# DEVELOPPEMENT D'UN ALGORITHME POUR LE TRAITEMENT D'IMAGES MEDICALES SUR UN SYSTEME EMBARQUE

## Universidade Federal do Paraná

## Curitiba - Brésil

## Du 27 février 2012 au 04 septembre 2012


| | |
|---|---|
| Rémi Megret | Tuteur de stage |
| André Mariano | Maître de stage |
| Michelle Galassi | Département électronique - TSI |

# TABLE DES MATIERES

# TABLE DES ILLUSTRATIONS

# Remerciements

Mes remerciements s'adressent tout particulièrement à André Mariano, mon Maître de stage, pour m'avoir accueillie dans son département ainsi que pour son soutien et son encadrement tout au long du stage.

Je tiens également à remercier ma famille qui m'a permis de faire mon Projet de Fin d'Etudes à l'UFPR.

# PARTIE N°1. INTRODUCTION

La biométrie désigne, de façon générale, l'étude des êtres vivants sur le plan quantitatif. Le terme de biométrie se rapporte de plus en plus à l'usage des techniques permettant de reconnaître et d'identifier un individu. [1]

L'utilisation de l'iris en tant que caractéristique d'identification biométrique est intéressante, car, comme pour le visage, l'image de l'iris peut être capturée sans contact avec l'individu, à condition toutefois qu'il soit assez près de l'appareil de capture. Il faut savoir que l'œil n'est pas seulement utile à la reconnaissance d'individus. En effet, il peut aussi être utilisé à des fins médicales. Dans notre cas, l'œil, et plus particulièrement la pupille, sert au dépistage de maladies.

La contraction et la dilation de l'iris sont contrôlées par deux muscles antagonistes : le muscle sphincter pupillaire et le muscle dilateur de la pupille. Ainsi, quand on parle de « pupille dilatée », il s'agit en fait d'une dilatation de l'iris due à un relâchement du muscle sphincter et une contraction du muscle dilatateur. Réciproquement, la contraction de la pupille est causée par une contraction du muscle sphincter.

La contraction ou la dilatation de l'iris est un réflexe physiologique pour adapter la vision à la luminosité ambiante. Quand la luminosité ambiante est forte, l'iris se contracte, ce qui diminue l'intensité lumineuse qui vient frapper le centre de la rétine, et vice-versa. [2]

Les modifications normales (émotion forte par exemple) ou pathologiques (prise de drogues par exemple) de l'état physiologique de l'organisme modifient aussi le diamètre pupillaire.

La pupillométrie, mesure du diamètre pupillaire, trouve des utilisations en recherche médicale et dans le domaine de la publicité. [3]

Le choix du Projet de Fin d'Etudes dans l'un des laboratoires de l'UFPR s'est fait dans le but d'étudier la pupille et son comportement. La biométrie est un domaine fascinant et en plein essor. De plus, ce projet allie l'électronique à la médecine et a pour but d'améliorer le quotidien des patients en faisant des dépistages plus simples et plus rapides.

L'objectif de ce stage est de mettre en œuvre une méthode de reconnaissance de la pupille pour la mesurer et d'utiliser ces mesures à des fins médicales (lors du dépistage de maladies par exemple).

Le système final stimule l'œil du patient avec un flash de lumière. Les réactions de la pupille sont enregistrées sous forme de vidéo et sont analysées. L'analyse développée dans ce rapport est celle de la mesure de la taille de la pupille en fonction du temps. Les résultats pourront être utilisés et analysés par des personnes travaillant dans le domaine médical afin de dépister des maladies telles que la maladie d'Alzheimer, le diabète ou encore de faire un test d'alcoolémie.

Le projet sera d'abord décrit de façon générale, donnant ainsi une idée générale de son fonctionnement. Par la suite les parties de traitement d'image et du signal seront développées avec plus de détails. Et pour finir, les résultats obtenus par simulation seront observés et expliqués.

Vous trouverez dans la partie suivante un résumé du rapport en anglais écrit pour l'UFPR.

# PARTIE N°2. RESUME DU RAPPORT EN ANGLAIS

## 2.1 Introduction and objectives

Biometrics is the study of living in terms of quantity. The use of the iris as a biometric characteristic is interesting because the iris image can be captured without any contact with the individual. The eye can also be used for medical purposes. In our case, it will be used to detect diseases.

The contraction and dilatation of the iris are controlled by two antagonistic muscles: the pupillary sphincter muscle and the dilator muscle of the pupil. The contraction or dilatation of the iris is a physiological reflex to adapt the vision to the ambient brightness. When the ambient light is strong, the iris contracts, which reduces the intensity of light which strikes the centre of the retina, and vice versa.

Normal or pathological changes of the physiological state of the body also modify the pupil diameter.

The objective of this project is to implement a recognition method of the pupil in order to measure it and to use these measures for medical purposes.

The final system stimulates the patient's eye with a flash of light. The pupil reactions are recorded on video and analyzed. The analysis in this report is the measurement of the size of the pupil as a function of time. The results will be used and analyzed by people working in the medical field in order to detect diseases such as Alzheimer's disease, diabetes or even to do an alcohol test.

## 2.2 Equipment

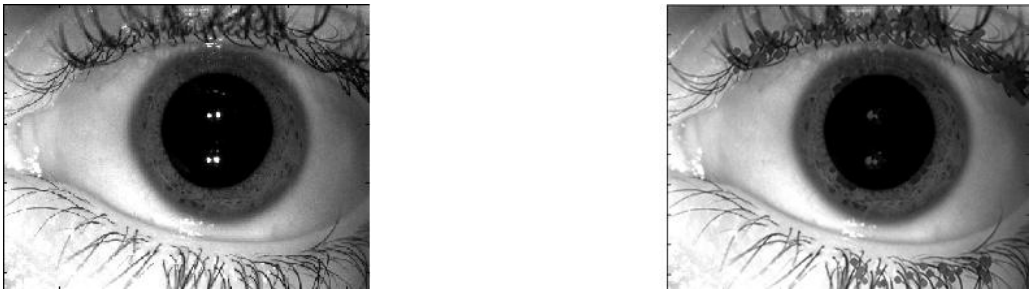
The following equipment is provided:

- A FPGA DE2-115 from Altera
- A camera TCM8230MD from Toshiba compatible with the FPGA and allowing to record video of the eye
- The software Quartus II allowing the analysis and synthesis of HDL designs
- The simulation software ModelSim which provides a full simulation and debugging environment for complex designs in ASIC and FPGA.
- The software MATLAB allowing the analysis of the obtained data

## 2.3 Methods

- The pupil stays in a dark room and after a while is stimulated by a flash of light. The reaction of the pupil is recorded with a camera and is analyzed with the following methods.

