and they allow controlling the sessions.

Particle-based treatments are very precise (millimetre precision). The patient must not move during the irradiation, even for a few millimetres. The *Centre de Protonthérapie d'Orsay* (*CPO*) has designed a tool to monitor patient movements during treatment. The monitoring is done manually with a contours definition as support for therapists for detecting the movement. When the patient moves, it goes out of position (relating to the contours) and then the therapists stop the beam in order to not irradiate non-wanted zones.

An automatic detection of movement would bring some advantages like less observer difference (the software would detect motion independently of the operators working), better detection of small movements and so much faster reaction.

SCOPES

Medusa is software used by the *Centre de Protontherapie d'Orsay (CPO, Paris)* for monitoring and controlling the proton therapy sessions. It is also responsible of the contour detection which is so important since it is used for controlling the patient's movement. It uses the *OpenCV* signal processing libraries in order to accomplish all its functions.

The main purpose is to create a motion detection system that is able to detect the motion of the image, i.e. the movements of the patient, and display a warning signal when the motion is relevant, i.e. when the patient moves over a kind of threshold. If this is working properly, next step is to automatically stop the beam, fact that, nowadays, without this proposed system is executed manually by the therapists.

Therefore, the first stage of the work is focused on studying different motion detection techniques. Motion detection is the process of detecting changes in an sequence of images. It is computed between two frames and it can be done both in video files and in real-time applications. Many methods exist for doing this detection and the most suitable ones are used depending on every case. The main motion detection techniques are those based in optical flow computation, background subtraction, motion history detection, Kalman-based filtering and shape and color tracking. A study of the methods which are available in *OpenCV* was made and is detailed in chapter three. These studied methods are the following:

- The Lucas-Kanade optical flow algorithm.
- The Dual TV-L1 optical flow algorithm.
- The SimpleFlow optical flow algorithm.
- The Gunnar Farnebäck's optical flow algorithm.
- The Motion History Images method.
- The CamShift method.
- The Kalman filter for tracking objects.
- The Gaussian Mixture-based Background/Foreground segmentation algorithm.

Then, best methods which fit our needs were chosen and implemented within *Medusa*. Moreover, the possibility of mixing them could be considered and executed if relevant results are obtained.

In order to understand how the software works, a previous familiarization was done. *Medusa* is implemented in C++, using *Qt* libraries for making the GUI and *OpenCV* for the video processing. It has different classes responsible of controlling each feature like the cameras, the contours detection, the display, etc. and all are linked by a graphical user interface which is the software's main window, where the cameras are displayed and all the possible options and settings are available. *Medusa* is based on a multithread structure which allows doing all the stipulated functions.

The motion detection algorithm, as said above, must be adapted and fitted within *Medusa*. Therefore, some modifications and new classes were necessary.

After this step, the possibility of automatically stopping the beam could be considered, studied

and implemented if it is finally possible.

Once all the tasks above were made, the integration of *Medusa* within the *iMagX (Université Catholique de Louvain & IBA)* software platform was planned (but not implemented within the thesis schedule due to a lack of time). In order to achieve this it is needed to understand how the platform works.

These are the main objectives of this thesis. After that, if it possible and within the accorded time, other extra objectives are presented. These are:

- Video loading option
- Video recording option
- Pixels to millimetres calibration method

Therefore, the thesis fulfilment can be defined in three blocks, which are the following:

1. Study of the different motion detection techniques and decision of the best ones according to our needs.
2. Implementation of the chosen methods and integration to *Medusa*.
3. Development of new features and improvements.

Chronological and typological scopes can be placed into the present biomedical engineering sector as a way to help the improvement of medical industry, specifically cancer treatment and more specifically the proton therapy.

SOFTWARE OPERATION WITHIN THE APPLICATION CONTEXT

Cameras and patient disposition

At *Centre de Protonthérapie d'Orsay* we can find three different treatment rooms: the eye treatment room, the head and neck treatment room and the body treatment room. Depending on tumour's location, the patient is treated in one or another.

In the eye treatment the patient is sitting and the beam is applied through the eye. Two cameras are used in this case. One focused in the beam trajectory and another used for controlling the patient movements. Last one is mobile and is situated at the best possible position.

For the head and neck treatment patient stay lying on a special litter. There are also two cameras, but both are for controlling the movement. One of them focusing left side of the head and the other one the other side. Both are fixed this time.

One single fixed camera is used for the body treatment and is the patient who moves to the correct position.

How *Medusa* works with the cameras

Medusa uses the signal of the cameras to display them in its graphical interface. When the patient is ready and in correct position the software defines the contours automatically. Extra non-detected (but also important) contours can be manually defined. This is done with a digital pen for better precision. *Medusa* ensures permanently that the signal of the cameras is arriving properly and the connection is not lost.

How the therapists stop the beam

Once all is ready, the specialists, from a room outside the treatment one, initialise the beam with a controller and the irradiation starts. It is a 50-60 seconds process where the patient must be motionless. The controller has two buttons: a green one which is used for start/resume the irradiation and a red one which is used for stopping it. Hence, the therapists are continuously watching the video from the cameras with the defined contours and when the patient moves relevantly out from the contours they stop the beam pushing the red button. Once patient is in correct position anew, they resume the irradiation pushing the green button. When the full treatment time is reached the beam stops automatically.

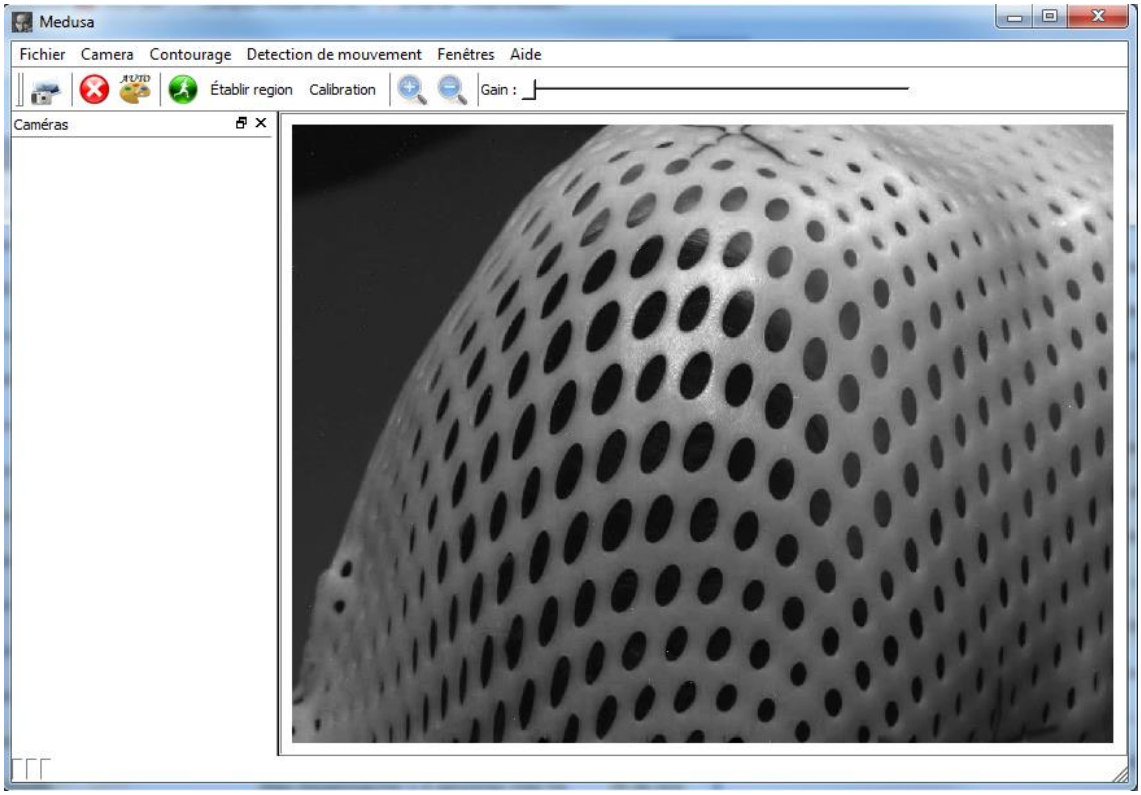


Figure 1. Medusa's main window

REPORT ORGANIZATION

- Chapter 1: Introduction, motivation and scopes.
- Chapter 2: Antecedents. Study and explanation of the different motion analysis and detection methods.
- Chapter 3: Comparison between methods, selection of the wanted ones and definition of the reasons.
- Chapter 4: Introduction to Medusa and integration of the algorithms.
- Chapter 5: Extra improvements.
- Chapter 6: Extra interpretation of tests and results.
- Chapter 7: Conclusions, personal opinion and future working.

CHAPTER 2

ANTECEDENTS

Motion analysis is one of the big fields in signal processing, specifically in image and video processing. Motion detection techniques have been developed through the years and used in coding standards, security applications, monitoring, robots' artificial intelligence, military tasks, biomedical applications, virtual reality interfaces, gestures interpretation and so on.

This motion analysis consists in extracting knowledge from image sequences. For this, the images of a sequence are compared analysing possible similarities or differences between them. Thus, it is possible to treat two types of possibilities:

1. Motion detection. The objective is to detect any type of movement considered as relevant in the image sequence. In order to achieve this, different algorithms exist focused on different computing methods. Depending on the application, sometimes is better to use a specific method and sometimes is better to use other ones or even a mix of some of them. Then, motion information will be used for a wanted purpose. There are dense algorithms which analyse every whole frame (each pixel of the image) and sparse ones which analyse particular regions of the image or even several pixels.

2. Object tracking. While motion detection just detects if there is movement or not, object tracking has the task of identify and follow an object (or some of them) through the sequence, knowing his shape in the image plane. Its path can be estimated for obtain the future position. In order to do this, segmentation, motion estimation and tracing techniques are required. It is possible to determine the number of objects that are in the image, their speed and other particular features.

There are lots of papers that reflect all this research with lots of motion-based methods, tests, example applications, etc.

During this thesis, we used an open source image processing library called *OpenCV*. Therefore, we limited our study to motion detection techniques available in *OpenCV*. Papers and other sources which define them are analysed and explained in next chapters. These methods, as said in the introduction, are the next ones:

- The Lucas-Kanade optical flow algorithm.
- The Dual TV-L1 optical flow algorithm.
- The SimpleFlow optical flow algorithm.
- The Gunnar Farnebäck's optical flow algorithm.
- The Motion History Images method.

- The CamShift method.
- The Kalman filter for tracking objects.
- The Gaussian Mixture-based Background/Foreground segmentation algorithm.

EXPLANATION OF THE MOTION DETECTION METHODS

In this section an exhaustive explanation of the motion detection methods is done. All the following algorithms are computed by *OpenCV* and are used for testing the application.

LUCAS-KANADE ALGORITHM

This method starts with an initial estimate of h, which is a disparity vector between frames, and uses the spatial intensity gradient at each point of the image to modify the current estimate of h to obtain an h which yields a better match. It can combine with a coarse-to-fine strategy. Next figure (based on one-dimensional case) describes better this procedure.

One dimensional case

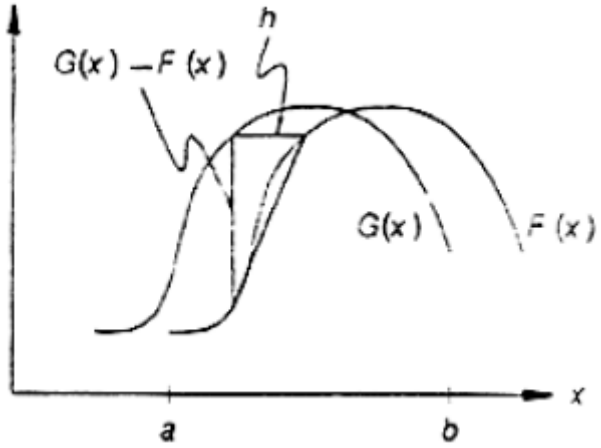


Figure 2.1. Two curves to be matched. (Source: reference [1]).

We want to find the horizontal disparity *h* between two curves F(x) and G(x) = F(x + h). The solution to this depends on a lineal approximation of the behaviour of F(x) in the neighbourhood of x. For small *h*:

$$F'(x) \approx \frac{F(x+h) - F(x)}{h} = \frac{G(x) - F(x)}{h} \qquad (1)$$

So that:

$$h \approx \frac{G(x) - F(x)}{F'(x)} \qquad (2)$$

In order to this approximation be adequate *h* requires to be small enough.

A natural method for combining the various estimates of *h* at various values of x would be to average them:

$$h \approx \sum_x \frac{G(x) - F(x)}{F'(x)} / \sum_x 1 \tag{3}$$

We can improve this average if we notice that the approximation in (1) is good where F(x) is almost lineal and is worse where |F''(x)| is large. Hence, we could weigh the contribution of each term to the average in (3) in inverse proportion to an estimate of |F''(x)|:

$$|F''(x)| \approx \frac{G'(x) - F'(x)}{h} \tag{4}$$

Since the estimate is to be used as a weight in the average, we can reject the factor 1/*h* in (4) and use

$$w(x) = \frac{1}{|G'(x) - F'(x)|} \tag{5}$$

This is done due to cases where the estimate of *h* computed by (2) is zero, which is not good. The weight corresponding to that estimation is small since the difference between F'(x) and G'(x) is large in this point. The average weighting is

$$h \approx \sum_x \frac{w(x)[G(x) - F(x)]}{F'(x)} / \sum_x w(x) \tag{6}$$

where w(x) is given by (5).

Once this estimate is obtained, we can move F(x) by our estimate of *h*, and repeat this procedure, yielding the following iteration:

$$h_0 = 0$$

$$h_{k+1} = h_k + \sum_x \frac{w(x)[G(x) - F(x + h_k)]}{F'(x + h_k)} / \sum_x w(x) \tag{7}$$

The sequence of estimates of *h* will converge to the best *h*.

Alternative derivation

The derivation given above does not generalize well in two-dimensions because the two-dimensional linear approximation occurs in a different form. Also, is undefined where F(x) = 0, i. e. where the curve is level. Both problems can be corrected by using the linear approximation of (1)

$$F(x + h) \approx F(x) + hF'(x) \tag{8}$$

to find the *h* which minimizes the L_2 norm of the difference between the curves:

$$E = \sum_x [F(x + h) - G(x)]^2$$

To minimize the error with respect to *h*, we set

$$0 = \frac{\partial E}{\partial h} \approx \frac{\partial}{\partial h} \sum_x [F(x) + hF'(x) - G(x)]^2$$

$$= \sum_x 2F'(x)[F(x) + hF'(x) - G(x)]$$

and

$$h \approx \sum_x \frac{F'(x)[G(x) - F(x)]}{\sum_x F'(x)^2} \tag{9}$$

This is essentially the same solution as in (6), but with w(x) = F'(x)². The form of the linear approximation that we have used here generalizes to two or more dimensions.

The iterative form now is

$$h_0 = 0$$

$$h_{k+1} = h_k + \sum_x \frac{w(x)F'(x + h_k)[G(x) - F(x + h_k)]}{\sum_x w(x)F'(x + h_k)^2} \tag{10}$$

where w(x) is given by (5).

Performance

We can improve the range of convergence by suppressing high spatial frequencies in the image which can be accomplished smoothing the image, i.e. replacing each pixel by weighted average neighbouring pixels. Trade off: smoothing suppresses small details and thus makes the match less accurate.

This suggests a coarse-to-fine strategy, i.e. to use a low resolution smoothed version to obtain an approximation match and apply the algorithm to higher resolution images for refine the match. The weighted function serves to improve the accuracy of the approximation and to speed up the convergence.

Implementation

We cannot calculate F'(x) exactly, but for the purposes of this algorithm and for implementing (10) we can estimate it by

$$F'(x) \approx \frac{F(x + \Delta x) - F(x)}{\Delta x}$$

where Δx is approximately small.

Multiple dimensions

The algorithm above can be generalized to two or more dimensions. The L_2 norm minimization is the following:

$$E = \sum_{x \in R} [F(x+h) - G(x)]^2 \tag{11}$$

Where *x* and *h* are n-dimensional row vectors.

The linear approximation analogous to that in (8) is

$$F(x+h) \approx F(x) + h\frac{\partial}{\partial x}F(x)$$

Where $\frac{\partial}{\partial x}$ is the gradient operator with respect to x, as a column vector:

$$\frac{\partial}{\partial x} = \left[\frac{\partial}{\partial x_1}\ \frac{\partial}{\partial x_2} \cdots \frac{\partial}{\partial x_n}\right]^T$$

To minimize E we set

$$0 = \frac{\partial E}{\partial h} \approx \frac{\partial}{\partial h}\sum_x \left[F(x) + h\frac{\partial F}{\partial x} - G(x)\right]^2$$

$$= \sum_x 2\frac{\partial F}{\partial x}\left[F(x) + h\frac{\partial F}{\partial x} - G(x)\right]$$

where *h* is

$$h = \left[\sum_x \left(\frac{\partial F}{\partial x}\right)^T [G(x) - F(x)]\right]\left[\sum_x \left(\frac{\partial F}{\partial x}\right)^T \left(\frac{\partial F}{\partial x}\right)\right]^1$$

which is similar to the one-dimensional version in (9).

Further generalizations

The technique can be extended to an arbitrary linear transformation, such as rotation, scaling and shearing (not only simple translation):

$$G(x) = F(xA + h)$$

A is a matrix expressing the linear spatial transformation between F(x) and G(x).

The energy to minimize in this case is

$$E = \sum_x [F(xA + h) - G(x)]^2$$

The linear approximation is the following:

$$F(x(A+\Delta A)+(h+\Delta h)) \approx F(xA+h)+(x\Delta A+\Delta h)\frac{\partial}{\partial x}F(x) \quad (12)$$

This generalization is useful in applications such as stereo vision, where two different views of the object will be available, due to the position of the cameras or the processing of the two images. This difference can be modelled as a linear transformation:

$$F(x) = \propto G(x) + \beta$$

Where $\propto$ is a contrast adjustment and β a brightness adjustment.

Combining this with the general problem we obtain

$$E = \sum_x [F(xA+h) - (\propto G(x) + \beta)]^2$$

as the quantity to minimize.

DUAL TV-L¹ ALGORITHM

It begins from the *Horn and Schunk* formulation for solving the undetermined system

$$\Delta I \cdot \boldsymbol{u} + \frac{\partial}{\partial t} I = 0 \quad (1)$$

which is the expression of the derivative of *I(x,y,t)* equal to zero after applying the chain rule. It comes from assuming that the intensity values of *I(x(t), y(t) ,t)* are constant during the motion. Then, **u** is a vector field defined by the velocities for the trajectories of every point at one instant. $\boldsymbol{u}(x,y) = (u_1(x,y), u_2(x,y))$, i.e. the displacement field. This leads to a system twice as many variables as equations and results to be undetermined.

Therefore, the proposal of *Horn and Schunck* is to select the **u** that minimizes the following equation:

$$E_{Horn\_Schunck}(\boldsymbol{u}) = \min\left\{\int_\Omega \left[\lambda\left(\Delta I \cdot \boldsymbol{u} + \frac{\partial}{\partial t} I\right)^2 + (|\nabla u_1|^2 + |\nabla u_2|^2)\, d\Omega\right]\right\} \quad (2)$$

$|\nabla u_1|^2 + |\nabla u_2|^2$ is called the regularization term and it penalizes high gradients of **u** in order to obtain smooth displacement fields and it effectively disallows discontinuities. Equation (1) is suitable if the image data is continuous in time. For fitting general image sequences this expression is replaced by the non-linear formulation $I_1(x+\boldsymbol{u}) - I_0(x) = 0$. Hence, we have the following expression:

$$\min\left\{\left[\int_\Omega \lambda\big(I_1(x+\boldsymbol{u}) - I_0(x)\big)^2 + (|\nabla u_1|^2 + |\nabla u_2|^2)\right] d\Omega\right\} \quad (3)$$

With two frames given I_1 and I_0, the objective is to find the disparity map **u** which minimizes an image-based criterion together with a regularization force. We focus on the plain intensity difference between pixels as the image similarity score.

Hence, **u** is the minimizer of:

$$\int_{\Omega} \{\lambda \emptyset\big(I_1(x+\boldsymbol{u}) - I_0(x)\big) + \varphi(u, \nabla u, \dots)\}\, d\Omega \tag{4}$$

Note that for $\emptyset(x) = x^2$ and $\varphi(\nabla u) = |\nabla u|^2$ results on the *Horn and Schunck* model.

The choice of $\emptyset(x) = x$ and $\varphi(\nabla u) = |\nabla u| = |\nabla u_1| + |\nabla u_2|$ leads to the L_1 data penalty term and total variation regularization:

$$E(\boldsymbol{u}) = \int_{\Omega} \{\lambda |I_1(x+\boldsymbol{u}) - I_0(x)| + |\nabla u_1| + |\nabla u_2|\}\, d\Omega \tag{5}$$

Then non-linear term $I_1(x+\boldsymbol{u})$ can be linearized using Taylor expansions:

$$\rho(\boldsymbol{u}) = \nabla I_1(x+\boldsymbol{u}^0)\cdot(\boldsymbol{u}-\boldsymbol{u}^0) + I_1(x+\boldsymbol{u}^0) - I_0(x) = 0 \tag{6}$$

with $\mathbf{u^o}$ a close approximation to **u.** Thus, we have

$$E(\boldsymbol{u}) = \int_{\Omega} \{\lambda |\rho(\boldsymbol{u})| + |\nabla u_1| + |\nabla u_2|\}\, d\Omega \tag{7}$$

An efficient way to minimize the equation above is to introduce the following convex approximation of E:

$$E_\theta(\boldsymbol{u}, \boldsymbol{v}) = \int_{\Omega} \left\{ |\nabla u_1| + |\nabla u_2| + \frac{1}{2\theta}|\boldsymbol{u}-\boldsymbol{v}|^2 + \lambda|\rho(\boldsymbol{u})| \right\} d\Omega \tag{8}$$

Setting θ to a very small value forces **u** and **v** to be nearly equal, reducing the expression to (7). E_θ can be minimized by alternatively fixing **u** or **v** and solving for the other variable.

1. For fixed **v**, solve

$$\min_u \int_{\Omega} |\nabla u_1| + |\nabla u_2| + \frac{1}{2\theta}|\boldsymbol{u}-\boldsymbol{v}|^2 \tag{9}$$

2. For fixed **u**, solve

$$\min_v \int_{\Omega} \frac{1}{2\theta}|\boldsymbol{u}-\boldsymbol{v}|^2 + \lambda|\rho(\boldsymbol{u})| \tag{10}$$

The first sub-problem fits the total variation denoising model of *Rudin-Osher-Fatemi*, which can be solved by *Chambolle's* duality-based algorithm. The second sub-problem can be solved point-wise by thresholding, since it does not depend on spatial derivatives of **v**.

- An efficient solution for 1 is obtained by computing the fixed point of the following iteration over the dual vector fields p_1 and p_2:

$$p_d^{k+1} \coloneqq \frac{p_d^k + \tau/\theta \nabla(v_d^{k+1} + \theta div(p_d^k))}{1 + \tau/\theta \left|\nabla\left(v_d^{k+1} + \theta div(p_d^k)\right)\right|}, d \in \{1,2\} \tag{11}$$

and recovering **u** as

$$u_d^{k+1} \coloneqq v_d^{k+1} + \theta div\left(p_d^k\right), d \in \{1,2\} \tag{12}$$

- The second minimization problem can be solved as follows:

$$v^{k+1} \coloneqq u^{k+1} + TH(u^{k+1}, u^0) \tag{13}$$

with TH the thresholding operation

$$TH(u, u^0) \coloneqq \begin{cases} \lambda\theta\nabla I_1(x + \boldsymbol{u}^0) & if \rho(\boldsymbol{u}, \boldsymbol{u}^0) < -\lambda\theta|\nabla I_1(x + \boldsymbol{u}^0)|^2 \\ -\lambda\theta\nabla I_1(x + \boldsymbol{u}^0) & if \rho(\boldsymbol{u}, \boldsymbol{u}^0) > \lambda\theta|\nabla I_1(x + \boldsymbol{u}^0)|^2 \\ -\rho(\boldsymbol{u}, \boldsymbol{u}^0)\frac{\nabla I_1(x+\boldsymbol{u}^0)}{|\nabla I_1(x+\boldsymbol{u}^0)|^2} & if\ |\rho(\boldsymbol{u}, \boldsymbol{u}^0)| \leq \lambda\theta|\nabla I_1(x + \boldsymbol{u}^0)|^2 \end{cases} \tag{14}$$

When the objects move beyond the image limits it is not possible to compute ρ and is convenient to disable this thresholding and to use only the regularization term.

Therefore, the input of the algorithm is a pair of images $I_0(x)$ and $I_1(x)$, with $x = (i, j)$ the pixel index. The output is a vector field $\boldsymbol{u}(x) = (u_1(x), u_2(x))$. The residual $\rho(\boldsymbol{u})$ is a scalar field (a grey-valued image) and its computation involves a warping of I_1 and ∇I_1 by the deformation $\boldsymbol{u}^0$, which has to be close to **u**, so that the approximation error of the Taylor expansions above is small**.** This deformation is computed by a multiscale scheme.

Implementation

The algorithm can be understood as two modules: one for computing the optical flow at a given scale using the numerical scheme defined previously and another to implement the pyramidal scheme, since the energy minimization procedure is embedded into a coarse-fine approach to avoid convergence to unfavourable local minima. The execution follows the next steps:

- Beginning with the coarsest level, we solve $E(\boldsymbol{u}) = \int_\Omega \{\lambda|I_1(x + \boldsymbol{u}) - I_0(x)| + |\nabla u_1| + |\nabla u_2|\}\ dx$ at each level of the pyramid and propagate the solution to the next finer level.

- At the coarsest level **u** starts with zero.

- At the beginning of a new level **v** is initialized with **u** and all p_d's are set to zero. Hence, the procedure updates the **u** vector and it uses three temporal vector fields **v**, p_1 and p_2 for computing the intermediate steps.

- The minimization procedure alternates one step for the fixed-point scheme to update all p_d (and therefore **u**, (11) and (12)), and other step the thresholding from (13) to improve **v**.

- For stopping the algorithm before the default number of iterations ($N_{maxiter}$) we use stopping criterion based on the L_2 norm between consecutive values of **u** (u^k, u^{k+1}).

$$\frac{1}{N_x N_y}\sum_{i,j}\left(u_1^{k+1}(i,j)-u_1^k(i,j)\right)^2+\left(u_2^{k+1}(i,j)-u_2^k(i,j)\right)^2 < \varepsilon^2 \quad (15)$$

- To downscale an image, it is first convolved with a Gaussian kernel and then sampled using bicubic interpolation. We use a downscaling factor, $\eta \in (0,1)$, that allows for smoother transitions between the scales.

For better understanding [2] gives a working scheme of the algorithm. That scheme is the following:

Algorithm 1: Pyramidal structure management

Input: $I_0, I_1, \tau, \lambda, \theta, \varepsilon, \eta, N_{maxiter}, N_{warps}, N_{scales}$
Output: $\mathbf{u}$

Normalize images between 0 and 255
Convolve the images with a Gaussian of $\sigma = 0.8$
Create the pyramid of images I^s using η (with $s = 0, \ldots, N_{scales}-1$)
$\mathbf{u}^{N_{scales}-1} \leftarrow (0,0)$
for $s \leftarrow N_{scales}-1$ **to** 0 **do**
    TV-L^1_optical_flow$(I_0, I_1, u^0, \tau, \lambda, \theta, \varepsilon, N_{maxiter}, N_{warps})$
    if $s > 0$ **then**
        $\mathbf{u}^{s-1}(\mathbf{x}) := 2\mathbf{u}^s(\mathbf{x}/\eta)$
    end
end

Procedure TV-L^1_optical_flow$(I_0, I_1, u^0, \tau, \lambda, \theta, \varepsilon, N_{maxiter}, N_{warps})$

$\mathbf{p}_1 \leftarrow (0,0)$
$\mathbf{p}_2 \leftarrow (0,0)$
for $w \leftarrow 1$ **to** N_{warps} **do**
    Compute $I_1(\mathbf{x}+\mathbf{u}^0(\mathbf{x})), \nabla I_1(\mathbf{x}+\mathbf{u}^0(\mathbf{x}))$ using bicubic interpolation
    $n \leftarrow 0$
    while $n < N_{maxiter}$ **and** *stopping_criterion* $> \varepsilon$ **do**
        $\mathbf{v} \leftarrow TH(\mathbf{u}, \mathbf{u}^0)$
        $\mathbf{u} \leftarrow \mathbf{v} + \theta \mathrm{div}(\mathbf{p})$
        $\mathbf{p} \leftarrow \frac{\mathbf{p}+\tau/\theta\nabla\mathbf{u}}{1+\tau/\theta|\nabla\mathbf{u}|}$
        $n \leftarrow n+1$
    end
end

Figure 2.2. Algorithm procedure. (Source: reference [2])

SIMPLE FLOW ALGORITHM

The Simple Flow is an iteration-free optical flow computation method. It calculates only a sparse set of samples in regions with a uniform motion. The algorithm runs a multiscale pyramid approach and one of the main features is that where the motion is smooth we can afford to interpolate some of the flow vectors from their neighbours instead of analysing the frame data. It manipulates probabilities derived from pixel-wise color differences instead of flow vectors.

Simple likelihood model

We seek flow vectors (u, v) such that $F_t(x, y)$ and $F_{t+1}(x + u, y + v)$, which are the RGB colors of the (x, y) pixel in F_t and the $(x + u, y + v)$ pixel in F_{t+1}, are similar. We model this by an energy term:

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

That corresponds to the likelihood probability $p \propto \exp(-e)$. p also represents the reliability of the info that it embeds. If p is nearly uniform provides only weak evidence about the flow, whereas if p has peaked distributions indicates reliable data.

Local validity as smoothness prior

Flow vector (u_0, v_0) at pixel (x_0, y_0) is a good explanation for the motion at (x_0, y_0) and other pixels in a neighbourhood:

$$(u_0, v_0) = \underset{(u,v)\in\Omega}{\operatorname{argmax}} \prod\nolimits_{(x,y)\in N_0} p(x, y, u, v)$$

The Ω expression is the set of possible vectors (u, v) that we consider. Also, we can understand this behaviour in the energy cost domain:

$$(u_0, v_0) = \underset{(u,v)\in\Omega}{\operatorname{argmin}} \sum\nolimits_{(x,y)\in N_0} e(x, y, u, v)$$

Weights are added to account for how pixels relate each other:

$$E(x_0, y_0, u, v) = \sum\nolimits_{(x,y)\in N_0} w_d w_c e(x, y, u, v)$$

$$w_d = \exp\left(-\frac{\left|\left|(x_0, y_0) - (x, y)\right|\right|^2}{2\sigma_d}\right)$$

$$w_c = \exp\left(-\frac{\left|\left| F_t(x_0, y_0) - F_t(x, y)\right|\right|^2}{2\sigma_c}\right)$$

w_d is for weighting the distance between pixels and w_c is for the color difference. These weights correspond, furthermore, at the bilateral filter weights used for accounting for edges.

Detecting occlusions

In order to detect occlusions, we compare the forward flow (u_f, v_f) from t to $t+1$, and the backward flow (u_b, v_b) from $t+1$ to t.

$$\left|\left|(u_f, v_f) - (-u_b, -v_b)\right|\right|$$

When computed difference is high, we mark the pixel as occluded between the frames.

Practical implementation

The next steps describe the operation of the algorithm.

- It computes the color difference between $F_t(x_0, y_0)$ and every pixel in an n x n window in F_{t+1}, i.e. the *e* difference, centred in (x_0, y_0).
- This produces an n-dimensional *e* vector at each pixel. We will have n² measured distances for every (x_0, y_0) pixel.
- Then, we compute E by applying a bilateral filter on these *e* vectors using the frame data F_t to define the color weights w_c.
- Finally, we estimate the flow as the (u_0, v_0) vector that minimizes E. This produces values that are integers. To get a sub-pixel estimate we fit a parabola to the 3x3 pixels centred at (u_0, v_0) and extract his minimum.
- The result can be regularized by applying a bilateral filter on the flow vectors. For this, we discard the occluded pixels and use w_d and w_c with an additional weight w_r that represents how reliable is our flow estimate at (x, y):

$$w_r(x, y) = \underset{(u,v)\in\Omega}{\text{mean}}\, e(x, y, u, v) - \underset{(u,v)\in\Omega}{\min}\, e(x, y, u, v)$$

Multiscale approach

With the model described above we have to consider large n x n windows to be able to recover large motion. In order to avoid that, a multiscale approach can be implemented. Two different schemes can be done:

Multiscale flow estimation

This is the standard method, which follows the next steps:

- To construct an image pyramid for each frame. Each level is twice coarser than the previous one (the l-th level has a resolution 2^l times lower than the original frame).
- At the coarsest level of the pyramid we estimate the flow using the scheme described before. We compute the flow at level l assuming that the flow al $l+1$ is known:

- An initial estimate $(\bar{u}_l, \bar{v}_l)$ of the flow is computed by upsampling the previous level (u_{l+1}, v_{l+1}) using joint-bilateral upsampling, followed by a x2 multiplication to account for the change in resolution.

- Then, we follow the standard flow computation process described before with the only change that now we aggregate probabilities in a neighbourhood $\bar{N}$ centred on $(x_0 + \bar{u}, y_0 + \bar{v})$.

- For the flow vector at (x_0, y_0), we select (u_0, v_0) that minimizes $E(x_0, y_0, u, v)$.

Multiscale efficient scheme

This particular model tries to isolate image regions where the flow changes slowly. For pixels in these regions we change the energy minimization operation by a simple interpolation, which has negligible cost.

- For each layer a flow irregularity map is estimated. Thus, we locate the regions where the variation is smooth and regions where it varies more.

- At each pixel (x_0, y_0), we compute the irregularity value as:

$$\max_{(x,y)\in N_0} \left|\left|\big(u(x_0, y_0), v(x, y)\big) - \big(u(x_0, y_0), v(x_0, y_0)\big)\right|\right|$$

  During the upsampling, if the value is above a threshold τ, we run the full pipeline on the corresponding upsampled pixels, whereas if it is below τ, we compute the flow at the corners of the patch and find the flows at other pixels using bilinear interpolation.

- We find smooth flow regions as we upscale, which effectively yields constant time computation to upsample these areas.

This efficient scheme allows the Simple Flow method to perform sub-linear computation and permits to process HD videos and movie sequences in original format.

FARNEBÄCK ALGORITHM

The first step of this algorithm is to approximate by quadratic polynomials each neighbourhood of two frames. A method for estimate the displacement fields from the polynomial expansion coefficients is derived in order to observe how a polynomial transforms under translation and after some treatments leads to a robust algorithm.

Polynomial expansion

The idea consists in approximating some neighbourhood of each pixel with a polynomial. We are interested in quadratic polynomials, giving us the local signal model, expressed in a local coordinate system.

$$f(x) \sim x^T A x + b^T x + c \tag{1}$$

Coefficients are estimated from weighted least squares (given a set of pairs, we select the continuous function that best represents the data. This is done according to the minimum quadratic error criteria) fitted to the signal values in the neighbourhood.

The weighting has two components called certainty and applicability. Certainty is coupled to the signal values in the neighbourhood. It is a good idea to set it to zero outside the image. Thus, neighbourhood points outside the image have no impact on the coefficient estimation. Applicability is the relative weight of points in the neighbourhood based on their position. Typically give most weight to the centre point and let the weights decrease radially. The width of the applicability determines the scale of the structures which will be captured by the expansion coefficients.

Displacement estimation

Since each neighbourhood is approximated by a polynomial, we analyse what happens if a polynomial undergoes an ideal translation. Consider the exact quadratic polynomial:

$$f_1(x) \sim x^T A_1 x + {b_1}^T x + c_1 \tag{2}$$

We construct a new signal $f_2(x)$ by a global displacement by d:

$$f_2(x) = f_1(x - d) = (x - d)^T A_1 (x - d) + {b_1}^T (x - d) + c_1$$

$$= x^T A_1 x + (b_1 - 2A_1 d)^T x + d^T A_1 d - {b_1}^T d + c_1$$

$$= x^T A_2 x + {b_2}^T + c_2 \tag{3}$$

Equating the coefficients in the quadratic polynomials yields

$$A_2 = A_1 \tag{4}$$

$$b_2 = (b_1 - 2A_1 d) \tag{5}$$

$$c_2 = d^T A_1 d - {b_1}^T d + c_1 \tag{6}$$

By (5) we obtain the solution for the translation d, at least if A_1 is non-singular, i.e. it has a matrix inverse. Therefore, we solve d as follows:

$$d = -\frac{1}{2}A_1^{-1}(b_2 - b_1) \tag{7}$$

Practical considerations

Assumptions about an entire signal being a simple polynomial and global translation relating the two signals are unrealistic. The basic relation (7) can be used for real signals, although errors are introduced when assumptions are relaxed. We have to try if these errors can be small enough to give a useful algorithm.

We replace the global polynomial in (2) with local polynomial approximations. Thus, we start by doing a polynomial expansion of both images, giving us the expansion coefficients $A_1(x)$, $b_1(x)$ and $c_1(x)$ for the first image, and $A_2(x)$, $b_2(x)$ and $c_2(x)$ for the second image.

We have seen that $A_1 = A_2$, but in practice we have to establish the next approximation:

$$A(x) = \frac{A_1(x) + A_2(x)}{2} \tag{8}$$

Also,

$$\Delta b(x) = -\frac{1}{2}(b_2(x) - b_1(x)) \tag{9}$$

in order to obtain the primary constraint

$$A(x)d(x) = \Delta b(x) \tag{10}$$

$d(x)$ indicates that we have also replaced the global displacement in (3) with a spatial varying displacement field.

Estimation over a neighbourhood

We assume that the displacement field is slowly varying, so we can integrate info over a neighbourhood of each pixel.

We try to find $d(x)$ satisfying (10) as well as possible over a neighbourhood I of x, or more formal, minimizing

$$\sum_{\Delta x \in I} \omega(\Delta x) \left|\left|A(x + \Delta x)d(x) - \Delta b(x + \Delta x)\right|\right|^2 \tag{11}$$

The minimum is obtained for

$$d(x) = \left(\sum \omega\, A^T A\right)^{-1} \sum \omega\, A^T \Delta b \tag{12}$$

The minimum value is given by

$$e(x) = \left(\sum \omega\, \Delta b^T \Delta b\right) - d(x)^T \sum \omega\, A^T \Delta b \tag{13}$$

The value $e(x)$ can be used as a reversed confidence value. Small numbers indicate high confidence and big numbers low confidence.

Parameterized displacement fields

In order to improve robustness the displacement field can be parameterized according to some motion model. We derive this for the eight parameter model in 2D:

$$d_x(x,y) = a_1 + a_2 x + a_3 y + a_7 x^2 + a_8 xy,$$
$$d_y(x,y) = a_4 + a_5 x + a_6 y + a_7 xy + a_8 y^2$$

It is possible to rewrite this as

$$d = S_p$$
$$S = \begin{pmatrix} 1 & x & y & 0 & 0 & 0 & x^2 & xy \\ 0 & 0 & 0 & 1 & x & y & xy & y^2 \end{pmatrix}$$
$$p = (a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6 \quad a_7 \quad a_8)^T$$

Inserting into (11) we obtain the weighted least squares problem:

$$\sum_i \omega_i \left|\left|A_i S_i p_i - \Delta b_i\right|\right|^2 \tag{14}$$

Index i indicates the coordinates in a neighbourhood. The solution is

$$p = \left(\sum_i \omega_i\, S_i^T A_i^T A_i S_i\right)^{-1} \sum_i \omega_i\, S_i^T A_i^T \Delta b_i \tag{15}$$

Incorporating a priori knowledge

We assume that local polynomials at the same coordinates are identical excepting a displacement. This is not completely true and they will vary spatially introducing errors to the constraints.

For small displacements there is no problem, but for larger ones the problem appears and increases. If we have a priori knowledge of the displacement field we can compare the polynomial at "x" in the first signal and the one at $x + \tilde{d}(x)$ in the second signal, where $\tilde{d}(x)$ is the priori displacement field (rounded to integers). So we only need to estimate the relative displacement between the real value and the one estimated a priori.

In order to work with this incorporation, we replace:

$$A(x) = \frac{A_1(x) + A_2(\tilde{x})}{2} \tag{16}$$

where $x = x + \tilde{d}(x)$.

$$\Delta b(x) = -\frac{1}{2}(b_2(\tilde{x}) - b_1(x)) + A(x)\tilde{d}(x) \tag{17}$$

$b_2(\tilde{x}) - b_1(x)$ computes the remaining displacement and $A(x)\tilde{d}(x)$ adds back the rounded a priori displacement.

Iterative and multi-scale displacement estimation

Since a priori displacement field exists, we can close the loop and iterate in order to obtain a better estimation. The lower the relative displacement is, the better displacement estimation we obtain.

Iteration is done using the displacement field of a step as a priori displacement in the next step. The first displacement field is set to zero. If the displacements are too larges we cannot expect an improvement, so a multi-scale analysis can be done. The procedure starts in a coarse scale in order to obtain an approximate estimation of the displacement and continues to finer scales for improving the precision. The disadvantage is that a recalculation of the polynomial expansion coefficients is needed for each scale. We can reduce the computational cost through subsampling between scales.