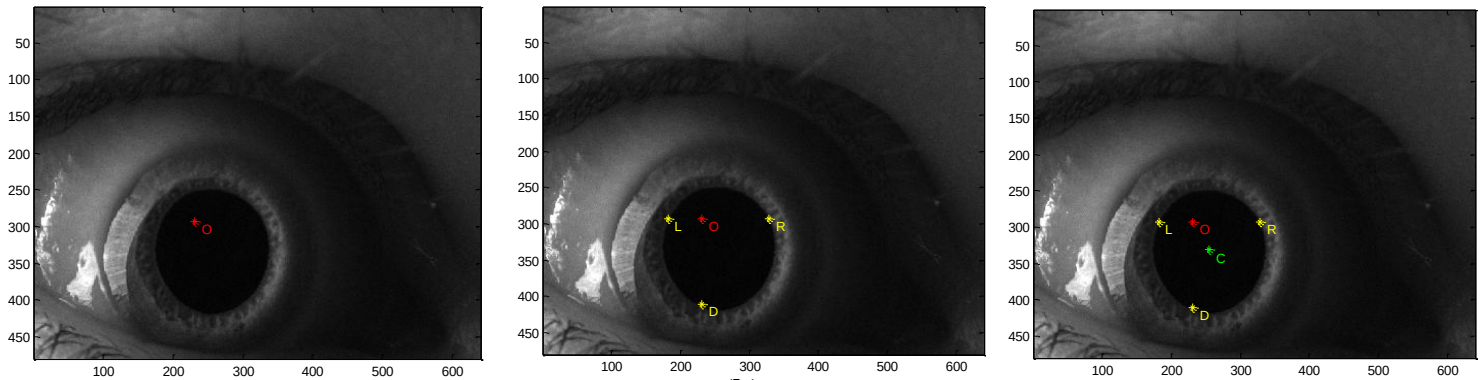
- Pre-processing: This step allows suppressing the light spots in the eye with the algorithm described in [1].



(a) (b)

**Figure 1 : (a) image with light spots and (b) image after the pre-processing**

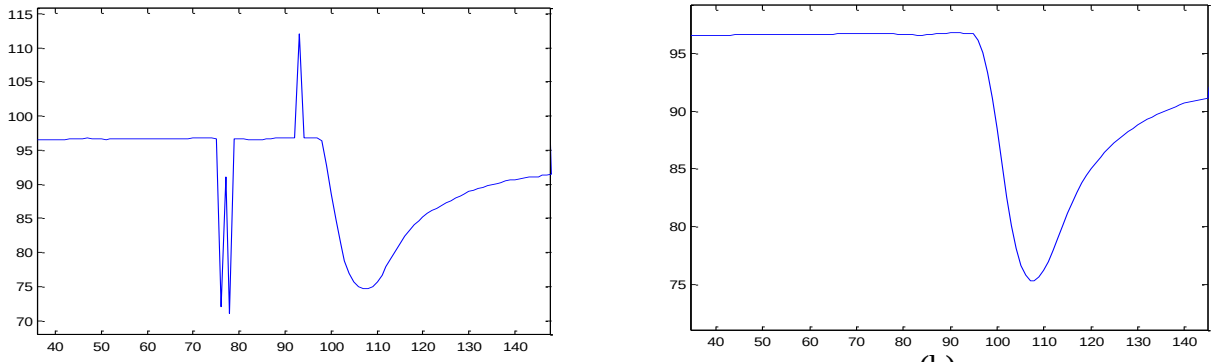
- Image processing algorithm (Algorithm of Weiqi Yuan, Lu Xu and Zhonghua Lin): this algorithm is used to find the centre of the pupil and the radius value, as explained in [2].
  - 1st step: Find a point O inside the pupil.
  - 2nd step: Starting from the O point, find 3 points on the left, on the right and down from this point, points belonging to the pupil circle.
  - 3rd step: Calculate the coordinates of the centre and the **radius value** of the pupil, as detailed in the first annex of the report.




**(a)** **(b)** **(c)**

**Figure 2 : (a) Point O inside the pupil, (b) L, R and D belonging to the pupil circle and (c) point C, centre of the pupil**

- Filtering: Once all the images were processed, all the radius values are filtered in order to suppress the pupil localisation errors. For more information, see [1].
  - Averaging filter: This filter takes care of the points for which the localisation failed. It gives a new value to these points using an average method.
  - Gaussian filter: This filter homogenizes results. The formula of the gaussian filter is $G(i) = \frac{1}{\sigma} * e^{-\frac{i^2}{\sigma^2}}$ and has the following parameters: resolution $\sigma = 5$ and window of 7 elements.



(a) (b)

**Figure 3 : (a) Radius with errors and (b) radius as a function of time after filtering**

## 2.4 Results

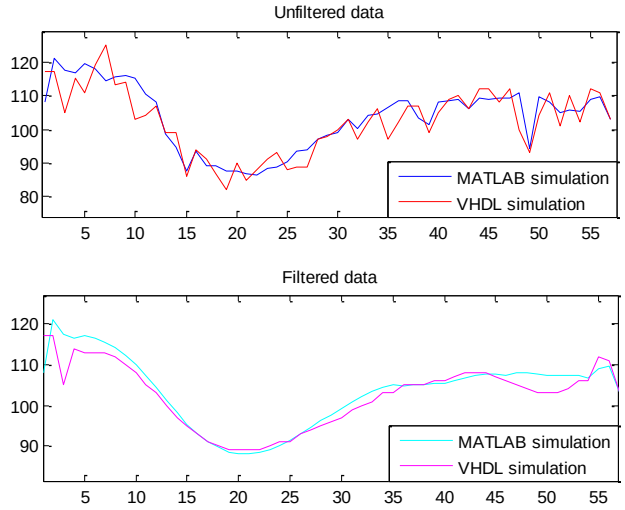



**Figure 4 : Comparison between the MATLAB and the VHDL values**

The first graphic represents the MATLAB and VHDL results obtained just after the image processing part. The shape of the curve shows the contraction of the pupil after the stimulation and its return to the dilated state.

The second graphic represents the results obtained after the filtering part. It is easy to see that the filtering has an important effect on the shape of the curves. The points far away from the curve disappeared thanks to the filtering.

The three first and last points of the curves are not filtered because of the algorithm used.

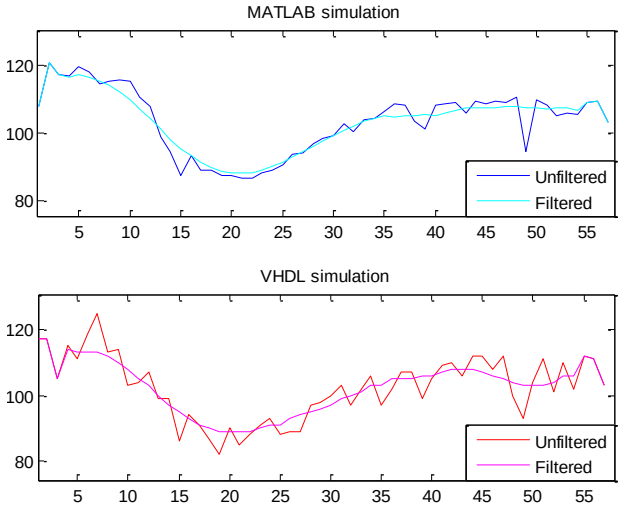



**Figure 5 : Comparison between the input and output values of the filtering part**

These graphics represent the comparison between the non filtered values and the filtered values from MATLAB and VHDL. They allow knowing exactly if the filtering algorithms chosen are efficient or not.

In both cases, the results are better after the filtering.

The VHDL results seem to be less precise. It is simply due to the input values which are integer values. At the end of the curve, the filtering did not work as good as the filtering on MATLAB results because the VHDL values are really different.