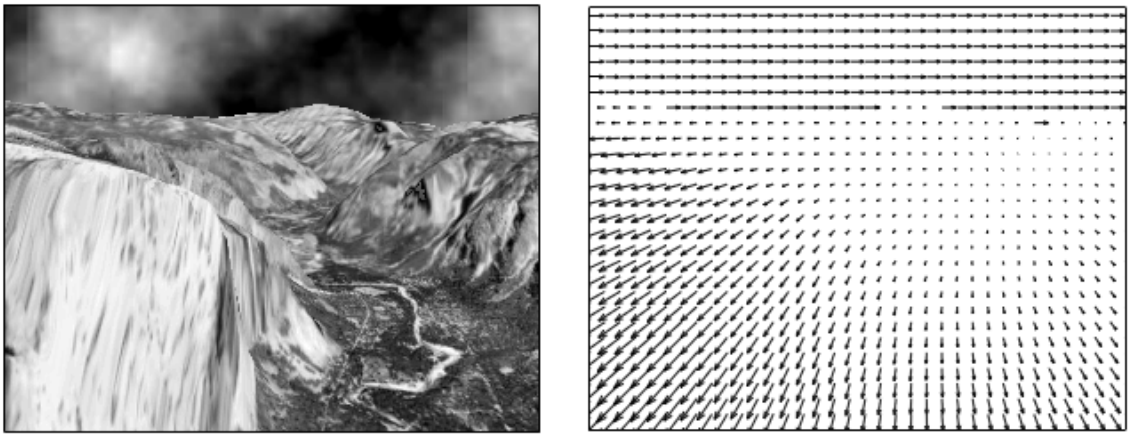


Figure 2.3. Optical flow field of the sequence at left (just a frame shown). (Source: reference [4])

MOTION HISTORY IMAGES METHOD

Successive layering of image silhouettes is used to represent patterns of motions. Every new frame, the existing silhouettes are decreased in value subject to some threshold and a new silhouette is overlaid at maximum brightness. This layered motion is called a Motion History Image (MHI).

The advantage of the MHIs is that a range of times from frame to frame to several seconds may be encoded in a single image. MHIs span the time scales of human gestures.

A new type of MHI is introduced in this case. A variation is done to directly encode actual time in a floating point format called time Motion History Image (tMHI).

The following scheme describes the whole process, step by step, which is explained later.

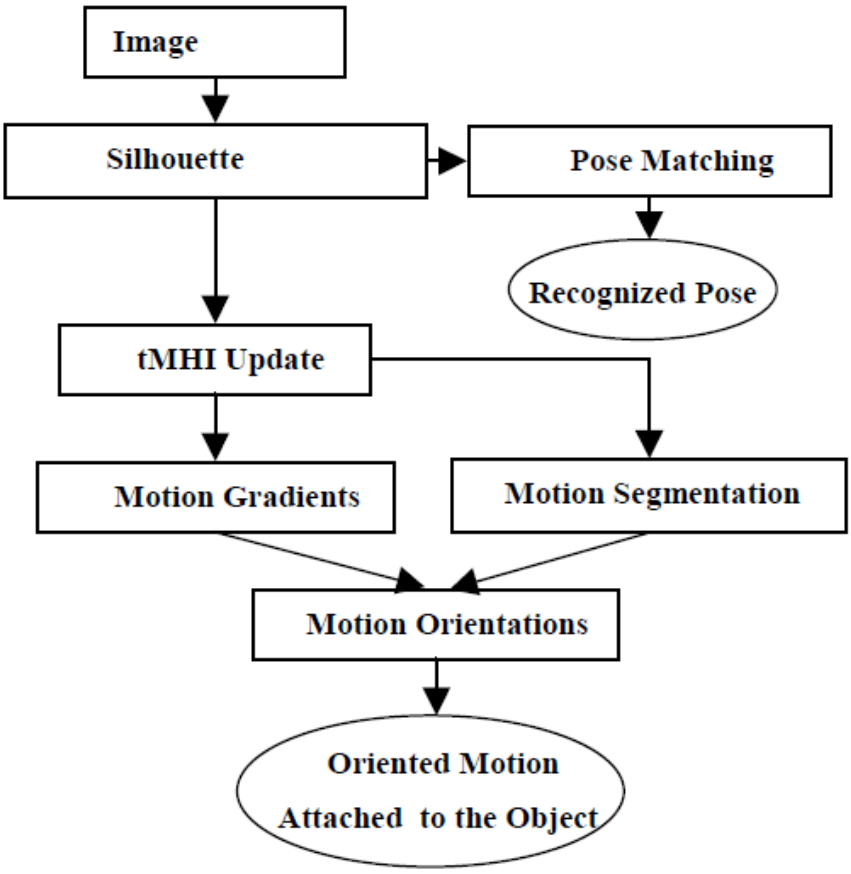


Figure 2.4. Process flow chart. (Source: reference [5])

The final result is a recognized pose and motion to that pose.

Pose and motion recognition

Silhouettes and pose recognition

The algorithm depends on generating silhouettes of the object of interest. The method selected to do this generation is a background subtraction method.

Silhouette generation

Pixels which are a set number of standard deviations from the mean RGB background are labelled as foreground. Then, dilation and region growing method are applied to remove noise and extract the silhouette. This process presents a limitation: no motion inside the body region can be seen. Possible solutions can be using multiple camera views or separately segmenting the relevant regions and overlaying them when they cross between themselves.

Mahalanobis match to Hu moments of silhouette pose

For recognition of the silhouette pose seven Hu moments provide shape descriptors which are invariant to translation and scale. These moments are of different orders and we need to use the Mahalanobis distance for matching based on a static measure of closeness of training examples.

$$mahal(x) = (x - m)^T K^{-1}(x - m) \quad (1)$$

where x is the moment feature vector, m is the mean of the training moment vectors and K^{-1} the inverse covariance matrix for the training vectors.

The discriminatory power of these moment features for the silhouette poses is indicated by a short example. Three different poses are done and distances to the trained model are computed. The one who presents less difference with a trained model is the pose that is identified.

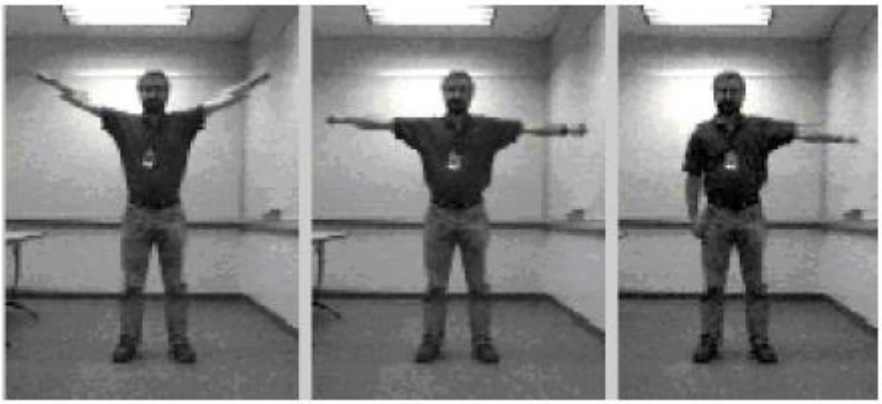


Figure 2.5. Test postures "Y", "T" and "|-". (Source: reference [5])

	Pose Y	Pose T	Pose |-
Test Y	**14**	204	2167
Test T	411	**11**	11085
Test |-	2807	257	**28**

Figure 2.6. Discrimination results of posture recognition. Distance to correct pose model is much smaller than distances to incorrect poses. (Source: reference [5])

Note: The discriminatory power is the average probability that a typing system will assign a different type to two unrelated strains randomly sampled in a population of a given taxonomy.

Timed Motion History Images (tMHI)

A floating point MHI is used where new silhouettes values are copied in with a floating point timestamp in format seconds.milliseconds. The MHI representation is updated as follows:

$$tMHI_{\delta}(x,y) = \begin{cases} \tau & if current\ silhouette\ at\ (x,y) \\ 0 & else\ if\ tMHI_{\delta}(x,y) < (\tau - \delta) \end{cases} \quad (2)$$

τ is the current timestamp and δ is the maximum time duration constant associated with the template.

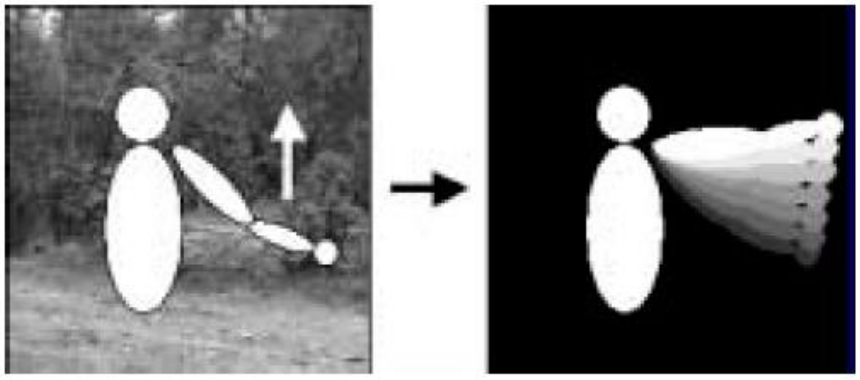


Figure 2.7. Successive silhouettes of an upward arm movement encoded on floating point timestamps yields the tMHI at right. (Source: reference [5])

Motion history gradients

If we take the gradient of the tMHI then we get the direction vectors pointing in the direction of the movement. These vectors will point orthogonal to the moving object boundaries at each step in the tMHI, giving normal optical flow representation.

Gradients of the tMHI are obtained through Sobel filters in X and Y directions yielding the spatial derivatives $F_x(x, y)$ and $F_y(x, y)$. Then, the gradient orientation at each pixel is:

$$\emptyset(x, y) = arctan \frac{F_y(x, y)}{F_x(x, y)} \tag{3}$$

The gradient is only valid within the tMHI. Surrounding boundary of the tMHI should not be used because non-silhouette (zero valued) pixels would be included in the gradient computation corrupting the results. So, only tMHI interior silhouette pixels should be examined.

Also, we must not use gradients of MHI pixels where contrast is too low (inside a silhouette) or too high (large temporal disparity) in their local neighbourhood.

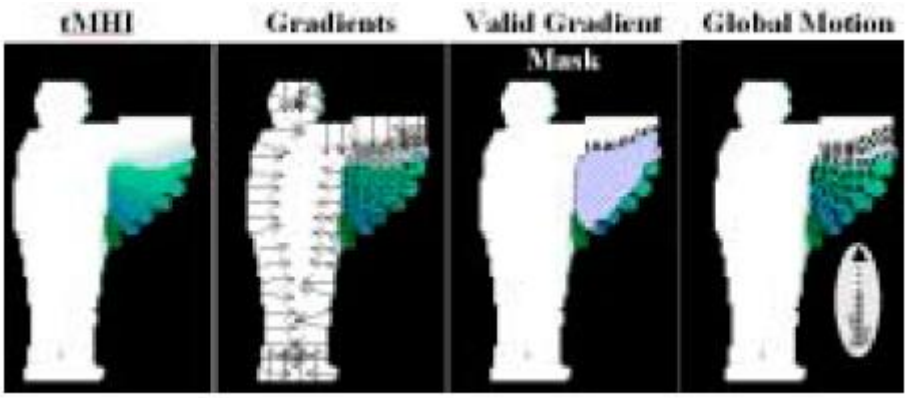


Figure 2.8. tMHI. Gradients. Mask and Global orientation (orientation histogram with global detection vector calculated). (Source: reference [5])

After the motion gradients computation it is necessary to extract motion features and find the global motion orientation.

Global gradient orientation

Calculation of the global gradient orientation should be weighted by normalized tMHI values to give more influence to the most current motion within the template.

Calculation of the global weighted orientation is as follows:

$$\overline{\emptyset} = \emptyset_{ref} + \frac{\sum_{x,y} angDiff\big(\emptyset(x,y), \emptyset_{ref}\big) x\ norm(\tau, \delta, tMHI_{\delta}(x,y))}{\sum_{x,y} norm(\tau, \delta, tMHI_{\delta}(x,y))} \quad (3)$$

$\overline{\emptyset}$ is the global motion orientation, $\emptyset_{ref}$ is the base reference angle (peaked values in the histogram of orientations), $\emptyset(x,y)$ the motion orientation map found from gradient convolutions, $norm(\tau, \delta, tMHI_{\delta}(x,y))$ a normalized tMHI value (linearly normalizing the tMHI from 0 to 1 using the current timestamp τ and duration δ) and $angDiff\big(\emptyset(x,y), \emptyset_{ref}\big)$ the minimum signed angular difference of an orientation from the reference angle.

A histogram-based reference angle $\emptyset_{ref}$ is needed due to problems associated with averaging circular distance measurements.

Motion segmentation

Motion segmentation is done to group motion vectors that were produced by the movement of parts or the whole object of interest.

Motion attached to object

Most recent silhouette has the maximal values in the tMHI. An image scanning is done until finding one of these values. Then, we walk along the most recent silhouettes contour to find attached areas of motion. We establish a *dT* interval which represents the time difference threshold, for example, time difference between two frames.

The algorithm for creating masks to segment motion regions works as explained below:

1. Scan tMHI until find a pixel of current timestamp. This is a boundary pixel of the most recent silhouette.
2. Walk around the current silhouette region looking outside for recent (within *dT*) unmarked motion history steps. When suitable step is found it is marked with a downward floodfill. If size of the fill is not big enough we zero out the area.
3. Store segmented motion masks that were found.
4. Continue the boundary walk until the silhouette has been circumnavigated.

In the image below "Fill Down" refers to flood fills that will fill (replaced with a labelled value) pixels with the same value one step (within *dT*) lower than the current pixel being filled.

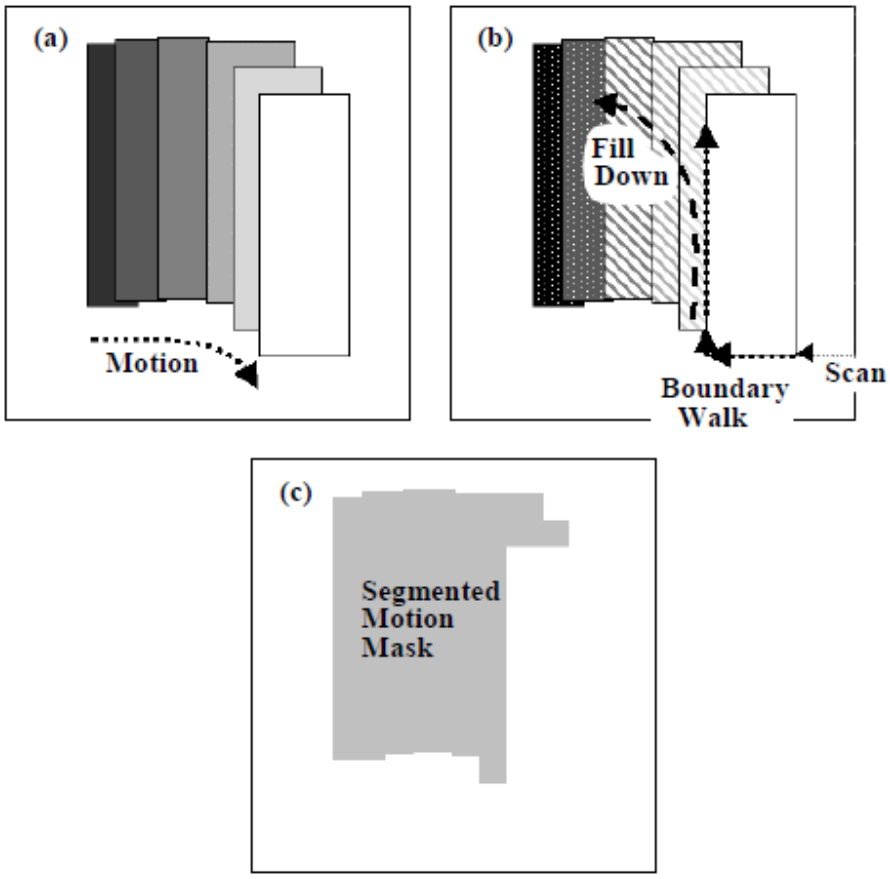


Figure 2.9. (a) tMHI from a moving block. (b) Find current silhouette region; walk the boundary and fill downwards wherever a step found; (c) Store the found motion masks. (Source: reference [5])

The segmentation algorithm is managed by two parameters:

- *dT:* maximum allowable downward step distance (how far back in time can a past motion be considered to be connected to the current silhouette).
- Minimum acceptable size of the downward flood fill (else zero it out because the region is too small. Motion "noise" region).

The algorithm above is based in segmentation masks used to select partitions of the valid motion history gradient. They can be labelled with their weighted regional orientation. The motion regions are directly connected to the object itself since they derive from past motion spilled from the current silhouette boundary.

CHAPTER 3

COMPARISON BETWEEN METHODS AND SELECTION OF THE WANTED ONES

This chapter shows a comparison depending on different parameters between the methods explained in chapter two and the selection of the best ones including the reasons, pros and cons.

THEORETICAL DIFFERENCES

The following table explains the main differences of each optical flow method presented in chapter two.

	LK	SF	TV-L1	FB
Energy	L2-norm (1 scalar)	L2-norm (3 scalars)	L1-norm (1 scalar) + total variation term* (minimization***)	Polynomial approximation**
Weights	1(contribution of each term to the direction)	2 (distance and color)	1 (between data fidelity and regul. force)	2 (points in the neighbourhood and control of boundaries)
Iterative	Yes	No	Yes	Yes
Sparse/Dense	Sparse	Dense	Dense	Dense
Others	NxN searching window. Extension to rotations and escalating. Stopping criterion.	Interpolation at smooth regions. Occlusion detection.	Stopping criterion before the total iterations (L2-based)	Estimates a priori displacement field.

Table 1. Theoretical differences.

*penalises strong variations in the displacement vector in order to obtain smooth displacement fields.
**every neighbourhood is approximated with a polynomial
***it computes "u" through its approx. "v" and auxiliary variables p1 and p2.
-for v or u fixed, we solve a minimization expression

It can be observed that the Lucas-Kanade, Simple Flow and the TV-L1 use a pixel difference based on L2-norm (LK, SF) or L1-norm (TV-L1, including a total variation term), while Farnebäck uses a polynomial approximation (however, it uses L2-norm for comparing polynomials later). L2-norm expresses quadratic values, giving more weight to the extremes than L1-norm. In case of, for example, existing white noise we could give too much importance to zones of the spectrum where the noise is much bigger than the signal. This would not represent properly what we want to identify. However, our application does not have to work with an important noise and this makes L2-norm being a good way for computing distances.

The SF algorithm works with color images and LK and FB with grey-scale images, which means working with one or three scalars. The recorded videos and images have a 3-channel structure. This implies that LK and FB need an "rgb" to grey-scale conversion.

All the methods hold a weight value which controls the difference criteria and are iterative, excepting the SF. Weighted values are used for giving more representation to the most important values. To be iterative implies to repeat certain operations certain times and also depending on past values, while non-iterative algorithms do not need to care about repetitions and depending on old values.

The unique sparse algorithm is the LK, so we could probably expect that this one will have a better running time when using a common region of interest for all the methods.

Other extra features are:

- Stopping iteration criterion of the LK and the TV-L1: it stops the iteration system if the resulting value is good enough before ending all the iterations.

- Interpolation at smooth regions and occlusion detection of the SF: following the smoothness criterion explained in chapter two, it is possible to interpolate regions where motion is not focused in and so gaining running time. The occlusion detection helps to identify when an object being tracked disappears behind some other one.

- Possible extension to rotations and scaling of the LK: this can help tracking the points of interest when regions change their scale or shape due to rotations.

- A priori displacement field of the FB: a priori prediction allows to getting close to the real point's new position at next frames. Thus, the difference computation takes a lower range because predicted point will be closer to the real one than without any previous information.

SIMPLE FLOW CASE

While testing the Simple Flow algorithm we observe that there are some difficulties in the running time for a real-time implementation. We finally concluded that this method is not able to work as we need and it is dismissed.

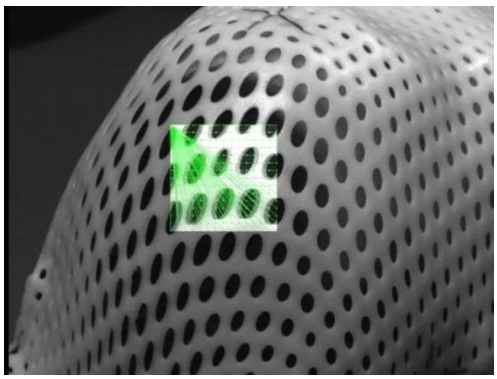
Observing the theoretical analysis it is seen that bilateral filters are used during the computation. These filters are for smoothing the image preserving the contours. In order to do this the coefficients of the filter need to be changed depending on the region of the image that is being treated. This fact represents a big computational cost, affects markedly to the running time and makes Simple Flow being a slow method.

The next table proves that conclusion, showing the running time and frame rate depending on different parameters.

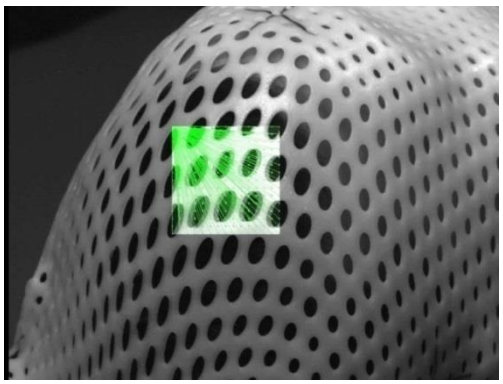
	Levels	Averaging block size*	Max flow	Running time (s)	Frame rate (fps)
SF (OpenCV default param.)	3	2	4	272	0.00367
SF (minimum parameters)	1	1	1	72	0.01388

Table 2. SimpleFlow case.