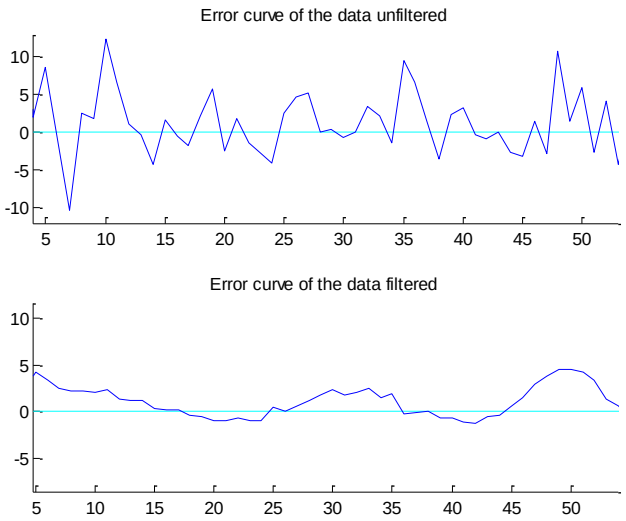



**Figure 6 : Errors from the FPGA**

These graphics represent the errors between the results MATLAB and VHDL. It is easy to see that the filtering part did a good work because the errors between the reference values (MATLAB) and the VHDL values decreased of half.

Another tool to measure the precision is the average error percentage which was calculated before and after the filtering part. Here is a table with the values:

| Average error percentage | Value |
|---|---|
| Before filtering | 0.0592 % |
| After filtering | 0.0201 % |

## 2.5 Conclusions

At the end of the project, the result observed is the reaction of the pupil to a stimulation of light. When there is the flash of light, the pupil contracts and then returns to its original state.

Some parameters of the algorithm can be changed in order to modify their efficiency and to adapt them to each system of pupilometry.

The results obtained with the simulation of the full project of image and signal processing allowed validating the project in simulation. The objectives of project simulation were reached.

The next step will be the test of the project with the FPGA. Then the project should be tested entirely from the video made by the camera until the curves displayed on MATLAB.

Once the project will work entirely, the curves will be used to find any anomaly with the pupil reaction.

## 2.6 References

[1] Vitor Yano, Alessandro Zimmer, Giselle Ferrari, “Utilização de pupilometria dinâmica para identificação pessoal baseada em reflexos humanos”, *Universidade Federal do Paraná, Departamento de Engenharia Elétrica*.

[2] Weiqi, Yuan, Lu. Xu, Zhonghua, Lin, “A novel iris localization algorithm based on the gray distributions of eye images”, *Computer Vision Group, Shenyang University of Technology*.

# PARTIE N°3. UNIVERSIDADE FEDERAL DO PARANÁ

## 3.1 L'université UFPR [4]

L'Université Fédérale du Paraná est la plus ancienne université du Brésil et est un symbole de Curitiba. Enveloppée d'une histoire de luttes et de conquêtes depuis 1912, l'UFPR est la référence dans l'enseignement supérieur pour l'Etat et pour le Brésil.

Symbole majeur de l'*inteligensia paranense,* l'université montre son importance et son excellence à travers les licences, spécialisations, masters et doctorats qui sont guidés par le principe d'indissociabilité entre l'enseignement, la recherche et l'activité d'extension. L'extension est une activité pratiquée par les universités fédérales du Brésil. Elèves et professeurs mettent leurs connaissances à profit de la société en développant des projets qui l'aideront. Un exemple de cette activité d'extension est le PET (programa educational tutorial) qui est une association dirigée par des étudiants recrutés sur concours et qui reçoivent une petite rémunération du gouvernement. Un des objectifs de cette association est d'apporter des compléments sur les enseignements suivis au département.

La fonction sociale de l'université est valorisée par le biais du trio enseignement, recherche et activité d'extension. De plus, ces trois actions doivent permettre à cette université, sous forme de connaissance, de technologie et de culture, d'avoir les ressources publiques qui lui permettent d'exister en tant qu'Institution Fédérale de l'Enseignement Supérieur.

Au-delà des campi à Curitiba, l'UFPR est présente dans la province paranense (du Paraná) et la côte de l'État fédéral, avec comme rôle actif de développeur socio-économique et de la qualité de vie des paranenses (habitants du Paraná) par le biais de l'accès à l'enseignement supérieur et des activités développées par la communauté académique en faveur de la société du Paraná et du Brésil.

## 3.2 Département d'ingénierie électrique (engenharia elétrica) [5]

Ce département qui regroupe 47 enseignants, dont 25 enseignants chercheurs, est en phase d'agrandissement avec notamment la construction d'un nouveau bâtiment permettant d'accueillir plus d'élève.

**Figure 7 : Bâtiment du département d'ingénierie électrique**

### 3.2.1 Histoire

Le Département d'Electricité (actuellement Département d'Ingénierie Electrique) de l'Université Fédérale du Paraná fut créé en 1966, conjointement avec la mise en place de la Licence en Ingénierie Electrique. Ce cours, pionnier dans l'état, est né d'une demande du gouvernement de l'état, menée principalement par la COPEL – Compagnie Paranaense de l'Energie, qui a commencé à l'époque un remarquable effort visant à l'électrification de l'état du Paraná. A cette époque, le Paraná était considéré comme l'un des états brésiliens les plus mal desservis en énergie électrique, qui servait seulement 20% des habitants. Une grande partie de l'installation électrique paranaense était composée de centrales diesel-électriques appartenant à des particuliers ou à des préfectures municipales, qui étaient en grande partie dans un état déplorable. Les coupes d'énergie électrique pouvaient durer des jours entiers et la fiabilité des services était en-dessous du tolérable. Malgré le fait qu'elle fut fondée en 1958, ce n'est que dans les années 60 que la COPEL a commencé à fonctionner de manière efficace, assumant les fonctions de la compagnie *Forza e Luz do Paraná*, de capital étranger, qui desservait la région métropolitaine de Curitiba.

En 1976, le Cours d'Ingénierie Electrique de l'UFPR a commencé à offrir, en plus de l'électrotechnique, une spécialité en télécommunications. Une fois de plus, cette décision fut prise en accord avec l'état économique de l'état, motivée par l'expansion du système téléphonique du Paraná. La TELEPAR (ancien nom de la compagnie de télécommunications Oi) a soutenu la création de cette spécialité dans le cours de l'UFPR, avec pour but d'obtenir des ingénieurs spécialisés afin de promouvoir l'expansion de ses services. Le Cours d'Ingénierie Electrique de l'UFPR fut le premier de la région Sud à offrir une formation dans le domaine de l'Ingénierie des Télécommunications et fut responsable de la formation de la majorité des Ingénieurs en Télécommunications de la TELEPAR. Après 1996, avec l'ouverture du marché des télécommunications du Brésil à l'initiative privée, d'autres entreprises de prestation de services apparurent dans ce domaine, qui ont également été se fournir en personnel provenant du Cours d'Ingénierie Electrique de l'UFPR.