*When computing cost function for pixel.



10ph. Default parameters



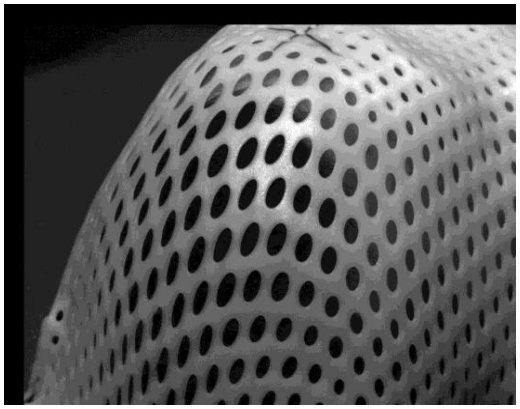
10ph. Min parameters

Furthermore, these configurations do not provide a good flow field and not all the flow is found as can be seen in captures above.

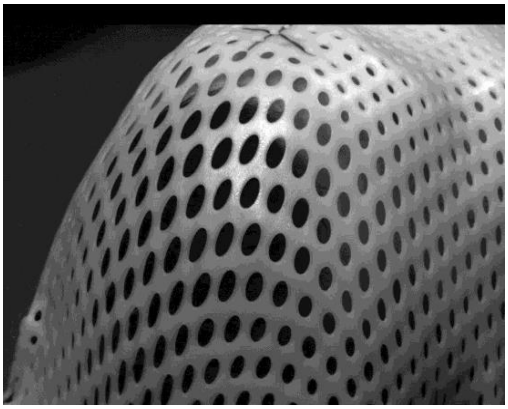
TESTED RESULTS

For the practical part, a frame was extracted from an original sequence and was displaced some pixels in different directions (horizontal, vertical and diagonal), which are used as a ground truth. The motion detection is computed for each algorithm fixing some parameters (same ROI, same runtime, etc.).

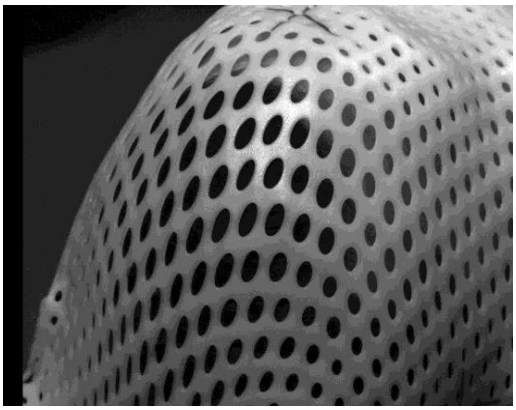
The following images show some displacement examples:



40 pixels «x,y» direction



40 pixels «y» direction



40 pixels «x» direction

The following displacements are done:

- 5, 10, 15, 20, 30, 40, 95, 190 pixels in the horizontal «(x,0)» positive direction (right).

- 5, 10, 15, 20, 30, 40, 95, 190 pixels in the vertical «(0,-y)» negative direction (down).

- 5, 10, 15, 20, 30, 40, 95, 190 pixels in the diagonal «(x,-y)» direction (right, down).

The next displacement detection results are shown fixing some parameters and comparing other ones in each algorithm. Three different tests are done:

1. Parameters established in order to work at 10 fps.
2. Fixing common parameters.
3. Fixing a Region Of Interest.

Parameters established in order to work at 10 fps

	LK	FB	FB(0.25)	TV-L1
$\mathbf{Levels}_1$	4/0,5	2/0,5	2/0,25	5/0,5
Iterations	20 (max)	1	2	1
Neighbourhood	40x40 (searching window)	2x2 (polynomial expansion)	2x2 (polynomial expansion)	-
ROI	600x550	225x225	225x225	225x225
Features	$150(\max)_2$	All pixels	All pixels	All pixels
Others	Epsilon = 0,03 (stopping criteria)	Avg. winsize = 50 (smooth image, against noise) $Sigma_3$= 1 Flags = 0	Avg. winsize = 50 Sigma = 2 Flags = 0	Tau_4 = 0,25 Lambda = 0,15 (weight) Warps_5= 1 Epsilon = 0,01

Table 3. Parameters for 10fps.

1. Pyramid levels/scale between levels. Every layer is 1/2 (0.5) or 1/4 (0.25) bigger than the next lower one.
2. Features found with *goodFeaturesToTrack()* method, which finds the most prominent corners in the image or in the specified image region. (http://docs.opencv.org/modules/imgproc/doc/feature_detection.html#goodfeaturestotrack)
3. Standard deviation of the Gaussian that is used to smooth derivatives used as basis for the polynomial expansion.
4. Time step of the numerical scheme.
5. Number of times that I1(x+u0) and grad(I1(x+u0)) are computed per scale. Ensures the stability of the method. Trade-off between quality and running time.

The following tables show the mean pixel displacement detected by the optical flow and its standard deviation.

The displacement's notation is understood as:

"Num. of" + "pixels" + "direction of the movement".

For example:

10ph = 10 pixels in horizontal direction.

Resultant vectors are read as follows:

- Average displacements: [x displacement, y displacement].
- Average standard deviation: [x deviation, y deviation].

The expected results to be obtained are all different pixel displacements in all different directions according to the actual displacements of the ground truth previously done. Hence, in case of a horizontal displacement of five pixels it is expected to obtain an average displacement vector as similar as possible to [5.0, 0.0] and an average standard deviation as small as possible [0.0, 0.0] (both ideal cases).

Methods which better fulfil these expectations are chosen as the best ones.

Horizontal displacements

Avg hor. displ.	LK	FB (0.5)	FB (0.25)(2iter)	FB (0.25)(1iter)	TVL1
5ph	[4.99984, 0.00134734]	[4.98506, 0.0007448]	[4.9813, 0.0042]	[5.014723, -0.0043047]	[2.34749, 0.653476]
10ph	[9.99771, 0.00477424]	[9.98569, 0.0117865]	[9.9891, 0.004518]	[9.994341, 0.0262794]	[3.3675, 0.975214]
15ph	[15.0595, 0.0460269]	[13.6162, 0.468119]	[14.9523, 0.015437]	[8.57792, 1.40839]	[3.49194, 1.21643]
20ph	[19.9665, -0.088947]	[6.6978, 1.75178]	[19.9143, 0.039766]	[5.452409, 1.309176]	[2.4104, 1.16223]
30ph	[-4.94359, 8.86255]	[-4.26669, -0.653512]	[-10.8604, -0.568842]	[-3.410196, -0.44795]	[-0.463063, 0.40637]
40ph	[-6.81518, -4.1994]	[-7.809, 1.52073]	[-13.0765, 5.832644]	[-6.43539, 1.120405]	[-2.28445, 0.391878]
95ph	[-3.71088, 8.03827]	[-4.10958, 2.51348]	[-8.213603, 6.58933]	[-3.426716, 2.220857]	[-2.07838, 0.100962]
190ph	[-3.9705, -23.1897]	[0.179788, -0.776041]	[0.38891, -2.35762]	[0.019967, -0.63657]	[-0.627039, 0.104125]

Table 4.1. Average horizontal displacements.

Avg std. dev.	LK	FB(0.5)	FB (0.25)(2iter)	FB (0.25) (1iter)	TVL1
5ph	[0.01001, 0.016648]	[0.02923, 0.01633]	[0.04415, 0.021765]	[0.02965, 0.015487]	[1.72381, 2.023143]

10ph	[0.030285, 0.041912]	[0.05509, 0.028237]	[0.092032, 0.031689]	[0.06218, 0.032938]	[3.3139, 3.21468]
15ph	[0.44926, 0.32756]	[1.374937, 0.51459]	[0.1386022, 0.047044]	[0.65132, 0.478512]	[4.43285, 3.892666]
20ph	[0.34148, 0.71083]	[0.59571, 0.43078]	[0.20247, 0.155437]	[0.42631, 0.352102]	[5.262707, 4.243175]
30ph	[23.1692, 17.30520]	[1.0371, 0.4739053]	[0.483386, 0.572852]	[0.877648, 0.414379]	[5.98005, 4.50276]
40ph	[8.10485, 30.9373]	[1.0944, 1.34515]	[1.546579, 3.698929]	[0.70086, 1.041005]	[4.61368, 3.96688]
95ph	[13.43199, 35.81834]	[1.1675, 0.86760]	[1.6965878, 1.128337]	[1.1387, 0.7915]	[5.5199, 4.2102]
190ph	[26.2546, 35.8367]	[1.22912, 1.132048]	[1.755236, 2.17163]	[1.051289, 1.264135]	[5.59865, 4.29532]

Table 4.2. Average standard deviation.

Vertical displacements

Avg vert. displ.	**LK**	**FB**	**FB (0.25)(2iter)**	**FB (0.25) (1iter)**	**TVL1**
5pv	[-0.01361, 5.378056]	[0.0035036, 4.98435]	[0.004096, 4.984595]	[-0.000473, 5.01139]	[0.43626, 1.49798]
10pv	[-0.00226, 10.873708]	[0.00109, 9.97935]	[-0.001432, 9.968469]	[0.008619, 9.99815]	[0.874742, 2.515049]
15pv	[-0.03195, 16.025611]	[0.168263, 14.48390]	[0.014885, 14.83778]	[1.18028, 8.81908]	[1.060398, 2.702613]
20pv	[-0.14729, 21.127639]	[2.272762, 7.272434]	[0.025278, 19.75997]	[2.01888, 5.57085]	[1.1451645, 2.75603]
30pv	[-0.134647, 30.27136]	[2.57982, 5.50527]	[3.306101, 13.93361]	[2.0872, 4.55502]	[1.038595, 1.95053]
40pv	[-0.90754, 34.897038]	[0.61636, 1.99853]	[2.130066, 6.961017]	[0.65107, 1.71346]	[0.173476, 1.143177]
95pv	[-5.037787, 18.24259]	[-2.10211, 8.686375]	[-2.823108, 11.21075]	[-1.47529, 7.01454]	[0.072385, 1.41722]
190pv	[-7.51526, 15.31906]	[-0.63014, 2.838724]	[-1.238965, 6.690215]	[-0.23955, 2.29519]	[-0.873524, 0.803187]

Table 5.1. Average vertical displacements.

Avg std. deviation	LK	FB	FB (0.25)(2iter)	FB (0.25) (1iter)	TVL1
5pv	[0.10948, 2.62964]	[0.00964, 0.03437]	[0.009039, 0.052554]	[0.010146, 0.03592]	[1.25285, 1.39292]
10pv	[0.13839, 4.48701]	[0.010516, 0.09926]	[0.012307, 0.1505169]	[0.01276, 0.094886]	[2.40455, 2.579464]
15pv	[0.40946, 4.534629]	[0.19992, 0.69644]	[0.047224, 0.41426]	[0.423432, 0.64669]	[3.52352, 3.343636]
20pv	[1.96445, 5.085027]	[0.43943, 0.68886]	[0.076315, 0.665646]	[0.481218, 0.609295]	[4.34142, 3.446348]
30pv	[1.96195, 1.91924]	[0.40858, 0.72817]	[0.47156, 1.20815]	[0.3986, 0.49375]	[5.524503, 4.40014]
40pv	[3.41960, 16.00431]	[0.62804, 0.912690]	[0.566798, 1.297629]	[0.58932, 0.873189]	[5.64955, 4.53825]
95pv	[5.331406, 19.16260]	[1.907729, 2.83529]	[1.90273, 4.48496]	[1.43343, 1.35683]	[4.138924, 3.30172]
190pv	[14.3010, 50.32497]	[0.91718, 2.62048]	[1.451545, 3.739478]	[0.46693, 2.071624]	[4.582428, 4.90072]

Table 5.2. Average standard deviation.

Diagonal displacements

Avg diag. displ.	LK	FB	FB (0.25)(2iter)	FB (0.25) (1iter)	TVL1
5pd	[4.985517, 5.379528]	[4.987461, 4.993966]	[4.9846, 4.99416]	[5.01239, 5.00607]	[2.3927, 1.92777]
10pd	[9.995974, 10.85413]	[9.907547, 9.761286]	[9.97628, 9.93457]	[7.38259, 5.08417]	[3.071619, 2.4118]
15pd	[15.02889, 16.08815]	[6.0151, 1.7659]	[14.8508, 14.03658]	[5.056087, 1.47397]	[2.04113, 1.96299]
20pd	[19.82180, 21.03489]	[-0.093959, 0.258317]	[1.491068, 0.389589]	[0.10141, 0.15946]	[0.865009, 1.3247]
30pd	[-14.2391, 10.89695]	[-2.26426, -0.39538]	[-4.82879, 1.408813]	[-2.024798, -0.27348]	[-1.2332, 0.245108]
40pd	[-9.85606, 7.217449]	[0.07429, 4.33641]	[-1.737295, 9.85784]	[-0.12485, 3.53569]	[-0.956001, 0.83578]
95pd	[-10.2963, 13.9119]	[-1.49552, 0.93391]	[-4.04312, 3.21219]	[-1.30066, 0.804057]	[-1.57599, 0.1907349]
190pd	[20.2197, -20.1811]	[0.259998, -3.39575]	[0.80659, -5.52032]	[0.176512, -2.920468]	[0.295641, -0.017594]

Table 6.1. Average diagonal displacements.

Avg std. deviation	LK	FB	FB (0.25)(2iter)	FB (0.25) (1iter)	TVL1
5pd	[0.11238, 2.61762]	[0.02832, 0.034112]	[0.04289, 0.054035]	[0.046142, 0.041966]	[2.398785, 2.34832]
10pd	[0.14790, 4.38037]	[0.321979, 0.71825]	[0.081906, 0.167869]	[0.606779, 0.653306]	[4.573298, 3.544003]
15pd	[0.610809, 4.598969]	[0.653098, 0.463353]	[0.46986, 1.632327]	[0.55036, 0.49405]	[5.65165, 3.935056]
20pd	[1.983482, 5.147813]	[1.21275, 1.051149]	[1.67267, 1.32746]	[0.86667, 0.88412]	[6.089509, 4.251306]
30pd	[13.53965, 31.07589]	[1.17395, 0.437554]	[1.94812, 0.5121525]	[0.96533, 0.387728]	[6.048216, 4.175616]
40pd	[10.43187, 27.57055]	[1.62703, 0.703576]	[1.94987, 0.81946]	[1.14160, 0.462398]	[6.33168, 4.46824]

95pd	[17.06462, 38.3422]	[1.62531, 0.900128]	[2.438537, 1.451698]	[1.385537, 0.65799]	[5.61155, 4.004356]
190pd	[31.4083, 58.70947]	[1.17753, 1.85779]	[1.45465, 2.049323]	[1.02806, 1.42658]	[4.254015, 3.412049]

Table 6.2. Average standard deviation.

Observations

Analysing these tables we can do the following observations:

- For horizontal displacements, the FB (0.25. 2 iterations) is the best method since it has a better standard deviation than the LK.
- For vertical displacements, the LK is the best one since it detects a 30 pixel displacement, followed by the FB.
- For diagonal displacements, the LK is better than the FB as it identifies a 20 pixel displacement.

Then, we can conclude that the best method in this case is the LK, followed close by the FB.

Common parameters fixed

The common parameters of the algorithms are not so many, but some of them can be fixed as next table shows.

	LK	FB	TV-L1 (warps=5)	TV-L1 (warps=1)
Levels	3/0,5	3/0,5	3/0,5	3/0,5
Iterations	5	5	5	5
ROI	600x550	225x225	225x225	225x225

Table 7.1. Common parameters fixed.

With this implementation, the following frame rates are obtained.

	LK	FB	TV-L1 (warps=5)	TV-L1 (warps=1)
fps	12	5-6	1	5

Table 7.2. Frame rates obtained with fixed parameters.

Once more, the Lucas-Kanade method is the best one in terms of frame rate.

The average displacements and standard deviations look as follows.

Horizontal displacements

Avg horiz. displ.	LK	FB	TVL1
5ph	[4.9995, 0.00125]	[4.98286, 0.00406]	[4.97438, 0.039143]
10ph	[9.997597, 0.004914]	[9.9893, 0.00846]	[9.13158, 0.278767]
15ph	[15.05646, 0.04567]	[14.97785, 0.0103117]	[9.40783, 1.418737]
20ph	[18.6269, 0.8440984]	[19.97179, -0.002898]	[7.592596, 1.80758]
30ph	[-11.55658, 15.40848]	[-24.5319, 9.68196]	[0.28754, 1.29204]
40ph	[-6.845435, 9.043778]	[-14.64869, 10.13253]	[-4.4184, 1.74297]
95ph	[-3.635612, 3.926243]	[-13.0691, 17.09149]	[-4.100047, 0.50576]
190ph	[0.291174, -3.5017869]	[3.89707, -10.87025]	[-0.084296, 0.41297]

Table 8.1. Average horizontal displacements.

Avg std. deviation	LK	FB	TVL1
5ph	[0.009937, 0.016585]	[0.04438, 0.022557]	[0.137002, 0.170785]
10ph	[0.030297, 0.041910]	[0.097418, 0.0319018]	[1.835784, 0.562304]
15ph	[0.428342, 0.3250307]	[0.15387, 0.05356]	[4.919454, 1.914956]
20ph	[6.009035, 4.309556]	[0.1907818, 0.079976]	[6.13637, 3.416206]
30ph	[16.25517, 18.82032]	[1.868619, 8.95372]	[8.049997, 4.738297]
40ph	[8.23442, 25.11331]	[1.782903, 9.251066]	[5.796616, 4.598502]
95ph	[11.62761, 20.796439]	[2.80512, 4.13946]	[7.254198, 4.986413]
190ph	[14.19549, 18.59725]	[3.09788, 3.805626]	[8.1845736, 5.3466464]

Table 8.2. Average standard deviation.

Vertical displacements

Avg vert. displ.	LK	FB	TVL1
5pv	[-0.01441, 5.1853478]	[0.00338, 4.98656]	[0.013479, 4.9959759]
10pv	[-0.0069449, 10.474729]	[-0.0012155, 9.966124]	[0.413863, 8.5566376]
15pv	[-0.0087313, 15.679012]	[0.006889, 14.836859]	[1.54625, 8.783728]
20pv	[-0.042001, 20.68615]	[0.012872, 19.80402]	[2.028967, 8.3580679]
30pv	[-0.2475998, 29.93038]	[0.078133, 29.475588]	[2.255446, 6.537654]
40pv	[-1.4292, 25.82048]	[0.582746, 37.68714]	[1.31323, 3.743277]
95pv	[-4.935345, 14.396408]	[-3.043792, 11.190488]	[-0.875596, 5.010505]
190pv	[-7.70838, 13.33551]	[-4.166886, 23.58706]	[-2.4499, 2.456117]

Table 9.1. Average vertical displacements.

Avg std. deviation	LK	FB	TVL1
5pv	[0.111735, 1.28643]	[0.009004, 0.053127]	[0.0947459, 0.108834]
10pv	[0.0854447, 2.484987]	[0.01301, 0.1595678]	[0.61348, 1.6145175]
15pv	[0.1373036, 3.04238]	[0.059336, 0.479862]	[1.951997, 2.674337]
20pv	[0.59377, 2.8891462]	[0.054755, 0.58809]	[3.64967, 3.502829]
30pv	[1.954413, 3.7288418]	[0.173109, 0.96862]	[5.88189, 4.522116]
40pv	[3.603253, 25.168029]	[1.1236, 4.15865]	[7.300709, 6.1805789]
95pv	[5.612721, 12.36915]	[1.99887, 5.67565]	[4.43622, 4.848916]
190pv	[9.4274372, 24.8711]	[3.242656, 5.93843]	[6.829946, 7.6540837]

Table 9.2. Average standard deviation.

Diagonal displacements

Avg diag. displ.	LK	FB	TVL1
5pd	[4.985019, 5.187826]	[4.984874, 4.995013]	[4.94822, 4.97806]
10pd	[9.98999, 10.48022]	[9.988275, 9.96154]	[7.6929, 7.4468429]
15pd	[14.95005, 15.18277]	[14.98158, 14.83758]	[5.30543, 5.380756]
20pd	[18.01087, 16.04952]	[19.9991, 18.59948]	[2.890539, 4.09839]
30pd	[-8.21079, -2.868057]	[-17.7356, 18.3366]	[-2.54794, 1.935316]
40pd	[-7.660479, 5.648376]	[-11.18455, 33.55788]	[-1.74802, 1.62423]
95pd	[-4.81727, 6.5244467]	[-16.27244, 18.44536]	[-3.019158, 0.808949]
190pd	[2.926358, -5.271628]	[2.266307, -10.895]	[0.22281, 0.218062]

Table 10.1. Average diagonal displacements.