En 1982, le cours d'Ingénierie Electrique de l'UFPR a ajouté la spécialité en Electronique à ses options. Poursuivant son duo avec le panorama socio-économique local, cette décision fut basée sur la demande croissante d'ingénieurs spécialisés pour les industries qui commencèrent à s'installer dans la zone industrielle de Curitiba. La multinationale Siemens, avec ses partenaires brésiliens de l'époque, a établi la société Equitel à Curitiba, usine d'équipement de télécommunications, suivie par la société japonaise Furukawa avec une usine de câbles téléphoniques. Conjointement avec d'autres entreprises nationales et étrangères, le scénario local dessine un avenir d'industrialisation rapide pour le Paraná, qui a nécessité la formation de professionnels en électricité avec un profil

différent de celui jusqu'alors formé. Le cours d'Ingénierie Electrique de l'UFPR a réorganisé son programme d'études à cette occasion et est devenu l'un des principaux fournisseurs de main-d'œuvre qualifiée pour les entreprises qui s'installèrent dans le Paraná. En quelques années, le profil socio-économique du Paraná a présenté un changement vers une base industrielle dans laquelle les ingénieurs formés par l'UFPR ont agi dans plusieurs domaines, contribuant ainsi à la croissance rapide des industries installées dans la région de Curitiba.


**Figure 8 : Partenaires du département électrique de l'UFPR**

### 3.2.2 Organisation

Comme vu précédemment les trois principaux domaines d'activité du département sont les suivants :

- **Energie**
  - Système de potentiel
- **Electronique**
  - Circuit d'instrumentation électronique
  - Control et automatique
- **Télécommunication**
  - Electromagnétisme appliqué
  - Système de communication

En 2009, l'événement de grande importance fut la création du programme nocturne du Cours d'Ingénierie Electrique. En plus de continuer à offrir, durant la journée, les spécialités de la formation en électrotechnique et en électronique-télécommunications, le groupe chargé du projet de l'équipe de nuit a constaté la nécessité d'un nouveau profil de professionnels dans la région de Curitiba et dans tout le Brésil. Vint alors la spécialité en systèmes électroniques embarqués, offerte d'une manière sans précédent dans le pays dans le programme nocturne du Cours d'Ingénierie Electrique de l'UFPR. L'offre de la spécialité en Systèmes Electroniques Embarqués au programme nocturne du Cours d'Ingénierie Electrique de l'UFPR représente l'opportunité d'une formation dans un domaine de haute technicité, avec une forte demande de professionnels et de la plus haute importance dans la scène électronique du Brésil.

## 3.3 GICS

Le département d'ingénierie électrique est composé du CIEL (Centro de Instrumentação Eletrônica) comprenant 4 laboratoires (LIEC, LAMMI, GICS et LASICO).

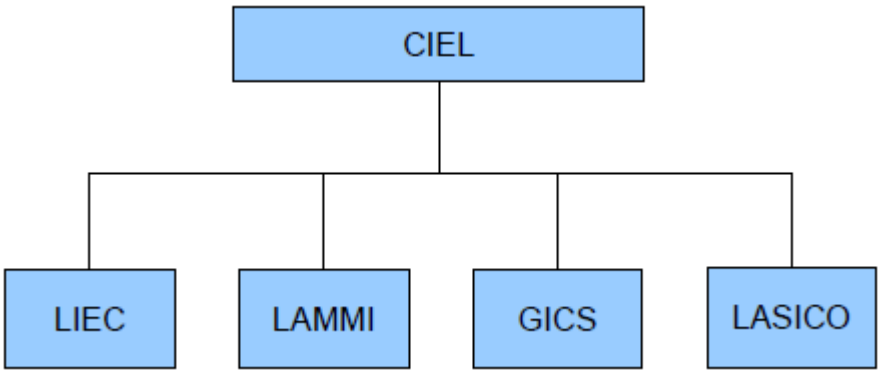



**Figure 9 : Laboratoires du département**

Le GICS, Group of Integrated Circuits and Systems, est dirigé par les professeurs Oscar Gouveia Filho et André Mariano, qui est responsable des élèves français en Projet de Fin d'Etudes. L'un des domaines de recherche du laboratoire est l'intégration de circuits et systèmes en technologie VLSI (Very-Large-Scale Integration) en partenariat avec les entreprises et laboratoires suivants.


**Figure 10 : Partenaires du GICS**

Le laboratoire travaille aussi dans les domaines suivants : Near Field Communication (NFC), compensation de variation dans le processus de fabrication, dispositifs mobiles téléphoniques, Software Defined Radio (SDR), communication sans fil et Radar mobile de nouvelle génération.

# PARTIE N°4. OBJECTIFS DE LA MISSION TECHNIQUE

## 4.1 Sujet de stage

Le but du projet est d'utiliser un pupillomètre, c'est-à-dire un appareil de mesure de la pupille, afin de mesurer l'évolution de sa taille en fonction du temps. Les résultats serviront à des fins médicales. En effet, ces informations peuvent aider au dépistage de maladies telles que la maladie d'Alzheimer, le diabète ou encore au dépistage du taux d'alcoolémie.

Le projet est constitué d'une caméra permettant de réaliser des vidéos de l'œil et plus précisément de la pupille qui est stimulée à l'aide d'un flash lumineux. La vidéo est ensuite traitée à l'aide d'un système embarqué (un FPGA DE2-115 de chez Altera). Les résultats obtenus en fin de traitement sont les valeurs du rayon de la pupille en fonction du temps. Cela permet de visualiser la réaction de la pupille à une stimulation et d'en déduire si son comportement est normal ou non.

Le sujet de mon stage est le suivant : « Développement d'un algorithme pour le traitement d'images médicales sur un système embarqué ».

Mon travail au sein du projet est de développer la partie de traitement d'image sur la pupille et de filtrage des résultats. Je dispose des images obtenues via la caméra et je dois en extraire le rayon et le filtrer pour ensuite pouvoir exploiter ces résultats sous forme de courbe par exemple.

Les parties de traitement d'image et de filtrage des résultats seront développées à la fois sur MATLAB et sur FPGA pour permettre une comparaison des résultats obtenus.

Mon travail est de développer la partie MATLAB et la partie filtrage en VHDL, puis de comparer les résultats.

## 4.2 Cahier des charges fonctionnel

Nous voulons mesurer la taille d'une pupille en fonction du temps à partir d'une vidéo de l'œil. Pour cela il faut séparer la vidéo en images qui sont envoyées sur le FPGA (étape en vert sur l'organigramme de la figure 11).

Chaque image sera traitée afin d'acquérir la taille de la pupille et plus précisément le rayon de la pupille (étape en bleu, fig.11). Lorsque nécessaire, un pré-traitement sera effectué sur les images affectées par le flash lumineux. Puis suivra un traitement d'image permettant de localiser la pupille et d'en calculer sa taille.

Une fois que toutes les images auront été traitées, les résultats seront filtrés afin d'éliminer les erreurs éventuelles (étape en violet, fig.11).

Les algorithmes seront développés à la fois en VHDL et sur MATLAB afin de comparer les résultats en fin de traitement. Cela servira notamment à valider la partie VHDL.

Les résultats après le traitement d'image ainsi que les résultats en fin de chaîne de traitement, c'est-à-dire après filtrage, seront comparés (étape en rouge, fig.11).

L'organigramme suivant présente les différentes étapes du projet.