Avg std. deviation	LK	FB	TVL1
5pd	[0.112009, 1.27997]	[0.0412759, 0.056696]	[0.226758, 0.212298]
10pd	[0.08160, 2.492818]	[0.096413, 0.167304]	[3.966918, 1.789949]
15pd	[0.982669, 5.713568]	[0.1766638, 0.529974]	[6.9327143, 3.701658]
20pd	[5.762089, 14.59863]	[0.467942, 2.569607]	[7.960557, 4.928768]
30pd	[17.1437, 28.3125]	[3.632087, 2.8560727]	[8.963633, 5.422945]
40pd	[9.86138, 22.4595]	[3.234495, 4.9516102]	[7.903225, 5.9682267]
95pd	[13.6931, 21.25693]	[2.544027, 2.8910864]	[7.689733, 5.2551329]
190pd	[13.15028, 23.77543]	[1.965384, 3.5653486]	[6.169868, 4.39883]

Table 10.2. Average standard deviation.

Observations

The results show that this time Farnebäck's is the best algorithm in each three different displacements, identifying a forty pixels vertical displacement and showing better standard deviations, although LK's gets good results too.

Analysis fixing the Region Of Interest

A common intermediate ROI is set and fixed for all the methods.

	LK	FB	TV-L1 (warps=1)
ROI	350x350	350x350	350x350

*Table 11.1.*Fixed common ROI.

Resulting frame rates are:

	LK	FB	TV-L1 (warps=1)
fps	20	5	5

Table 11.2. Frame rates for fixed common ROI.

The different tables of results are the following.

Horizontal displacements

Avg horiz. displ.	LK	FB	TVL1
5ph	[5.12409, -0.729439]	[4.98656, 0.002298]	[3.304598, 0.7013345]
10ph	[9.987766, -0.729224]	[9.97907, 0.0061859]	[5.64731, 1.287852]
15ph	[15.0979, -0.280285]	[14.93269, 0.014986]	[6.351489, 1.367756]
20ph	[20.77364, -0.036638]	[19.59819, 0.102876]	[5.005138, 1.1017602]
30ph	[-21.27651, 0.3023917]	[-8.24087, -0.9401609]	[-0.95423, -0.2665746]
40ph	[-12.76466, -1.373418]	[-11.742195, 0.549124]	[-2.837203, -0.24891]
95ph	[-10.2736, -2.679787]	[-7.624821, 3.442834]	[-2.474645, -0.361153]
190ph	[6.554083, -29.258073]	[-1.007246, 0.4379109]	[-0.172615, -0.227635]

Table 12.1. Average horizontal displacements.

Avg std. deviation	LK	FB	TVL1
5ph	[1.011259, 5.960629]	[0.059224, 0.0202276]	[1.758805, 2.397925]
10ph	[0.1707649, 5.954908]	[0.111286, 0.0215209]	[3.41044, 4.403962]
15ph	[0.674397, 2.1158313]	[0.207445, 0.0332279]	[5.79759, 6.124823]
20ph	[3.0742078, 0.698006]	[1.02685, 0.3057989]	[7.09286, 7.047809]
30ph	[11.403228, 22.20615]	[2.70347, 1.393475]	[7.408157, 7.6697294]
40ph	[4.69241, 23.0294]	[1.828978, 7.3368579]	[6.23292, 7.550006]
95ph	[8.174438, 43.25827]	[1.788394, 3.13467]	[7.59437, 7.887819]
190ph	[24.7782, 31.066894]	[2.474055, 2.690558]	[7.68344, 7.26176]

Table 12.2. Average standard deviation.

Vertical displacements

Avg horiz. displ.	LK	FB	TVL1
5pv	[0.002579, 6.00269]	[0.0032936, 4.98501]	[0.20238, 3.4139486]
10pv	[-0.012412, 10.895109]	[-0.0030123, 9.990703]	[0.4712459, 5.999012]
15pv	[-0.004242, 15.646064]	[0.0065874, 14.946836]	[0.674335, 7.6615414]
20pv	[-0.0583353, 21.30921]	[0.0281857, 19.804645]	[0.861633, 7.702219]
30pv	[0.0273216, 31.58853]	[2.965536, 13.08949]	[0.492725, 4.8130045]
40pv	[-0.8651212, 38.41146]	[1.78014, 8.957887]	[-0.35716, 0.5288625]
95pv	[-2.938299, 12.18923]	[-2.618976, 7.9427368]	[-1.75659, 3.864309]
190pv	[-7.08305, 30.01003]	[-3.248853, 5.686355]	[-2.748021, 2.182636]

Table 13.1. Average vertical displacements.

Avg std. deviation	LK	FB	TVL1
5pv	[0.0300005, 5.719203]	[0.017698, 0.071898]	[1.38006, 1.53234]
10pv	[0.409804, 5.1051377]	[0.025016, 0.1019468]	[2.60019, 2.98055]
15pv	[0.5555319, 3.828344]	[0.0355097, 0.145287]	[4.05331, 4.30893]

20pv	[0.8474428, 5.715665]	[0.108799, 0.604185]	[5.24183, 5.268895]
30pv	[1.276986, 6.540956]	[0.868291, 1.6461344]	[7.354281, 7.031596]
40pv	[6.690667, 14.489335]	[0.984982, 3.0516539]	[7.82001, 8.22350]
95pv	[3.417633, 10.859895]	[2.74269, 5.620043]	[5.470495, 7.04007]
190pv	[7.588072, 29.14663]	[3.02642, 4.54146]	[7.063241, 8.5876916]

Table 13.2. Average standard deviation.

Diagonal displacements

Avg horiz. displ.	**LK**	**FB**	**TVL1**
5pd	[5.100172, 5.1878632]	[4.984738, 4.998369]	[3.3657803, 4.026659]
10pd	[9.958292, 10.146872]	[9.960889, 9.973809]	[5.246689, 6.0761118]
15pd	[15.043503, 15.736491]	[14.7986, 14.250617]	[3.208403, 4.899987]
20pd	[20.74433, 21.26949]	[4.0356, 1.56311]	[-0.9768315, 2.214335]
30pd	[-21.62755, 27.774207]	[-7.63227, 1.175486]	[-3.376975, 0.723109]
40pd	[-14.08627, 25.431308]	[-4.23053, 10.35461]	[-2.89196, 0.00324278]
95pd	[-16.12111, 32.02653]	[-5.70913, 2.735067]	[-2.310804, -0.076847]
190pd	[30.11132, -21.313225]	[2.05944, -3.440995]	[-0.0826684, 0.257866]

Table 14.1. Average diagonal displacements.

Avg std. deviation	**LK**	**FB**	**TVL1**
5pd	[1.025568, 7.999229]	[0.067905, 0.0776358]	[2.424205, 2.781635]
10pd	[0.424514, 7.8952738]	[0.145464, 0.091712]	[4.93134, 5.200391]
15pd	[0.6264249, 5.1894005]	[0.611476, 1.057908]	[8.0694492, 6.760094]
20pd	[3.1847358, 5.7864921]	[3.992808, 2.522754]	[8.638755, 8.0110117]
30pd	[9.8031798, 25.508949]	[4.433147, 1.3150928]	[8.0495404, 7.9706034]
40pd	[6.1243354, 25.425722]	[3.639137, 1.8013809]	[7.6721529, 7.801209]
95pd	[9.991651, 40.111808]	[2.672391, 2.1444898]	[7.763101, 8.0024202]
190pd	[33.88423, 52.01455]	[1.986633, 2.9766304]	[6.9827302, 6.2500758]

Table 14.2. Average standard deviation.

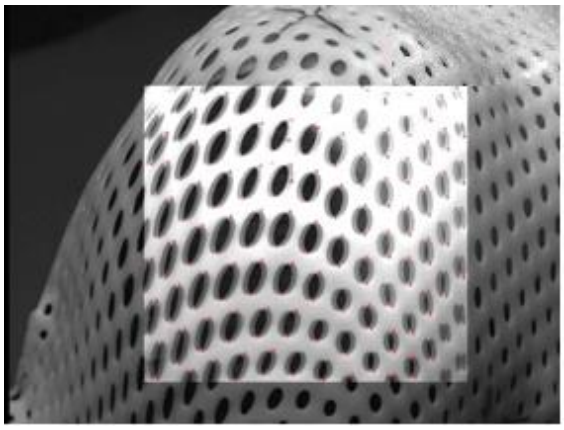
Observations

The analysis of these tables permits to do the following observations:

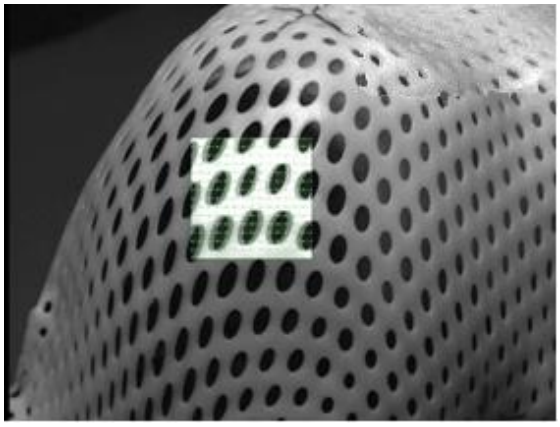
- For horizontal displacements, the FB is the best method since it has a better standard deviation than the LK.
- For vertical displacements, the LK is the best one since it detects a 40 pixel displacement.
- For diagonal displacements, the LK is better than the FB, identifying a 20 pixel displacement.

We can say that LK is the best method in this case, counting on the variations in the standard deviation. This standard deviation is still worse than FB's, due to the fact that it has fewer features to track and bad detections have more weight than in the FB method.

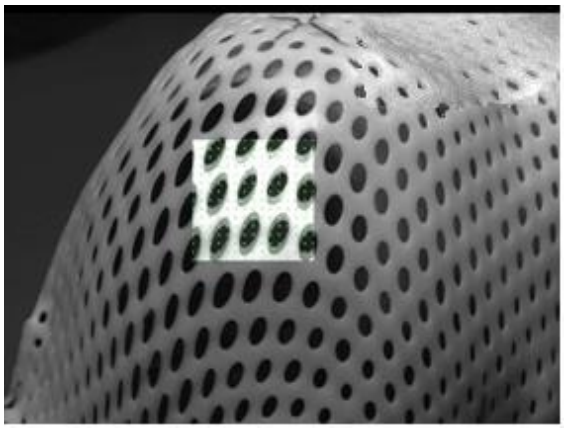
Next pictures show motion detection examples which have been tested.



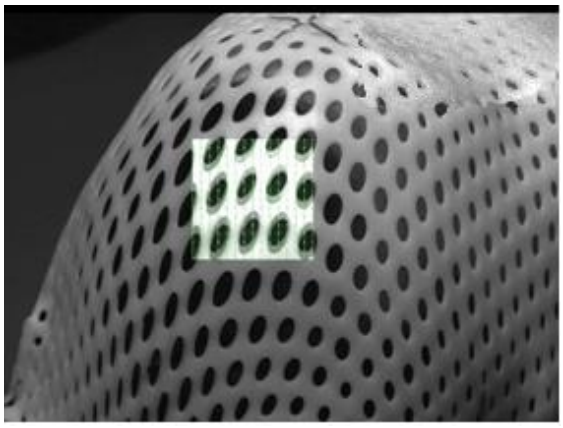
FB 10ph at 10fps configuration



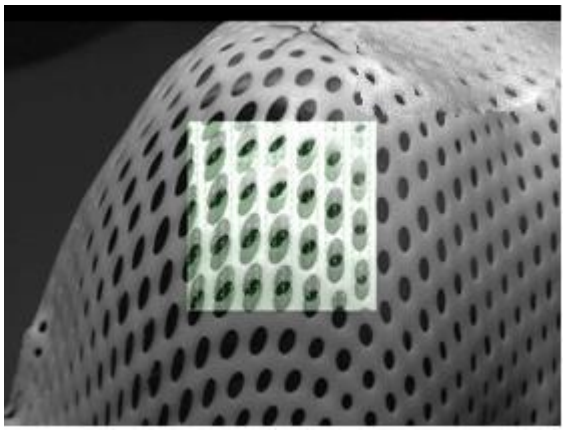
LK 10ph at 10fps configuration



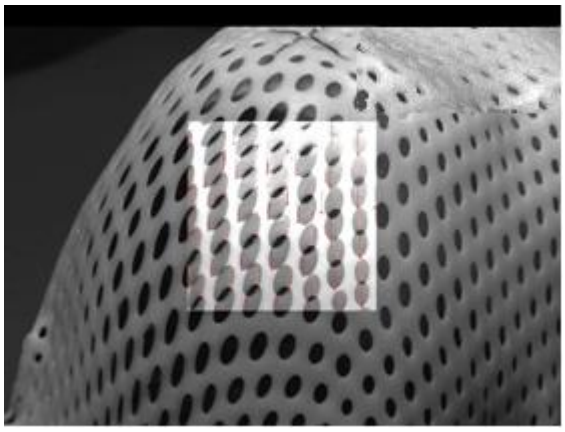
TVL1 15pv at 10fps configuration



FB 15pv at 10fps configuration



LK 40pv at fixed ROI configuration



FB 30pv at fixed ROI configuration

Conclusions

- Lucas-Kanade shows the best results in 10fps and fixed ROI approaches.
  - Pros:
    - Good results and high frame rate.
    - It is the method that detects the farthest movement.
    - It allows the biggest ROI.
  - Cons:
    - Irregular standard deviation (due to the number of features tracked).

- Farnebäck is a good option as a reliable dense method.
  - Pros:
    - Data uniformity (small standard deviation).
    - It follows closely the LK method in terms of detecting far movement.
  - Cons:
    - It needs a small ROI for fast working in comparison with LK.
    - It is more difficult to establish the good parameters for high frame rate working than LK.

- Analysing the results deeply, it can be checked that detection is better at "y" axis (vertical displacement) in terms of distance. This fact is due to the structure of the signal. There is a relation between the holes of the mask and the quantity of movement. These holes have an elliptical shape which explains that difference. The biggest diameter of the ellipse is y-oriented and the signal, since it is plenty of holes, shows a kind of periodical behaviour. The horizontal period of the signal is shorter than the vertical one because of the smaller diameter in "x" direction. Hence, x-motion will not be able to be detected as far as y-motion, because the hole next to will disturb the correct search.

- TV-L1 does not present a good configuration for real-time working, although it could do the desired job since it is possible to not care about the direction as long as some motion is detected.

- Simple Flow is not able to work in real-time because of its high computation time.
- A ROI is needed in order to do all the tasks properly. It is possible to manage the ROI dimensions in order to obtain a high frame rate. It is a trade-off between controlling the whole space and running time.
- The videos used for testing need to be compressed in order to be able to work with them.

SELECTION OF THE MOST INTERESTING METHODS

Once the three optical flow methods are tested, we need to decide which ones are useful for our needs and scopes. The Motion History Images algorithm is going to be considered too, in order to be able to work with another kind of motion detection method.

From four optical flow algorithms which were presented at the beginning, two of them can be rejected after proving that they are not able to work in real-time, because of a high running time, as happens in the Simple Flow, or bad results, in the TV-L1. So there are two methods left.

As it has been tested, commented and concluded, the Lucas-Kanade and the Farnebäck algorithms work pretty well in real-time and they are clear candidates for accomplish the scopes of the thesis and provide liable motion information.

Consequently, LK and FB both are going to be selected, implemented and integrated within Medusa. Also, MHI will be chosen since it does not require too many extra adaptations and will not suppose too many extra time to spend, giving to the user one more option for the motion analysis.

Hence, because of the testing results and the possibility of counting on another different method, three different motion detection ways are going to be available for working with.

Other possible methods and options

There are three other methods, listed in chapter two, which have not been implemented and tested due to a lack of time. These methods are the CamShift, the Kalman filter and the Foreground/Background subtraction. The first one was classified as “not priority” because of it works with a color model, instead of a grey-scale model. The other two were not tested because there was no time enough according to the project time plan and optical flow was considered a better option at the beginning, when deciding the best methods.

Another extra algorithm could have been the Block Matching, which has been deprecated in OpenCV and it is no longer available. Consequently, the priority of this method was established as a second option.

CHAPTER 4

INTRODUCTION TO *MEDUSA*

When the new facilities of the *Centre de Protonthérapie d'Orsay* were inaugurated, new software for monitoring the treatments was developed. This software was called *Medusa* and is the tool used at present in *CPO.*

Medusa has the responsibility of monitoring and controlling a proton therapy session using edge detection algorithms and contours definition.

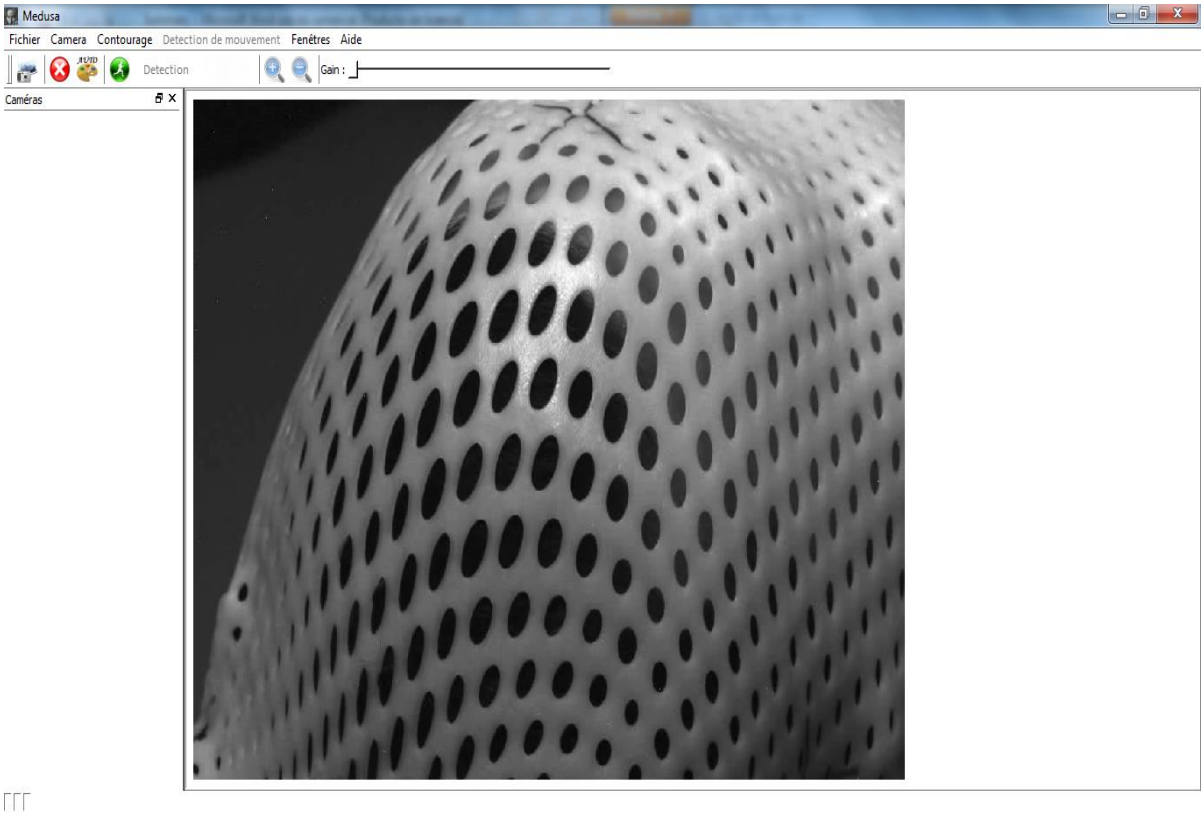


Figure 3.1. Medusa's main window

The picture above is the main window of the version before any implementation. It can be seen that motion detection menus are not available.

The main center window displays the image of the camera which is being used. At left, the "*Caméras*" window shows other available cameras in case of using more than one.

The upper menus provide all the different features as capturing a frame, start and stop the camera, histogram equalization functions, different contour detection methods and options, help window, a configuration window (shown below) with different parameters related with colors of drawing, the type of the cameras, etc.

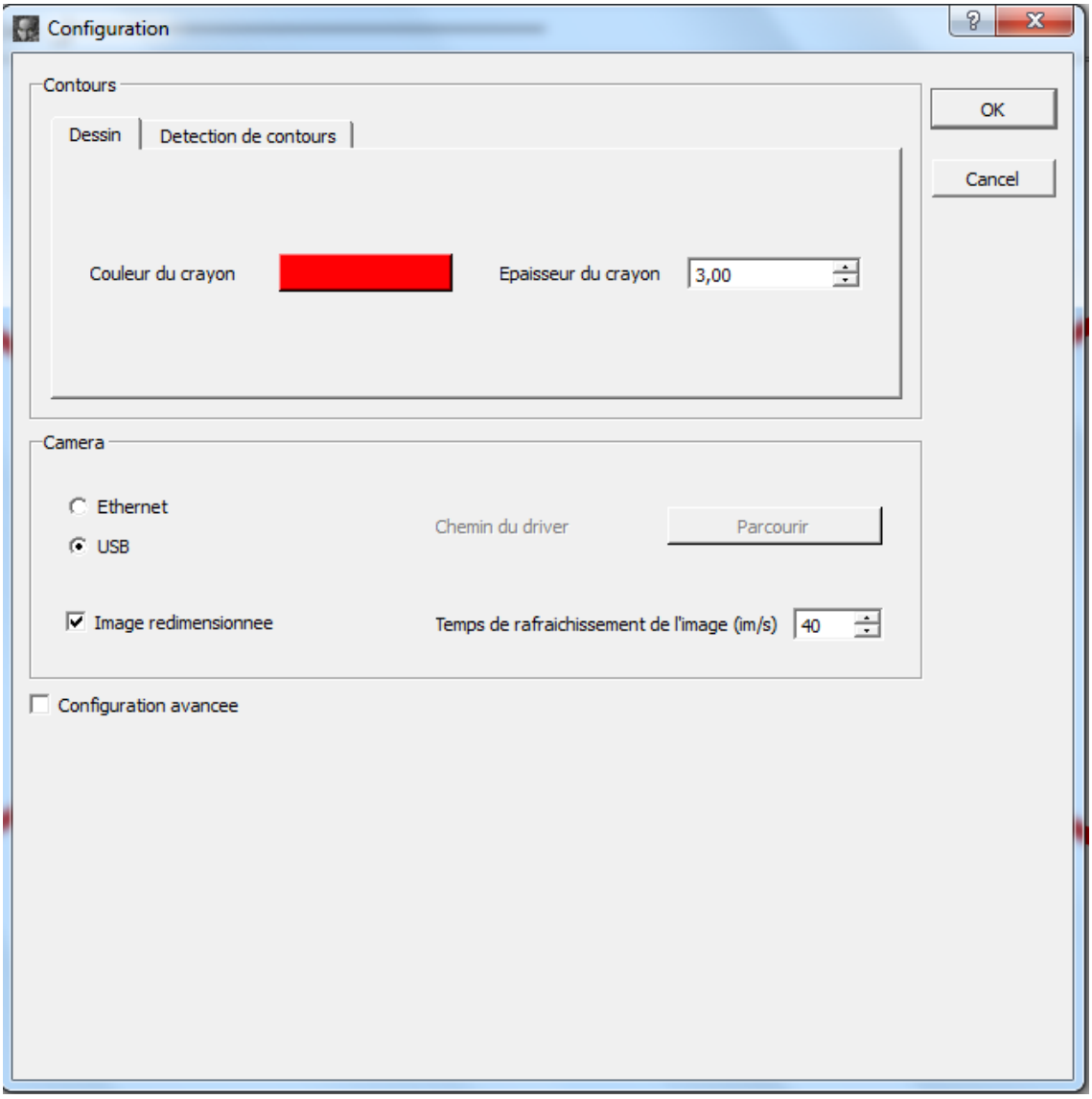


Figure 3.2. Medusa's configuration window

The actions bar below the menus allows the user to capture a frame, to erase the contours drawn, to enable the automatic detection of the contours, to detect the motion (disabled), to do some zoom in/out and increase/decrease the image brightness.

Basic operating mode

The patient is put in a correct position according with the zone to treat. The contours of the image from the camera are used as a reference position and are defined automatically and manually both, in case of head or sacrum treatment, and only manually in case of eye treatment.
Once the contours are correctly defined the beam irradiation starts. This process takes fifty or sixty seconds. If any motion from the patient occurs, therapists stop the beam. When patient remains quiet, the irradiation is resumed. This procedure is followed until the 50-60 seconds of irradiation.

Medusa architecture

Medusa is developed in C++, using the *OpenCV* and *Qt* libraries for processing images and creating the graphical interface.

There are two main blocks. One of them (*fiche_CrayonOptique.cpp* and *winConfig.cpp*) controls the graphical user interface including all the visualization functions, and the other one (*IC_videoWidget.cpp*) controls the different processing tools for every frame including drawing functions.

Through *IC_videoWidget.cpp* we use three important objects:

- Camera (created as *"camera"* and defined in *IC_camera.cpp*)
- Drawing (created as "*dessin*" *and* defined in *IC_dessin.cpp*)
- Motion detection (defined in *IC_mouvement.cpp but not used*)

- *IC_camera.cpp*:

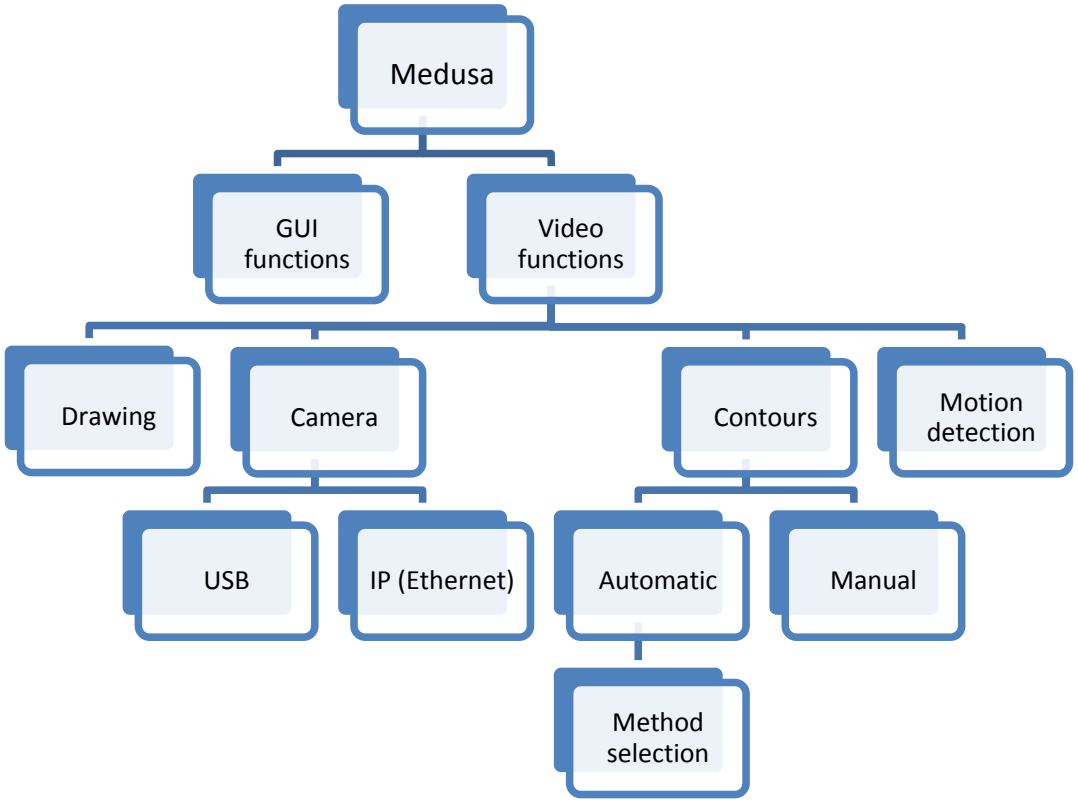
This file uses two more files which are inherited. *IC_camUSB.cpp* manages an USB camera and *IC_camIP.cpp* for an Ethernet camera. Generally, the Ethernet one is used.

- *IC_dessin.cpp*:

It does all the contour actions. Different contour detection methods are available (Suzuki, Prewitt, Sobel, Cani).

- *IC_mouvement.cpp*:

The responsible of motion detection features. It had got some motion detection algorithms implemented, but actually they were not being used and not correctly working. They were



very first attempts. Hence, they were removed for implementing and integrating the new ones.

Next scheme shows more visually that structure:

The contour detection looks as shown in the next image:

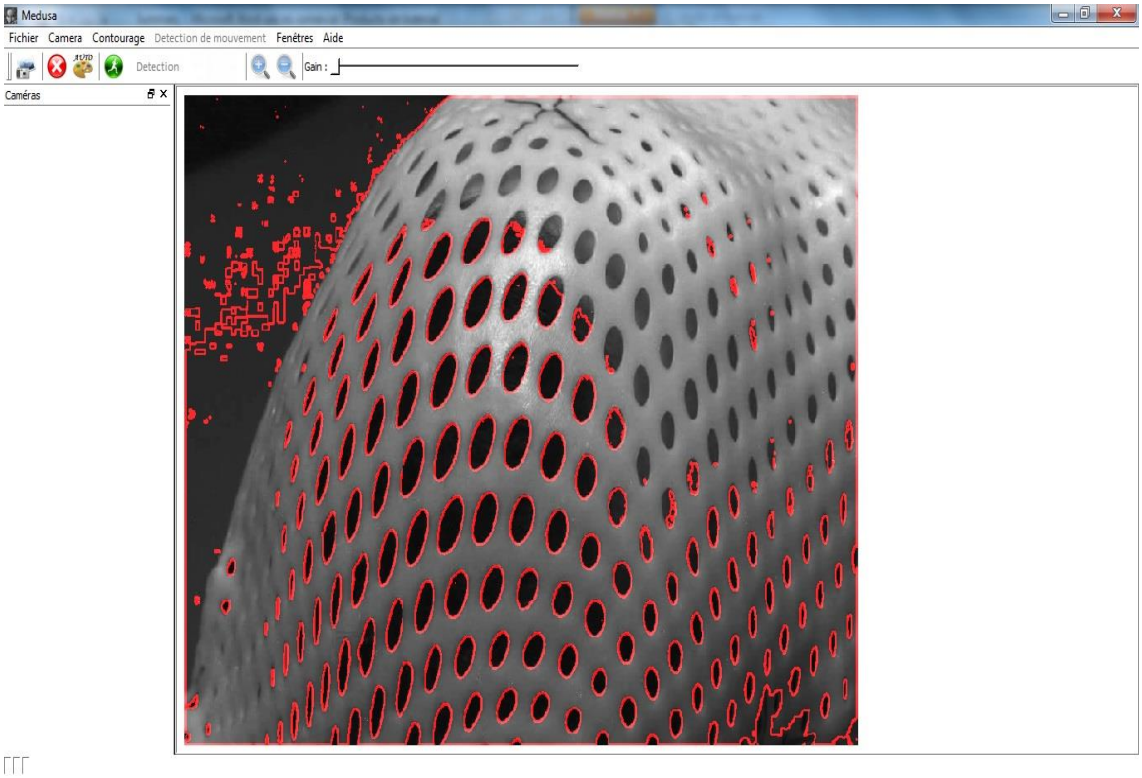


Figure 3.3. Medusa's main window. Contours (Suzuki) enabled.

Then, motion detection features have been integrated within that scheme, executing all changes needed.

INTEGRATION OF THE MOTION DETECTION ALGORITHMS

The main class which manages all the motion features is, as said before, *IC_mouvement.cpp.* So this class has been redeveloped and fitted in general scheme of *Medusa*.

We removed all old motion detection objects and previous functions which are not being useful anymore, and we renamed the class as *IC_motionDetection.cpp.* The old class had one different object for every motion detection algorithm. In order to do this question simpler, the new class has been declared as mother class and a new inherited class have been created for each motion detection method. With this configuration one single object is needed, fact that allows to simplify motion detection implementation.

Since we are working with three motion methods, three inherited classes are created. Those classes are *IC_motionLK.cpp, IC_motionFB.cpp* and *IC_motionMHI.cpp*.

Each class is responsible of the next main actions:

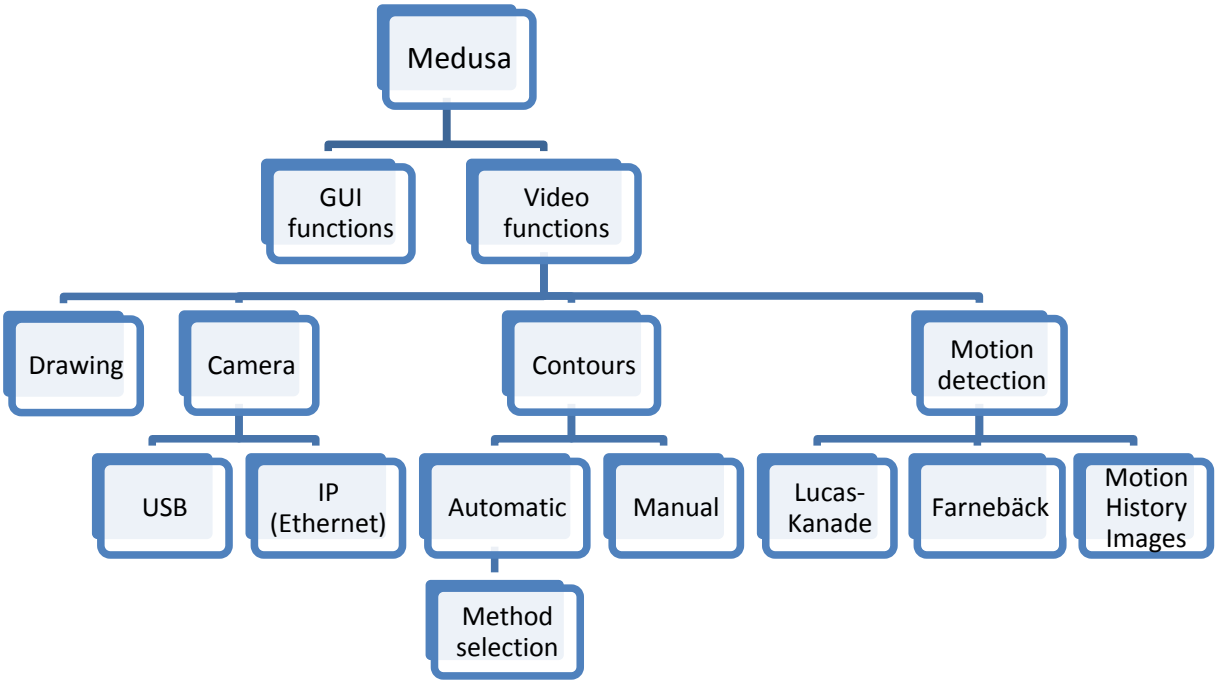
- Get the current frame.
- Analyse the motion and construct a flow field (LK and FB) or a gradient field (MHI).
- Check the motion difference between the current frame and the frame of reference.
- Control the status signal and change it if necessary.
- Execute all the painting functions which are related with the motion detection.

At the end of the whole implementation the description of the motion detection block would be the following:

- *IC_motionDetection.cpp*:

It is the responsible of all motion detection features. *IC_motionLK.cpp, IC_motionFB.cpp* and *IC_motionMHI.cpp* are inherited classes which define and compute the different detection types.

Then, the new scheme for *Medusa* would be:



With this new features and changes *IC_videoWidget.cpp* have got an object called "*moDetectThread*" which manages the motion detection process inside this class. Then, *IC_videoWidget.cpp* interacts with *fiche_CrayonOptique.cpp* and *winConfig.cpp* for linking the GUI and the motion detection thread, as it does with the other objects and threads.

GUI modifications

Looking now at GUI changes, motion detection menus are activated and some new features have been implemented.

The menu called "*Detection de mouvement*" is now active and has the three methods in order to choose the wanted one.

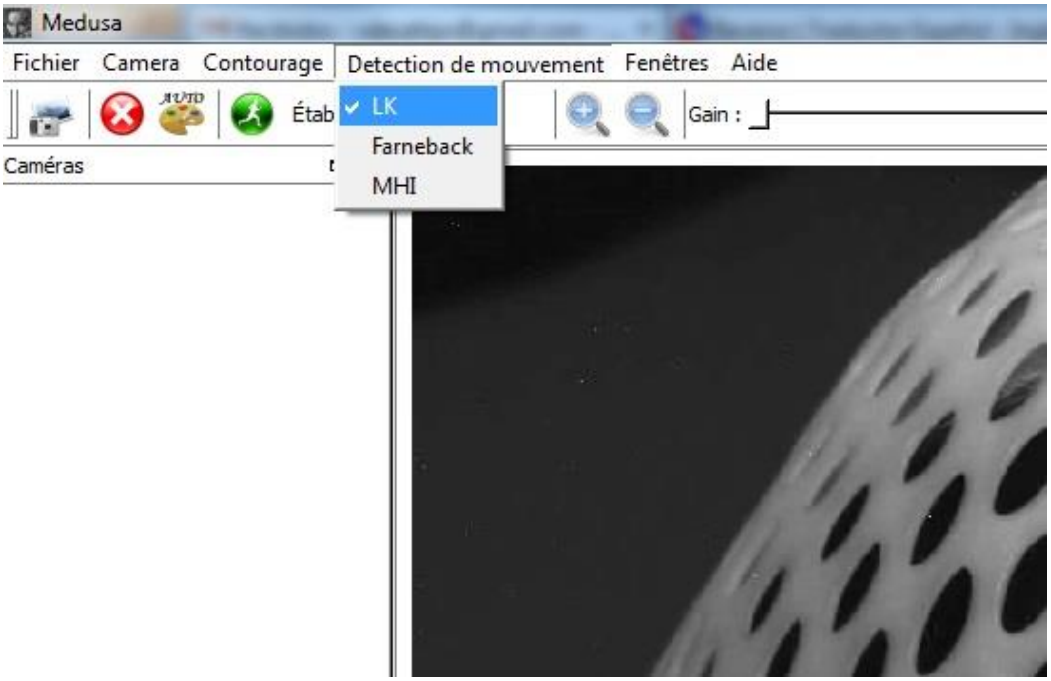


Figure 3.4. New motion detection menu

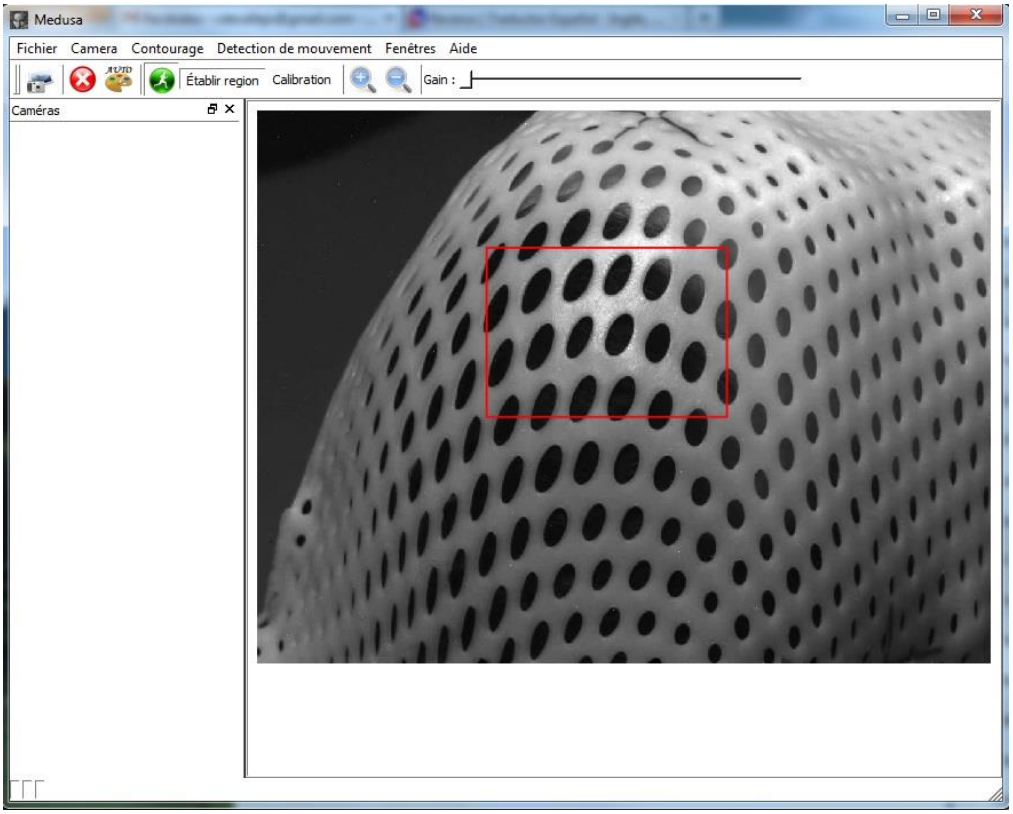


Figure 3.5. ROI selection

Two new functionalities are created. The start/stop motion detection button (which actually existed already in the GUI but it was not used), and the "*Établir region"* button which allows to select a desired region of interest (ROI) as shows the picture above.

When detection starts a new signal appears. This signal represents the detection status. When there is no motion is set to "OK". If motion occurs is set to "STOP!". The value in brackets is the displacement in pixels detected when the stopping signal is launched or the "no motion" signal is launched again (after the status being "STOP!").