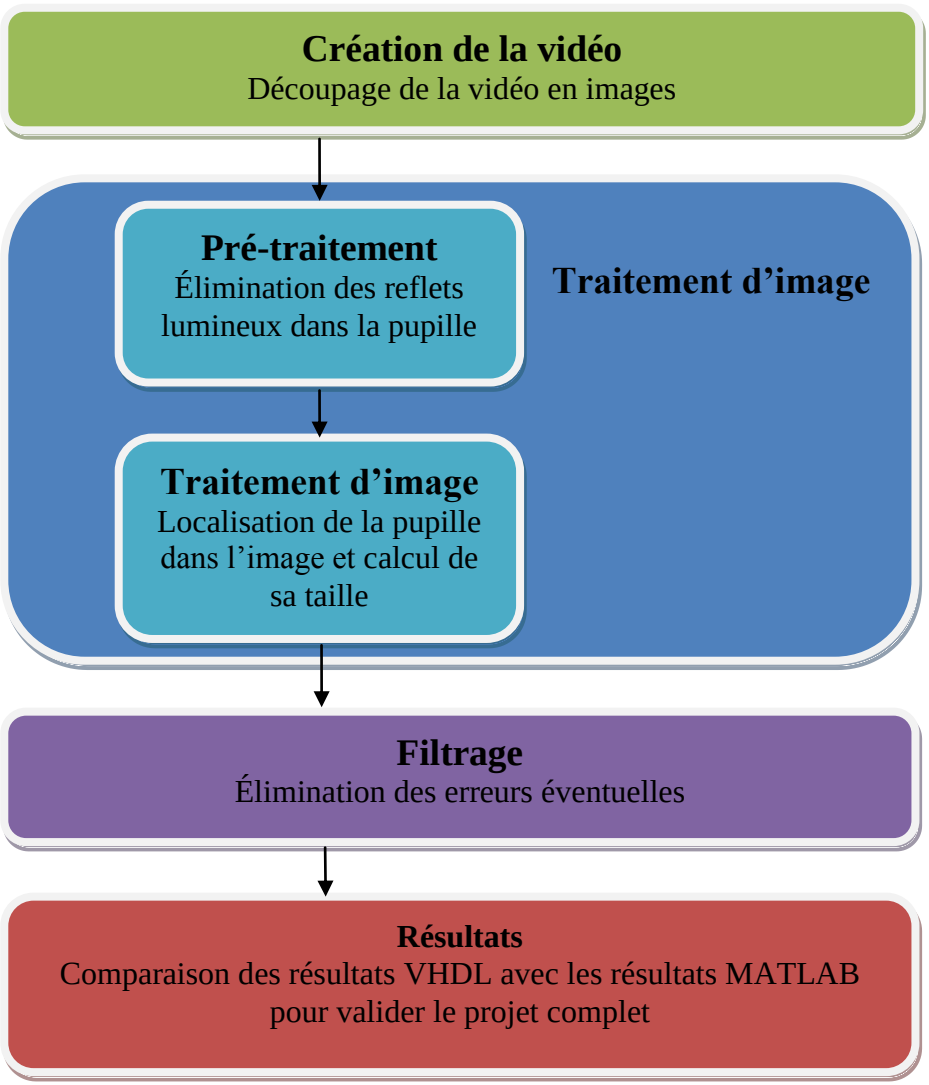



**Figure 11 : Organigramme**

Un résumé du rapport en anglais sera rédigé afin de permettre une réutilisation du projet par la suite par l'UFPR.

## 4.3 Critères de validation du projet

Le premier critère de validation concerne le bon fonctionnement de la partie de pré-traitement sur MATLAB, partie permettant d'éliminer les reflets lumineux dus au flash.

Le second critère de validation concerne le bon fonctionnement de la partie de traitement d'image sur MATLAB. Cette partie devra fournir la valeur du rayon de la pupille sur l'image.

Le troisième critère de validation concerne le bon fonctionnement de la partie filtrage tant sur MATLAB qu'en VHDL.

Le quatrième critère consiste à vérifier que les résultats obtenus à l'aide de MATLAB et du VHDL sont cohérents. Cette étape permettra de valider la partie VHDL et de l'utiliser dans des projets futurs.

La validation du projet complet se fera à l'aide de ce quatrième critère.

## 4.4 Moyens mis à disposition du stagiaire

Le matériel suivant est mis à disposition :

- Un FPGA de chez Altera de type DE2-115
- Une caméra de chez Toshiba de type TCM8230MD compatible avec le FPGA et permettant d'enregistrer des vidéos de l'œil
- Le logiciel Quartus II permettant l'analyse et la synthèse de designs HDL
- Le logiciel de simulation ModelSim qui fournit un environnement complet de simulation et de débogage pour les designs complexes en ASIC et en FPGA
- Le logiciel MATLAB permettant l'analyse des données obtenues

## 4.5 Planification du projet

Le projet a été divisé en plusieurs phases.

La première consistait à lire la documentation proposée.

La seconde phase consistait à utiliser une carte mémoire afin d'envoyer une image sur le FPGA puis d'envoyer directement l'image venant de la caméra sur le FPGA. Cette phase a été interrompue pour deux raisons :

- La phase concernant la carte mémoire a été abordée puis mise en suspend, car la caméra a été achetée et reçue à l'UFPR.
- La phase concernant la caméra a été abandonnée, car la caméra a grillé lors d'un test effectué par l'un des élèves du projet.

La troisième phase a vu le développement du programme de filtrage en VHDL, son amélioration et sa validation par simulation.

La quatrième phase consistait à faire la communication entre l'ordinateur, via MATLAB, et le FPGA. Cette phase a été abordée puis mise en suspend, car un autre groupe de projet l'a développée.

La cinquième phase a vu le développement des programmes de pré-traitement, de traitement d'image et de filtrage sur MATLAB ainsi que leur validation par simulation.

La sixième phase a servi à simuler le projet et à comparer les résultats à l'aide d'un programme écrit sur MATLAB.

La septième et dernière phase a servi à la rédaction du rapport et du résumé du rapport en anglais.

Le projet a été réalisé entre le 27 février et le 4 septembre 2012, soit 28 semaines.

Ci-dessous sont détaillés la planification du projet faite à l'aide de deadlines et le projet tel qu'il a réellement été développé avec des diagrammes de Gantt pour simplifier l'explication des différentes phases.

Voici le projet tel qu'il a été planifié :

| | février | mars | avril | mai | juin | juillet | août | septembre |
|---|---|---|---|---|---|---|---|---|
| Documentation | | | | | | | | |
| Carte mémoire - FPGA | | | | | | | | |
| Image caméra vers le FPGA | | | | | | | | |
| Filtrage VHDL | | | | | | | | |
| Communication PC-FPGA | | | | | | | | |
| Projet en MATLAB | | | | | | | | |
| Simulations/comparaison de résultats | | | | | | | | |
| Rapport | | | | | | | | |

**Tableau 1 : Diagramme de Gantt initial**

Voici l'avancement réel du projet dans le temps :

| | février | mars | avril | mai | juin | juillet | août | septembre |
|---|---|---|---|---|---|---|---|---|
| Documentation | | | | | | | | |
| Carte mémoire - FPGA | | | | | | | | |
| Image caméra vers le FPGA | | | | | | | | |
| Filtrage VHDL | | | | | | | | |
| Communication PC-FPGA | | | | | | | | |
| Projet en MATLAB | | | | | | | | |
| Simulations/comparaison de résultats | | | | | | | | |
| Rapport | | | | | | | | |

**Tableau 2 : Diagramme de Gantt final**

# PARTIE N°5. REALISATION DU PROJET

## 5.1 Analyse du cahier des charges et de ses contraintes

Les programmes développés devront être les plus généraux possibles pour pouvoir être réutilisés et adaptés à de nouvelles vidéos.