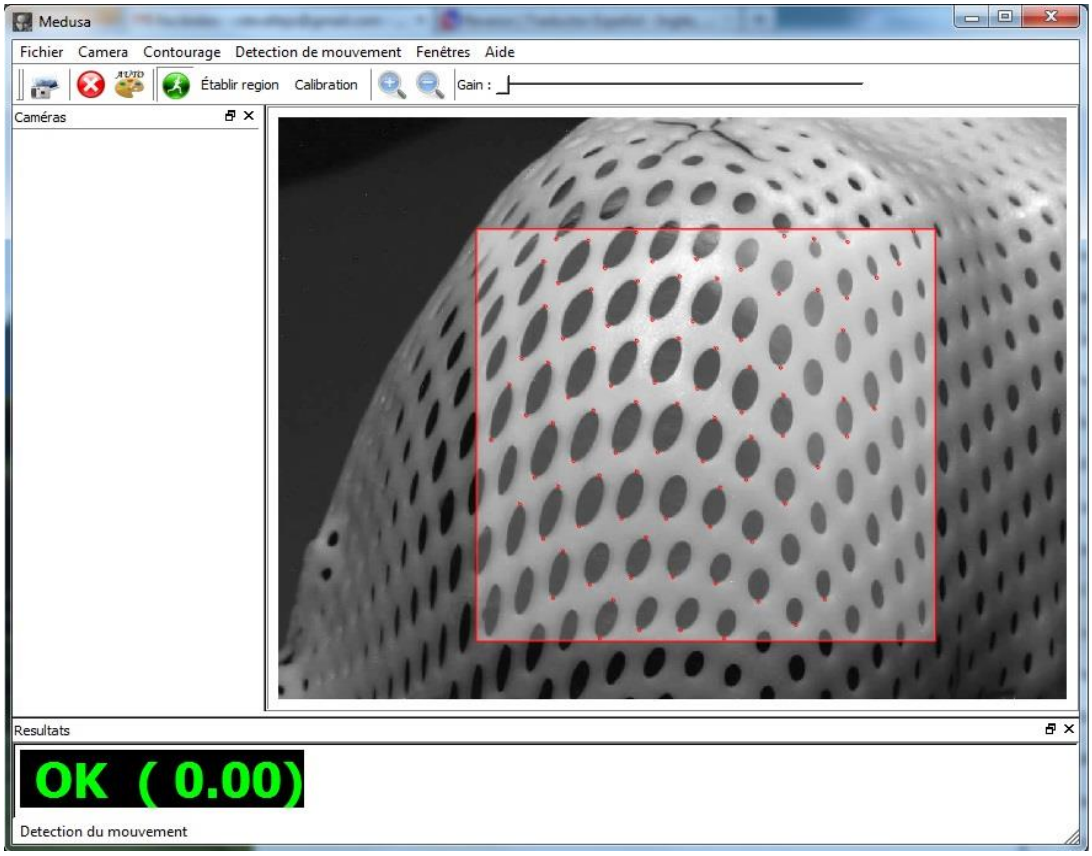


Figure 3.6. Motion detection (LK) enabled. Status: OK

Some new features are added to the configuration window. Advanced configuration option (*Configuration avancee*) is enabled and checked by default. These advanced settings allow the user to configure some motion detection parameters.

Four check boxes regulate the displaying options:

- *Montrer dessin*: general display function. If enabled, the motion information is drawn.
- *Montrer direction*: to show the motion vectors i.e. the flow field.
- *Montrer region*: displays the ROI.
- *Montrer ombre*: used for displaying a shadow of the reference frame (where there is no motion).

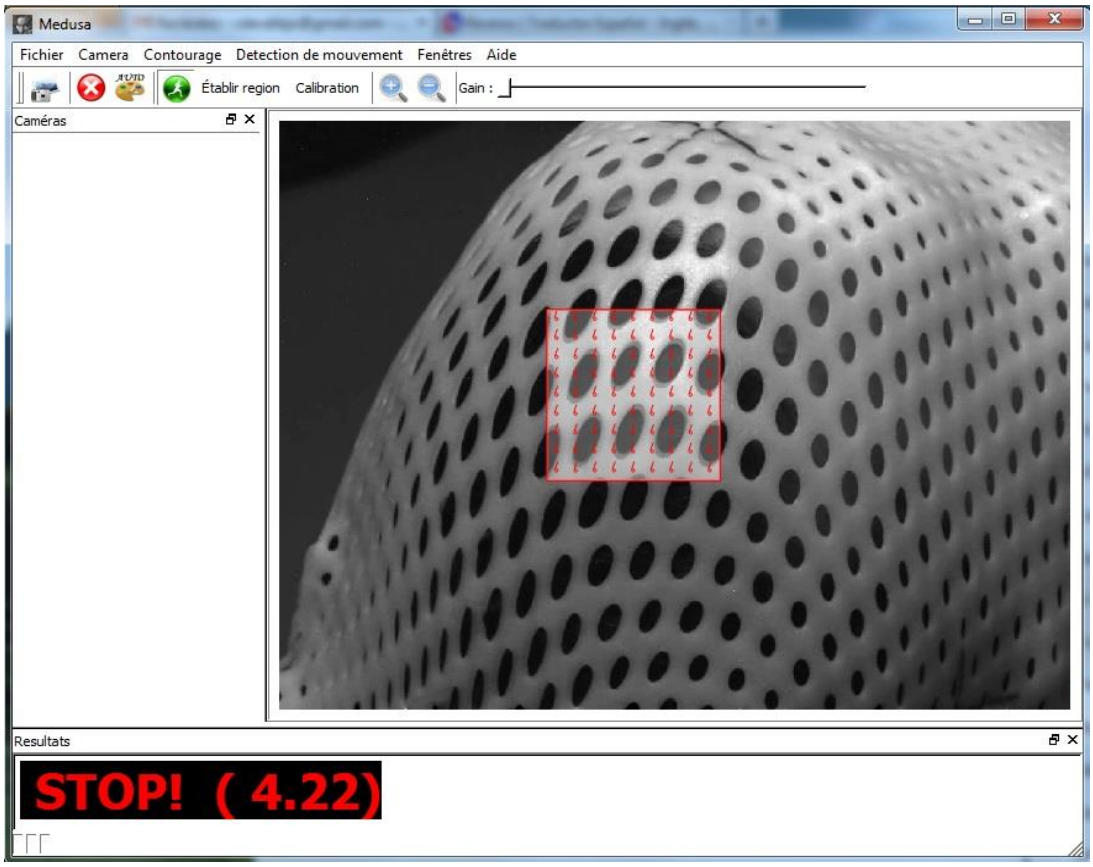


Figure 3.7. Motion detection (FB) enabled. Status: STOP

A table with three different tabs is available, one for each motion detection method (shown below).

For LK section, four parameters can be modified:

- *Nbre de points*: maximum number of points that LK will find for tracking.
- *Distance min*: minimum distance between the points.
- *Detection TH*: threshold (in pixels) for launching the stopping signal.
- *Couleur*: color of the displayed motion detection elements.

Two parameters can be changed for FB:

- *Detection TH*: threshold (in pixels) for launching the stopping signal.
- *Couleur*: color of the displayed motion detection elements.

MHI has two values which can be chosen:

- *Detection Criteria*: threshold for launching the stopping signal (based on the difference between gradients).
- *Couleur*: color of the displayed motion detection elements.

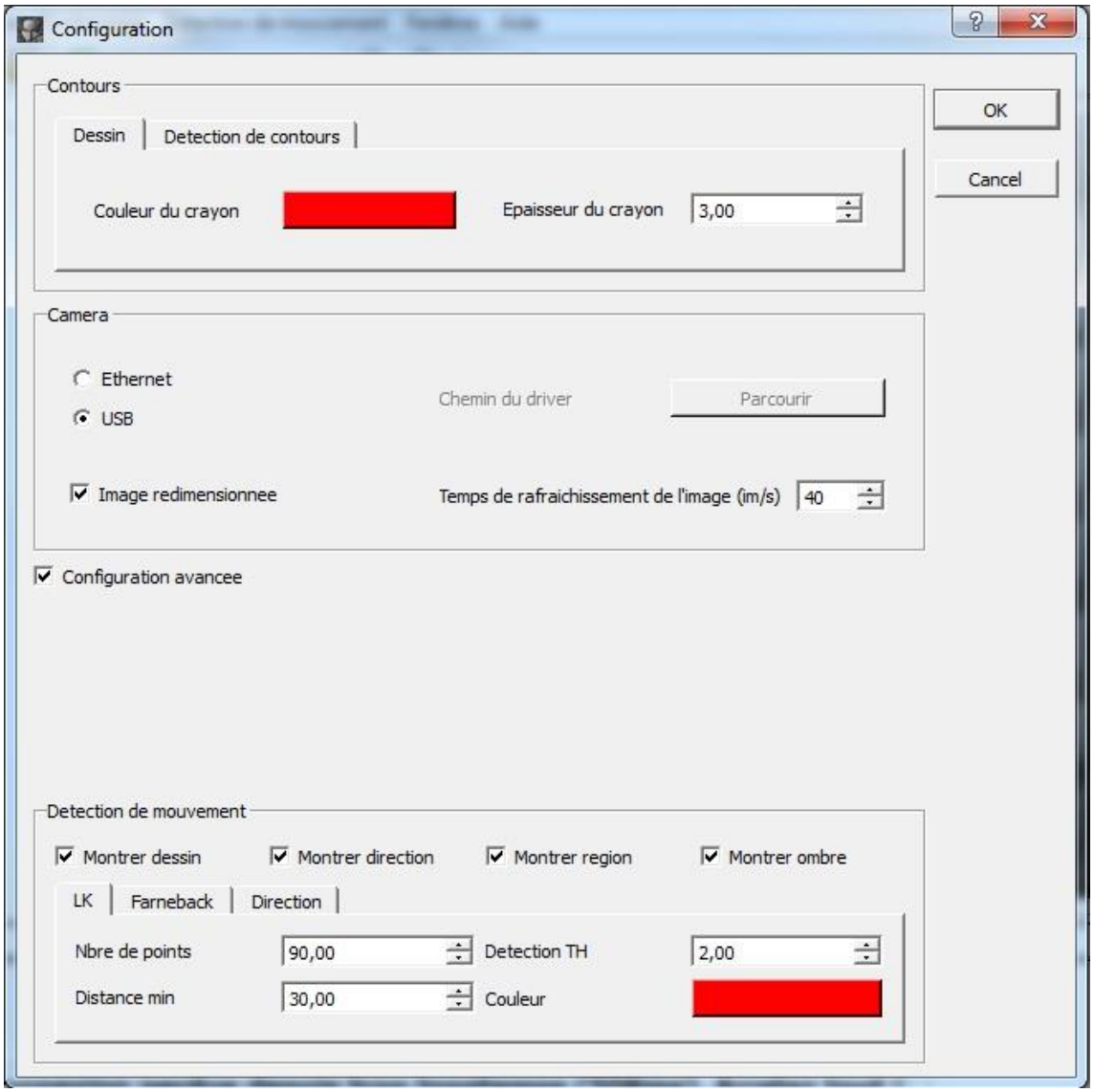


Figure 3.8. Medusa configuration window

Threshold's structure

In first tests two different thresholds were used, one for x-direction and another for y-direction. Both based on the average displacement of the flow field (in "x" and "y"). Later, the measure was improved and a single threshold, which involves both directions, is used. The procedure is based on the computation of the module of each velocity vector and the later calculation of the median value. This value allows discriminating distances (vectors) which can stand out of normal displacement boundaries (since they will be at the extremes of the sorted vector from which the median value is "extracted") and represent the actual motion properly without "intruder" values.

CHAPTER CONCLUSIONS

After all these new features implemented, the main integration in *Medusa* is already done. The user can now use a motion detection system where three different methods are available, choosing and manually defining the region where the motion have to be detected.

Displaying options are also defined, allowing the user to choose the most convenient way for working, as well as the possibility of modifying some detection parameters.

An *OpenCV* conversion is needed when computing the optical flow methods. *Medusa* is developed with a previous *OpenCV* version and LK and FB use new *OpenCV* C++ functions. The main difference is the class structure where images are defined and stored. The previous image object is called *iplImage* and the new one is called *Mat*. They have a different structure, so conversion from *iplImage* to *Mat* is needed. The step is simple but necessary. It is not the same case for MHI which is implemented with previous *OpenCV* functions.

CHAPTER 5

EXTRA FUNCTIONALITIES

Apart from the main topic (motion detection), some other extra features are implemented. These new options are to load a video file for evaluating and testing *Medusa* and to calibrate the system in order to have motion detection distances expressed in millimetres.

Loading a video file

The fact that *Medusa* needs to have a camera plugged in order to work adds some difficulties when testing. We would need to be inside the treatment room every time that we test the application. That is why a file loading function has been created.

This new functionality permits to choose a video file. Hence, being inside the treatment room will not be necessary anymore. Videos from different treatments can be recorded and loaded later at any time for evaluating the software. The disadvantage of this action is that compression is needed before loading a video.

Note: Usually it is a strong compression and the compression methods from the camera software are not good enough. It will be necessary to use an extra compression tool.

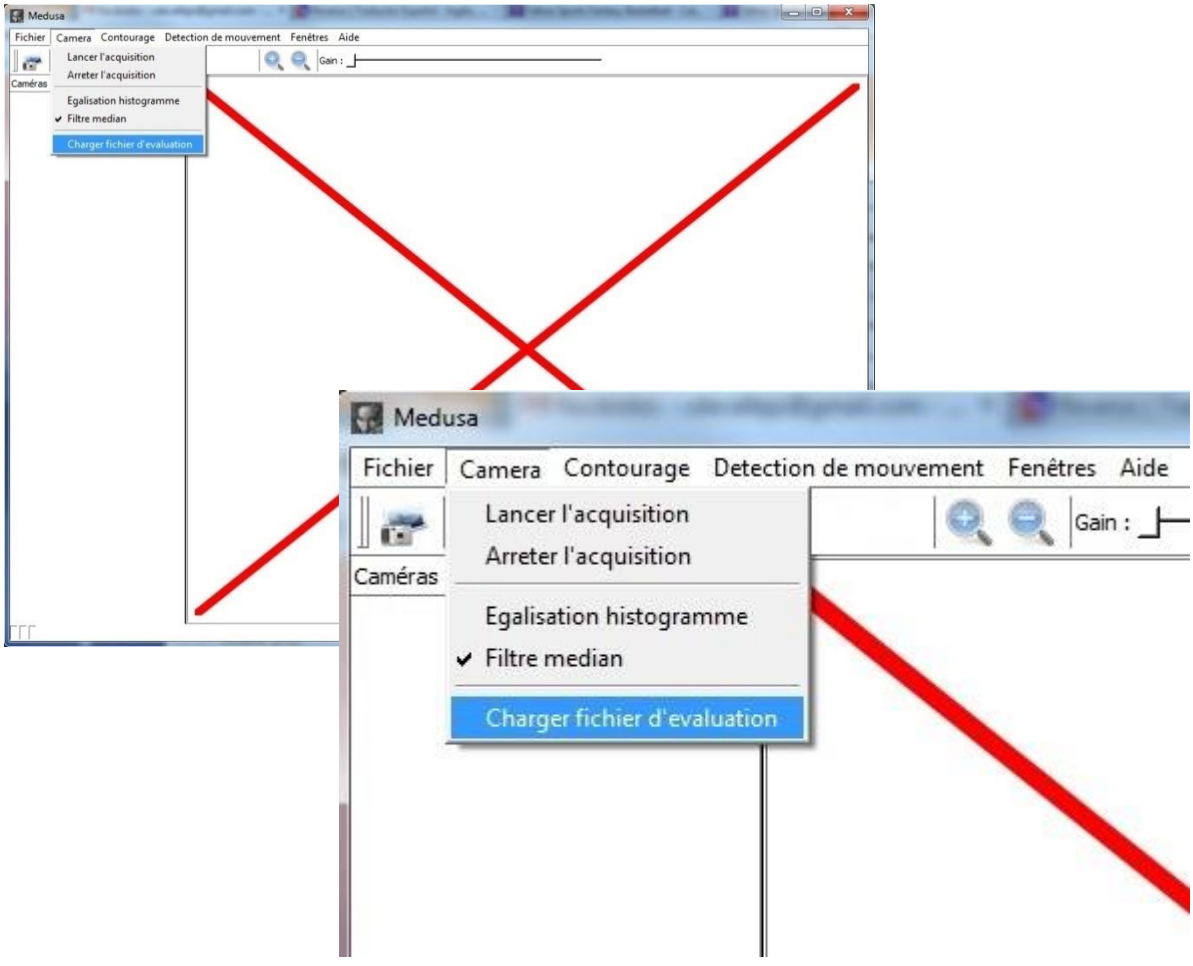


Figure 4.1. Medusa file loading option

For the graphical interface implementation a new option is created as can be seen in the image above. This menu option opens a window where a video file can be chosen.

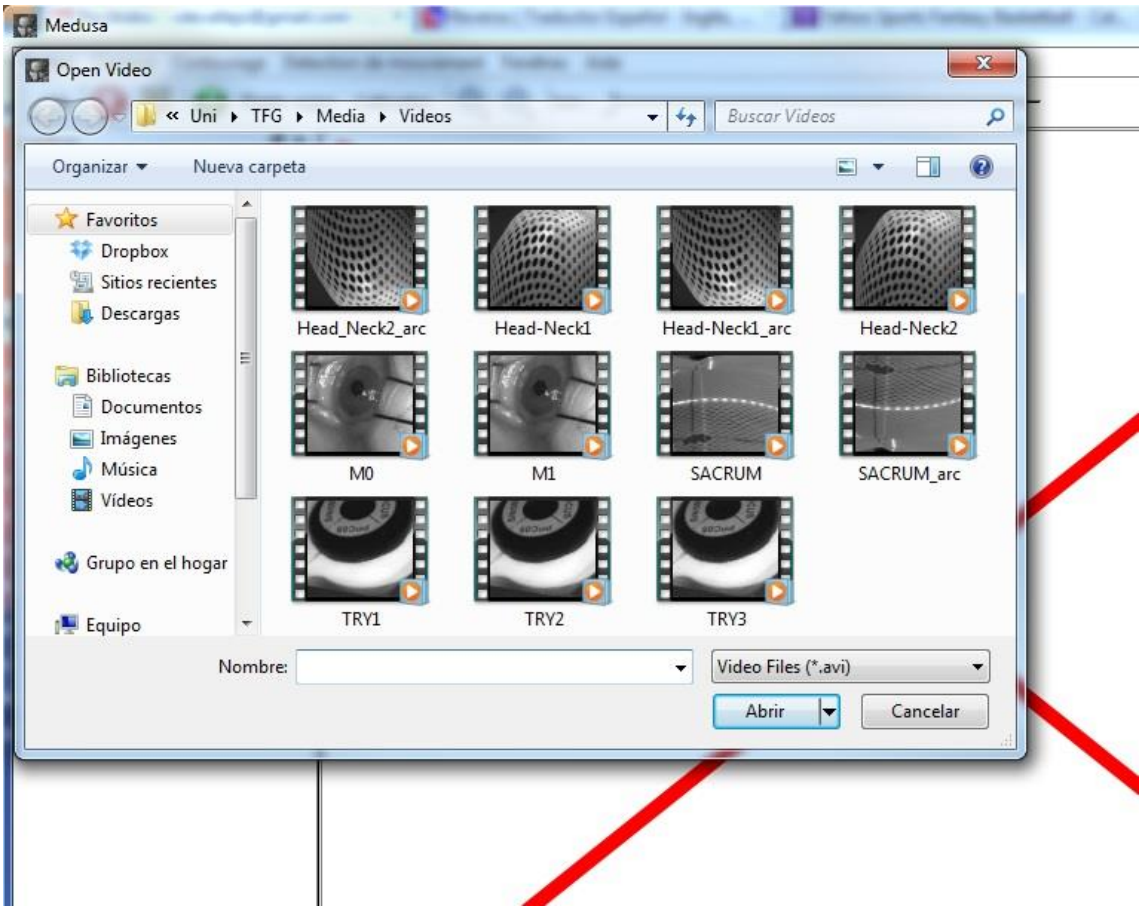


Figure 4.2. Medusa open video window

System calibration

To have a motion detection distance expressed in millimetres is more logical and useful than to have it in pixels. For that reason a calibration method for converting pixels to millimetres has been developed.

Since we do not have a 3D model and we work with 2D representations we cannot immediately to know how many millimetres correspond to one pixel in every different treatment. The creation of a 3D model would not fit in the project work and time plans, so another solution is presented.

With a measuring physical model and during a session treatment, before the irradiation, a real distance will be measured. In case of H&N treatment, for instance, a good measure would be the diameter of the holes of the mask. After getting that value, a line would be defined in *Medusa* starting from one extreme of the hole to the other. Then, the user would indicate the real distance in millimetres which corresponds to the one defined in pixels.

In order to do this, a new option is created in the actions bar. The new functionality is named "*Calibration*".

The procedure works as follows:

With new "*Calibration"* button enabled we define a line following the same procedure as when drawing manual contours.

We press the mouse left button, then we define the line and finally we release the button.

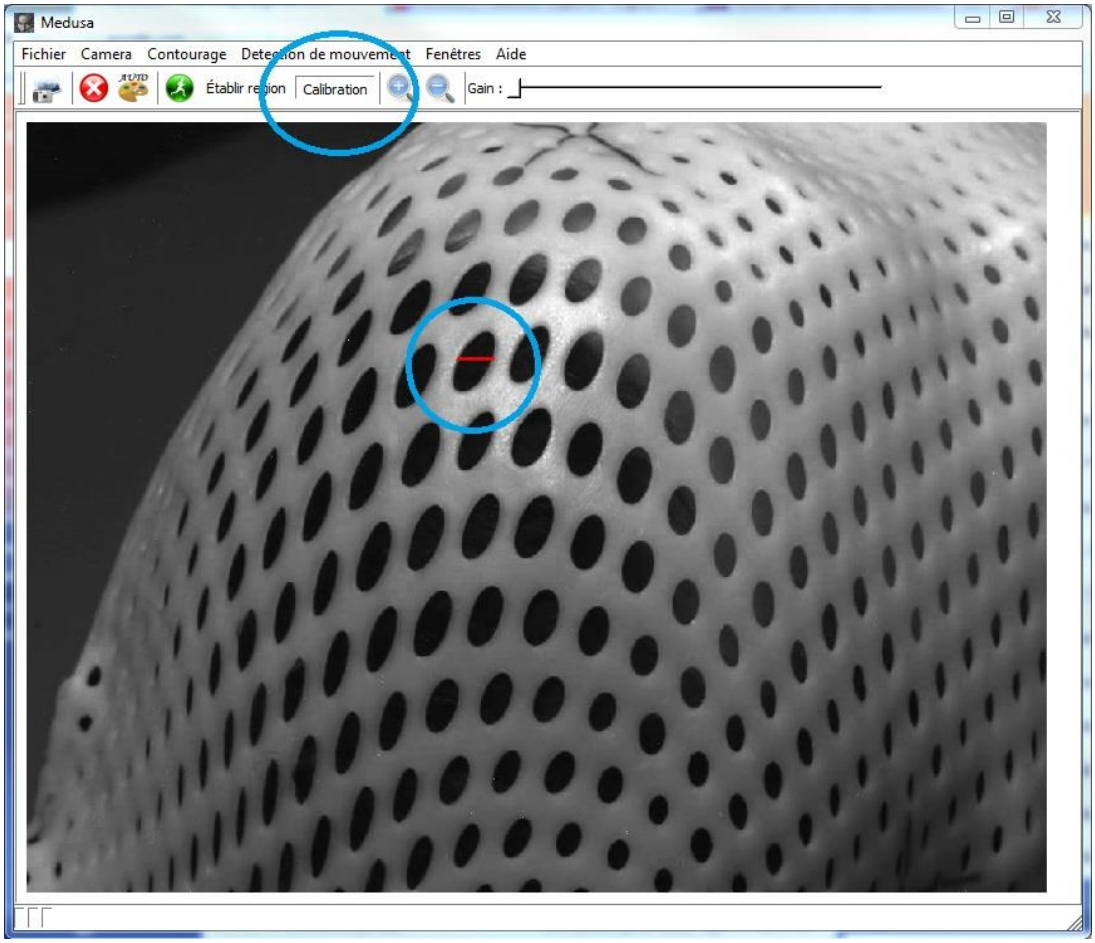


Figure 4.3. Calibration mode

When left button is released a pop-up appears.

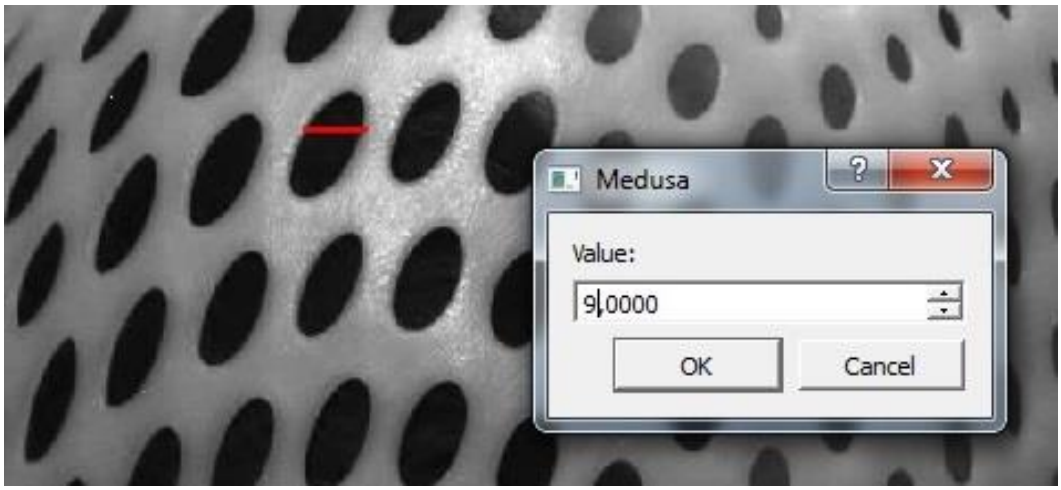


Figure 4.4. Calibration pop-up

A value in millimetres must be defined. After clicking “OK”, the system calibration is done.

Now, during the motion detection process the value next to the status signal will be expressed in millimetres.

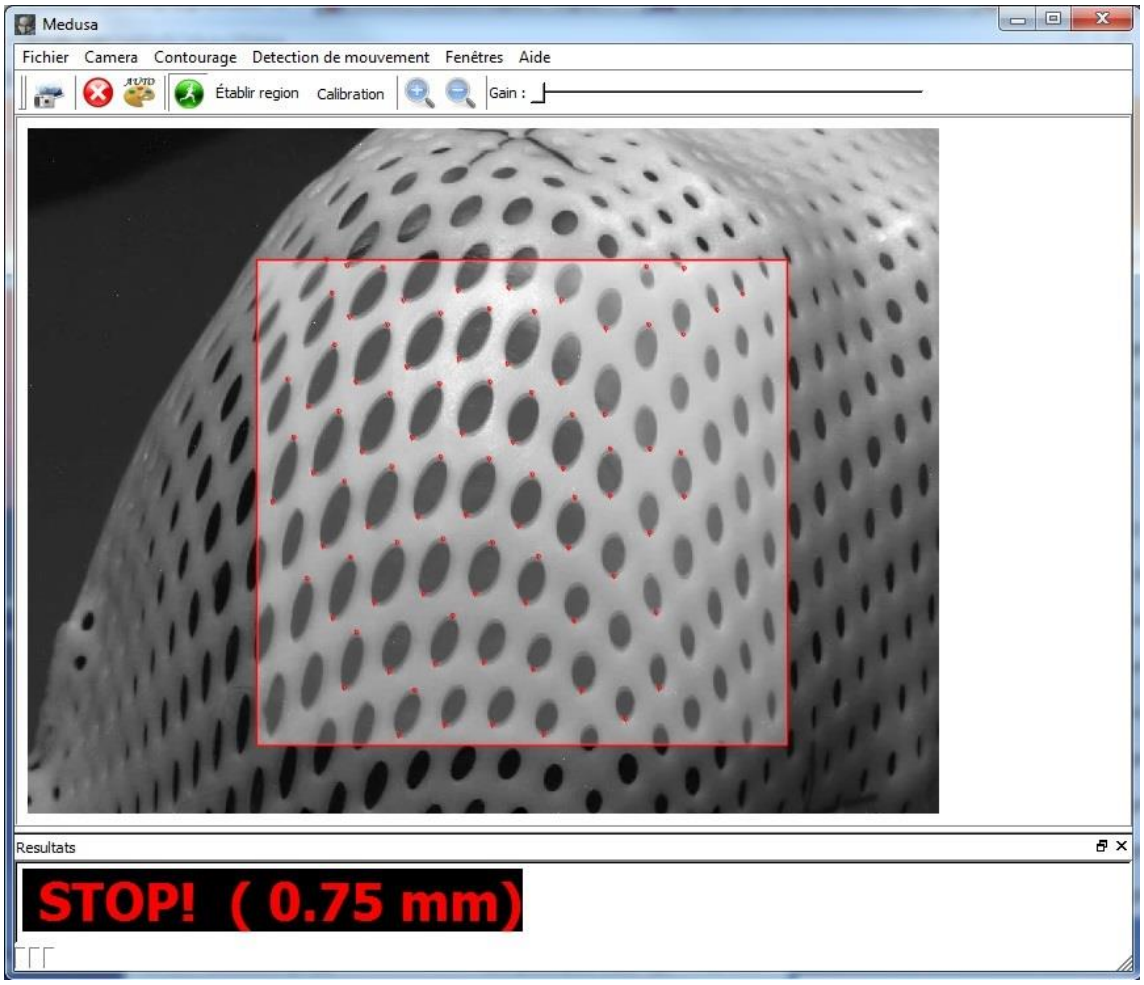
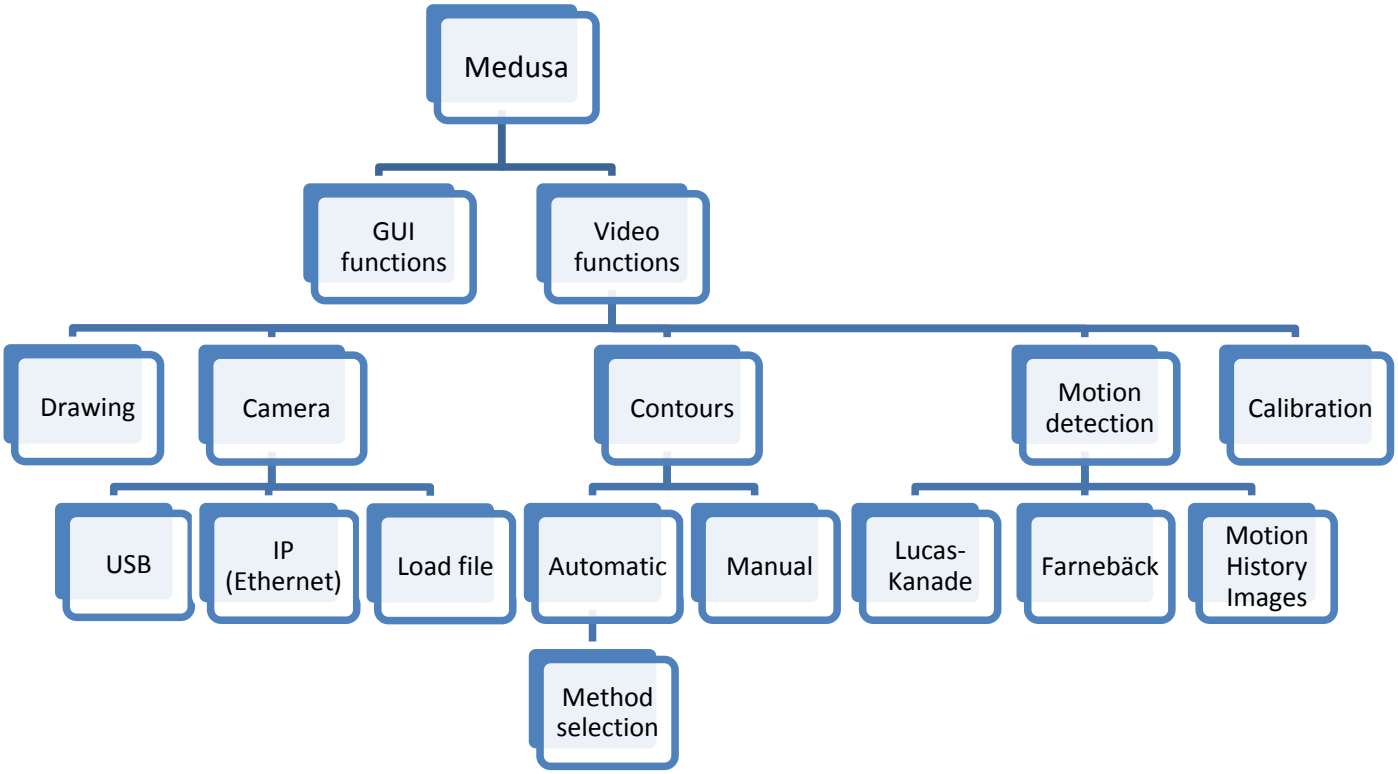


Figure 4.5. Lucas-Kanade method with calibration done previously

After that, a new structure scheme of *Medusa* can be defined as described in next page.



CHAPTER CONCLUSIONS

After the main integration other functionalities were considered to be useful. These are those which have been explained in this chapter.

The fact of being able to test any question of the software wherever you are gives more liberty to the developer, who will now be able to work anywhere else.

The calibration feature is a first way to get a pixels-to-millimetres conversion. However, if the calibration is done accurately it can display actual distances and great results as a first solution.

For doing the real measure properly it would be useful to create a physical pattern with different measuring scales in order to put it in front of the camera during the calibration step and being able to do the correct measures and establish the proper values.

At future improvements it would be interesting to create a 3D model, at least for the H&N case since we usually use two cameras for controlling the treatment, or also for the other cases if another camera is added. Other techniques could be implemented too, following its correspondent study and evaluation.

CHAPTER 6

EXTRA INTERPRETATION OF TESTS AND RESULTS

Practical and numerical results have been presented in chapter three, but there are other later aspects that can be commented and analysed too. These are other software considerations, warnings and extra problems which arise when developing the different tasks.

There are some problems or warnings that have been detected while implementing new features. Some of them have been improved and solved and others do not represent a real problem, but we need to take them into account. They are exposed in the following lines.

One of these obstacles was in the ROI. First of all, the region of interest was fixed at centre of the image. Since that it is not always the most interesting region a dynamic ROI was created in order to choose the best region of interest. The region dimensions were also fixed, so this was modified for establishing proper dimensions depending on the treatment. However, ROI dimensions are limited in order to not to harm too much the running time. Furthermore, the currently ROI can be defined in all the directions (from left-up to down-right, left-down to up-right, etc.)

Working with various threads requires using a mutex for not getting in trouble when changing values of variables. This fact represents a loss of time, so we need to be careful on how many mutex we use in order to avoid a conflict.

The motion detection displaying options in the GUI's configuration window do not work them all in MHI mode. *Montrer direction* option does not modify anything and the threshold criterion does not represent pixels but a difference between the silhouettes of the gradients. It is needed and important to say that motion detection is focused on the optical flow methods so improvements and optimization is done thinking in them. MHI is an additional method which can be further improved at future.

When it is not possible to get a camera and plug it in, two aspects must be considered. The "*TypeCamera*" field in the initialisation (.ini) file must be set to "*TypeCamera=1*" (USB camera) instead of "*TypeCamera =2*" (IP camera), otherwise *Medusa* will not run well when trying to load the camera drivers (which are available only when the camera is plugged in). The second step is to ignore the "runCamera()" instruction when initialising the application and loading all needed modules since no camera will be detected.

It is good to know that a laptop's webcam is not recognized, so an external camera is needed, and, if possible, Ethernet is better than USB. This problem could be checked and solved at future.

Some of the optical flow parameters can be chosen in the ".ini" file. Those are the followings:

- DETECTION\DisplayDetection (bool): Default option for displaying the motion information.

- DETECTION\DisplayFlow (bool): Default option for displaying the motion vectors.
- DETECTION\DisplayROI (bool): Default option for displaying the ROI.
- DETECTION\DisplayShadow (bool): Default option for displaying the reference frame shadow.
- DETECTION\NeedToInit (bool): Default option for displaying the LK's initialisation.
- DETECTION\ConfigurationAvancee (bool): Default option for enabling the advanced configuration in the configuration window.
- DETECTION\NframesDetection (int): Number of frames read for computing motion detection (computed every "X" frames).
- DETECTION\LKMaxPoints (int): Maximum number of LK points found.
- DETECTION\DistanceBetweenLKPoints (int): Minimum distance between the LK points found.
- DETECTION\LK_Color (QColor): LK color.
- DETECTION\FB_Color (QColor): FB color.
- DETECTION\MHI_Color (QColor): MHI color.
- DETECTION\LK_Threshold (double): LK detection threshold.
- DETECTION\FB_Threshold (double): FB detection threshold.
- DETECTION\MHI_Threshold (double): MHI detection threshold.

Motion detection algorithms behave quite fast by itself (without *Medusa*). Its running time within *Medusa* increases due to the multithreading and different mutex's and this affects the frame rate. Actual frame rates have not been carefully checked, due to the lack of time once more. However, it is not supposed to represent a real problem at all.

In terms of detection accuracy the system is capable to detect up to thirty pixels of difference or even forty in some cases. This range is more than enough since the objective is to detect a small movement.

CHAPTER CONCLUSIONS

During a planned task you notice some aspects and problems that you did not count on them before. This chapter summarizes all those questions explaining the actions done for solving them.

It is good also for the user to know about that kind of stuff in order to realize how the software works.

CHAPTER 7

THESIS CONCLUSIONS

At the end of this project, according to the established scopes and having analysed the results, the next conclusions are obtained:

- New needs have been identified in order to improve the control of proton therapy treatments as well as the tasks of the therapists.
- A new motion detection system has been developed, integrated within *Medusa* and tested. The system is capable to detect a very slight motion and notify the user in real time.
- Lucas-Kanade, Farnebäck and Motion History Images are the selected algorithms since they are able to work in real time with strong security and precision.
- Dual TV-L1 and SimpleFlow methods were discarded because of not working properly in real time.
- CamShift, Kalman filter and GMM were considered secondary and have not been finally implemented due to the lack of time.
- A video loading option has been developed. Now it is not necessary to always have the camera plugged in and being at the treatment room for working with real images, so it is easier for the developer to do tests and evaluations.
- A visual displayed signal notifies the motion status to the user. A calibration option has been done for converting pixels to millimetres in order to tell more useful information to the therapists.

Concluding, we can say that the main objectives have been reached and even extra ones have been developed with good visible results and an active future ahead.

PERSONAL OPINION

The realisation of this thesis has represented a great experience in all ways, from learning lots of new concepts to meeting new people, cities and cultures. I have improved and learnt a lot about image and video processing as well as C++, *OpenCV* and *Qt* programming. I have become used to work in a professional ambience and to be capable to drive some problems to their solutions.

From my point of view, the results are quite nice and the improvements and innovations will be useful for improving the treatments and getting easier the job of the therapists. As an appointment, I think that chapter seven was necessary for answering and explaining possible future doubts and questions which can firstly be non-important. It would have been great to have time for implementing more methods and features and better testing all of them, but the calendar have not allowed us to do so. Nevertheless, I am so happy and satisfied of the results obtained and work done.

I have been motivated since the first moment that I knew what I had to do. Cancer treatment is a very interesting and important field which needs people looking for improvements every day and I specially wanted to do my bit. I have learnt a lot about radiotherapy and proton therapy treatments and I am so happy for it and for having helped in this research.

Finally, I entrust the day in which cancer is not an important and critical illness will arrive.

FUTURE WORKING

Some improvement and evolution tasks that could be done at future are:

- Integration of *Medusa* within *iMagX* platform.
- To find better parameters (if possible) for LK and FB (to do more tests).
- To improve MHI method. To optimise the algorithm, create the best displaying functions and establish a relation between pixels and millimetres.
- Studying the possibility of adding some new motion detection methods. Since all of them represent a subclass, they can be easily integrated.
- Mixing some methods could be considered if there is a visible improvement in the results.
- The running time as well as the frame rate could be carefully studied in order to get a faster functioning.
- The complete integration of *Medusa* within *iMagX* following all the platform features and rules.
- The optimization of the whole source code in order to improve the general behaviour, including also the memory management.
- To build a 3D model for the calibration and pixels to millimetres conversion.

INDEXES

REFERENCES

[1]. "*An Iterative Image Registration Technique with an Application to Stereo Vision", Bruce D. Lucas, Takeo Kanade. Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1981.*

[2]. *"A Duality Based Approach for Realtime TV-L1 Optical Flow", C. Zach, T. Pock and H. Bischof*, and in "*TV-L1 Optical Flow Estimation", Javier Sánchez, Enric Meinhardt-Llopis, Gabriele Facciolo.*

[3]. *"SimpleFlow: A Non-iterative, Sublinear Optical Flow Algorithm" Michael Tao, Jiamin Bai, Pushmeet Kohli, and Sylvain Paris*.

[4]. "*Two-Frame Motion Estimation Based on Polynomial Expansion*", *Gunnar Farnebäck, Computer Vision Laboratory, Linköping University, SE-581 83 Linköping, Sweden.*

[5]. *Davis, J.W. and Bradski, G.R. "Motion Segmentation and Pose Recognition with Motion History Gradients", 2000.*

[6]. *Bradski, G.R. "Computer Vision Face Tracking for Use in a Perceptual User Interface", Intel, 1998.*

[7]. "Conception du logiciel d'imagerie numérique de surveillance du patient". Mohamed Amine Bouskia. (Report from *CPO*).

Consulted sites:

- C++ tutorials: http://www.cprogramming.com/tutorial/c++-tutorial.html
- *OpenCV* documentation: http://docs.opencv.org
- *OpenCV* forums: http://answers.opencv.org
- *Qt* documentation: http://qt-project.org/doc
- *Qt* forums: http://qt-project.org/forums
- Stackoverflow forums: http://stackoverflow.com
- *iMagX* internal documentation.

Books:

- C++ GUI Programming with Qt 4. - Prentice Hall.
- Learning *OpenCV*. Computer Vision in C++ with the *OpenCV* Library. - Adrian Kaehler and Gary Bradski. O'Reilly Media.
- Mastering *OpenCV* with Practical Computer Vision Projects. - Daniel Lelis Baggio

OpenCV and *Qt* versions:

- OpenCV 2.4.6
- Qt 4.8.5

INDEX OF FIGURES

INDEX OF TABLES

NOMENCALTURES

- CPO: Centre de Protonthérapie d'Orsay.
- UCL: Université Catholique de Louvain.
- LK: Lucas-Kanade algorithm.
- SF: SimpleFlow algorithm.
- FB: Gunnar Farnebäck algorithm.
- MHI: Motion History Image(s).
- ph: horizontal displacement in pixels.
- pv: vertical displacement in pixels.
- pd: diagonal displacement in pixels.
- H&N: Head and Neck treatment.