La partie concernant la découpe de la vidéo se fera sur MATLAB.

L'algorithme choisi pour l'étape de pré-traitement a été repris d'un projet développé à l'UFPR concernant l'identification de personnes (cf. 5.2 de l'article [6]).

L'algorithme servant à extraire la valeur du rayon de l'image a également déjà été utilisé par l'UFPR. Il a été développé par Weiqi Yuan, Lu Xu et Zhonghua Lin et est, en réalité, un algorithme de détection de l'iris. C'est pourquoi nous n'utiliserons que la première partie de l'algorithme, partie permettant de détecter la pupille (cf. article [7] jusqu'à la fin de la partie III.B.).

La partie de filtrage sera faite de la même manière que décrite dans l'article [6] à la page 4. Cet algorithme de filtrage a lui aussi été repris d'un projet développé à l'UFPR.

Enfin la comparaison des résultats VHDL avec les résultats MATLAB se fera sur MATLAB.

Pour une plus grande clarté, voici un schéma explicatif concernant les algorithmes à utiliser.

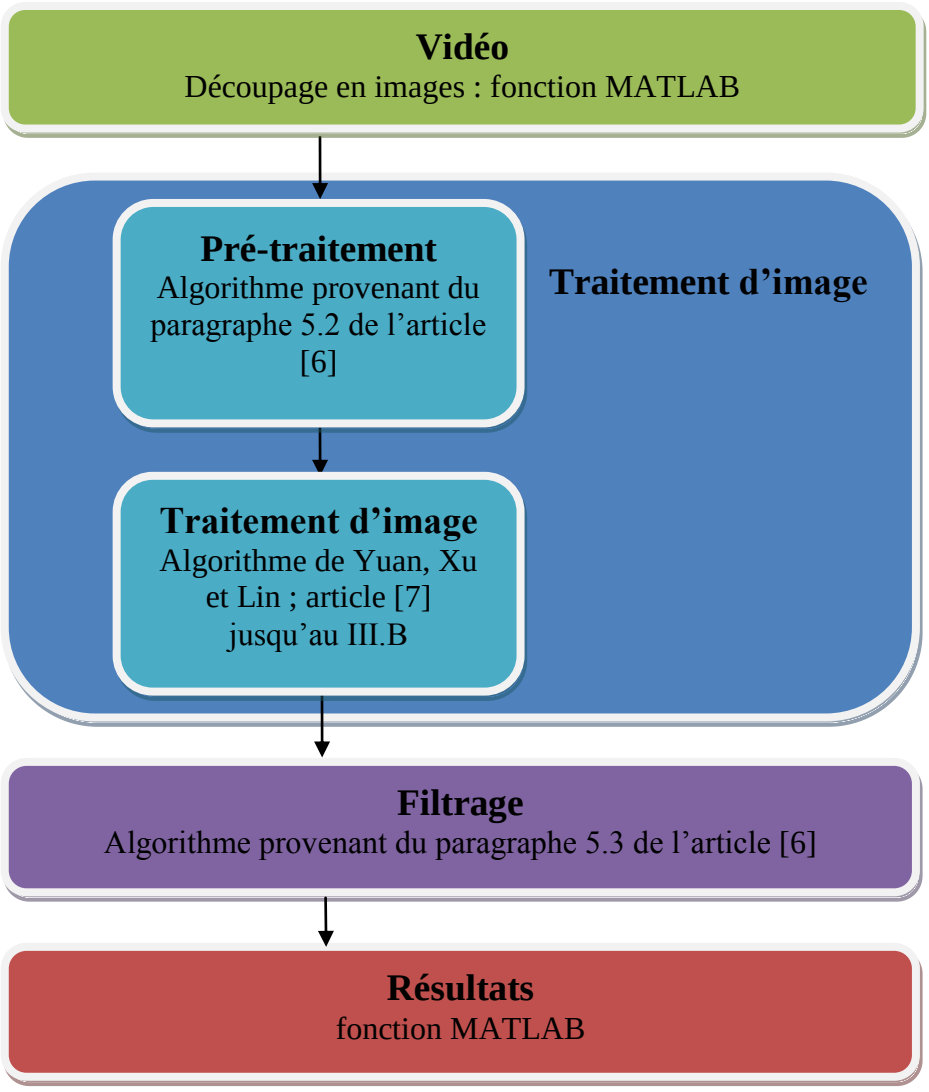



**Figure 12 : Organigramme des algorithmes utilisés**

## 5.2 Conception générale

Une équipe de quatre élèves a travaillé sur le projet et chaque membre de l'équipe s'est focalisé sur une partie spécifique. La figure 13 est un schéma représentant une vue globale du projet :

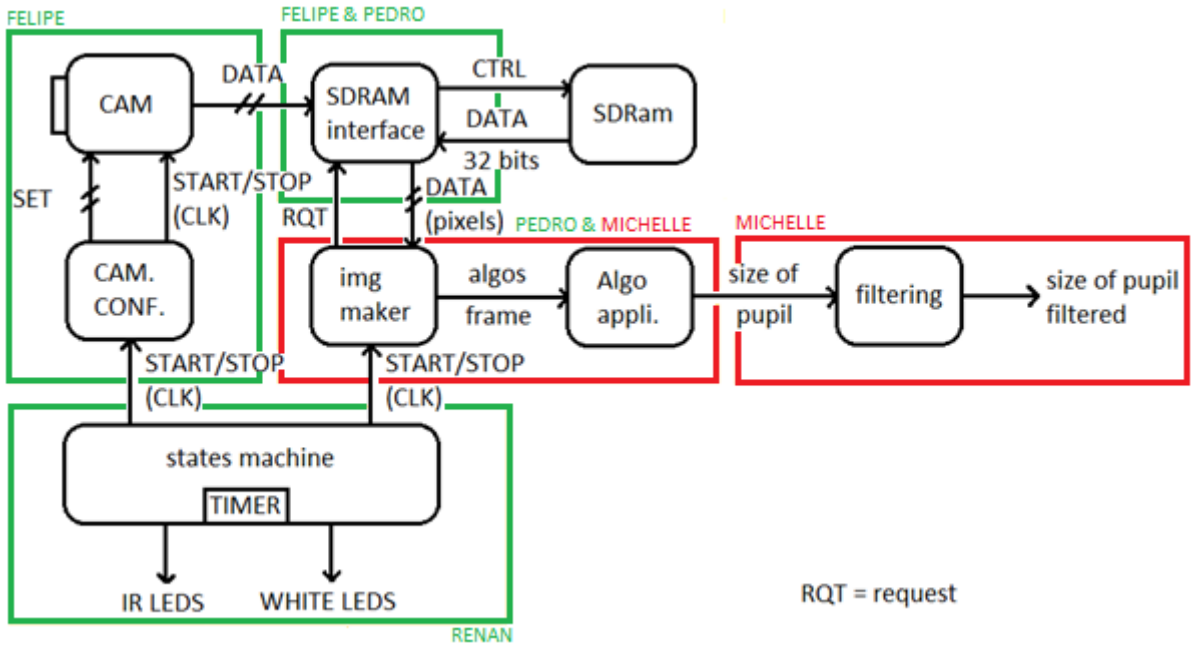



**Figure 13 : Vue globale du projet**

L'utilisateur interagit avec le système de reconnaissance de la pupille à l'aide d'une interface homme-machine. Cette interface permet de signaler les différents états de la machine à l'utilisateur par l'intermédiaire de LEDs.

La partie « machine » est constituée des sous-parties suivantes :

- Configuration de la caméra,
- Interface entre la caméra et la SD Ram,
- Mise en mémoire d'une image dans un buffer,
- Algorithme de traitement d'image permettant d'éliminer les reflets lumineux sur la pupille et de mesurer le rayon de la pupille,
- Filtrage des données (rayon de la pupille en fonction du temps) permettant l'élimination du bruit et le lissage de la courbe à l'aide de filtres moyenneur et gaussien.

Le travail à accomplir au sein du projet est le développement de la partie de filtrage en langage VHDL et les parties de traitement d'image et de filtrage sur MATLAB afin de permettre une comparaison des résultats obtenus. Le travail consiste aussi au développement d'une fonction MATLAB permettant de découper une vidéo en images et de les enregistrer au format de mon choix et d'une fonction permettant l'analyse et la comparaison des résultats VHDL et MATLAB.

## 5.3 Conception détaillée

### *5.3.1 Acquisition des vidéos [6]*

Un système de pupillométrie fut utilisé pour la stimulation de la pupille et pour l'enregistrement des réflexes pupillaires.

Le système, représenté sur la figure 14, est composé d'une caméra monochromatique de lentille 25 mm, d'un cône d'aluminium de 17 cm de hauteur, de diamètre le plus grand de 14,5 cm et le plus petit de 5,6 cm, peint intérieurement en noir pour éviter les reflets internes, d'un ensemble de quatre LEDs infrarouges de diamètre 5 mm et d'un autre ensemble de 5 LEDs blanches de haute luminosité de diamètre 8 mm. L'enregistrement d'images à travers la caméra est indépendant du système d'illumination et de stimulation.

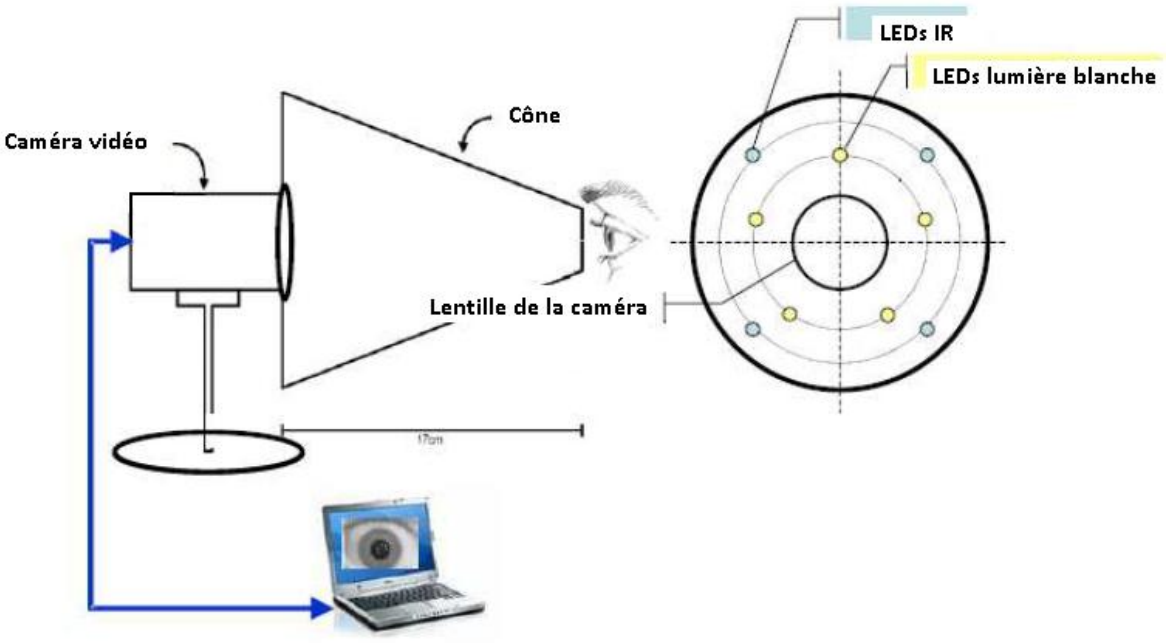



**Figure 14 : Pupillomètre**

La procédure suivante a été adoptée pour capturer les vidéos : le volontaire maintient l'un des yeux appuyé contre le cône pour éviter les interférences dues à la lumière extérieure tandis que l'autre œil est couvert à l'aide d'un tissu de couleur foncée pour qu'il n'y ait pas d'effet de réflexion consensuelle, c'est-à-dire, pour que la lumière incidente de l'autre œil ne cause pas d'altération sur la pupille que l'on enregistre. La lumière infrarouge, invisible à l'œil humain, permet la capture de l'image par la caméra sans stimulation de la pupille.

Après deux minutes avec l'œil ouvert en contact avec le cône, c'est-à-dire sans contact avec la lumière, la pupille devrait avoir atteindre son écarquillement maximal. C'est à ce moment que l'enregistrement des images de la pupille commence et qu'un flash de lumière d'une durée de 10 ms est envoyé à l'aide de l'ensemble de LEDs blanches. Cette durée ne cause pas d'interférences significatives dans le traitement de la vidéo, car elle est inférieure au temps de capture d'une image ; elle peut affecter, au maximum, une image de la vidéo. Par ailleurs, les normes de sécurité concernant la radiation appliquée directement à l'œil humain sont respectées [8].

### *5.3.2 Pré-traitement des images [6]*

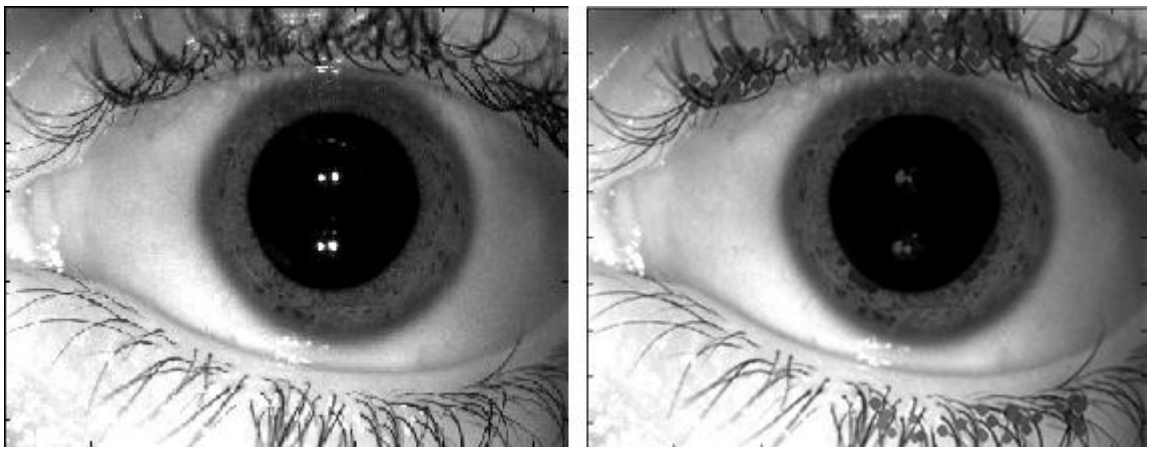
Un ensemble de LEDs est utilisé pour illuminer l'œil pendant la capture d'images, les reflets dus aux foyers lumineux sont aussi enregistrés sur les images. Cela peut interférer dans le processus de détection des bords de la pupille et, par conséquent, peut aussi interférer dans la localisation de la pupille. Un algorithme permettant la suppression de ces reflets lumineux fut développé.

Pour chaque pixel de l'image, $v_m$, la valeur moyenne de l'intensité du pixel et de ses voisins proches (un point dans chacune des directions : au-dessus, en-dessous, à gauche et à droite), et $v_b$, la moyenne des points localisés à une distance r dans chacune des directions, sont calculées. La valeur r correspond au rayon des reflets lumineux que l'on désire éliminer. Un rayon très grand pourrait induire à des détections incorrectes et à des altérations non voulues sur l'image, alors qu'un rayon très petit pourrait réduire l'efficacité de la méthode, en ne tenant pas compte d'un ou plusieurs reflets à éliminer. Après vérification de la taille moyenne des reflets lumineux dans l'image, nous avons choisi un diamètre égal à 13, c'est-à-dire r = 6,5.

Il a été établi que, dans le cas où l'intensité des points voisins est supérieure à deux fois celle des points distants ($v_m > 2 * v_b$), le point en question $(x, y)$ serait proche du centre d'un reflet lumineux. Cette relation a été obtenue expérimentalement par l'analyse des niveaux de gris des reflets de l'image et de leurs régions voisines. Une valeur inférieure peut entraîner une identification incorrecte des reflets lumineux, alors qu'une valeur supérieure peut avoir comme conséquence l'ignorance de certains reflets, car les niveaux de gris de la région sont plus élevés que ceux de la pupille.

Une fois le reflet localisé, les étapes suivantes doivent être suivies pour l'éliminer. Les points appartenant aux cercle de rayon r et de centre $(x, y)$ sont remplacés par la valeur moyenne des points distants, $v_b$.

La figure 15 montre la façon dont les reflets à éliminer apparaissent dans l'image originale et le résultat du processus d'élimination.



**(a)** **(b)**

**Figure 15 : (a) image originale et (b) image après suppression des reflets lumineux**

**Remarque :** S'il n'y a pas de reflet lumineux sur l'image, le pré-traitement est inutile et peut entraîner des erreurs dans le calcul du rayon.

### *5.3.3 Algorithme de traitement d'image : Algorithme de Weiqi Yuan, Lu Xu, Zhonghua Lin [7]*

#### 5.3.3.1 Description de l'algorithme

L'algorithme original a été légèrement modifié pour être efficace sur les images que nous possédons.

##### 5.3.3.1.1 Recherche d'un point dans la pupille

- **Création du GVSO (Gray Value Summing Operator)**

Sur l'image de l'œil humain, la pupille est la région la plus foncée. Cela signifie que la valeur de gris moyenne de la pupille est la valeur minimum. De plus, la distribution de gris de la pupille est relativement homogène et la forme de la pupille est proche de celle d'un cercle. En utilisant des éléments caractéristiques, nous pouvons facilement trouver un point dans la pupille de manière exacte.

Soit $I(x,y)$ un point de l'image de l'œil humain et $f(i,j)$ la valeur de gris du point $(i,j)$ ; ici $i$ est compris entre 1 et la largeur de l'image (width) et $j$ est compris entre 1 et la hauteur de l'image (height). La coordonnée $x$ du point $(x,y)$ correspond au nombre de colonnes dans l'image et la coordonnée $y$ correspond au nombre de lignes.

Le GVSO (formule (1)) est conçu de la façon suivante :

Sa taille est de $m * n$ pixels, c'est-à-dire qu'il a $m$ colonnes et $n$ lignes ; $m$ et $n$ sont des nombres impairs. Soit $P_c(x_c, y_c)$ le centre du GVSO, la valeur du GVSO est calculée en utilisant la formule (2) avec $S(x_c, y_c)$ qui a pour centre le point $(x_c, y_c)$.

$$\begin{pmatrix} f(x_c-\frac{n-1}{2}, y_c-\frac{m-1}{2}) & \dots & f(x_c, y_c-\frac{m-1}{2}) & \dots & f(x_c+\frac{n-1}{2}, y_c-\frac{m-1}{2}) \\ \vdots & \ddots & \vdots & \iddots & \vdots \\ f(x_c-\frac{n-1}{2}, y_c) & \cdots & f(x_c, y_c) & \cdots & f(x_c+\frac{n-1}{2}, y_c) \\ \vdots & \iddots & \vdots & \ddots & \vdots \\ f(x_c-\frac{n-1}{2}, y_c+\frac{m-1}{2}) & \cdots & f(x_c, y_c+\frac{m-1}{2}) & \cdots & f(x_c+\frac{n-1}{2}, y_c+\frac{m-1}{2}) \end{pmatrix} \quad \textbf{(1)}$$

$$S(x_c, y_c) = \sum_{i=x_c-\frac{n-1}{2}}^{x_c+\frac{n-1}{2}} \sum_{j=y_c-\frac{m-1}{2}}^{y_c+\frac{m-1}{2}} f(i,j) \quad \textbf{(2)}$$

- **Etape de recherche d'un point dans la pupille**

→ **Choix d'une aire de recherche**

Afin d'éviter les erreurs dues à des pixels oubliés ou à des débordements, il est nécessaire de déterminer un rectangle correspondant à l'aire de recherche dans l'image. La taille de l'aire du rectangle est la suivante : $(width - n) * (height - m)$. L'intervalle correspondant aux colonnes est de $[\frac{(n-1)}{2}, width - \frac{(n+1)}{2}]$ et celui des lignes est de $[\frac{(m-1)}{2}, height - \frac{(m+1)}{2}]$.

→ **Détection de la coordonnée horizontale $x_0$ de $P_0$**

Soit $m = width^{1/4}$ et $n = width^{1/2}$. Pour chaque pixel $(x_i, y_i)$ du domaine de recherche, pixel considéré comme le centre du GVSO, sa valeur $S_i(x,y)$ est calculée. Un vecteur de la forme $\{S_1(x,y), S_2(x,y), S_3(x,y), \dots\}$ est obtenu. Il faut ensuite trouver la valeur minimale du vecteur, valeur appelée $S_{min}(x,y)$, et la coordonnée $x_c$ du point central $(x_c, y_c)$ de $S_{min}$ sera la coordonnée horizontale du point de la pupille, c'est-à-dire $x_0 = x_c$.

→ **Détection de la coordonnée verticale $y_0$ de $P_0$**

Soit $m = width^{1/2}$ et $n = width^{1/4}$. Les étapes du paragraphe précédent sont répétées. Mais cette fois, la coordonnée $y_c$ du point central $(x_c, y_c)$ de $S_{min}$ sera la coordonnée verticale du point de la pupille, c'est-à-dire $y_0 = y_c$.

Après avoir effectué ces trois étapes, un point $P_0(x_0, y_0)$ situé dans la pupille est obtenu (cf. figure 16).

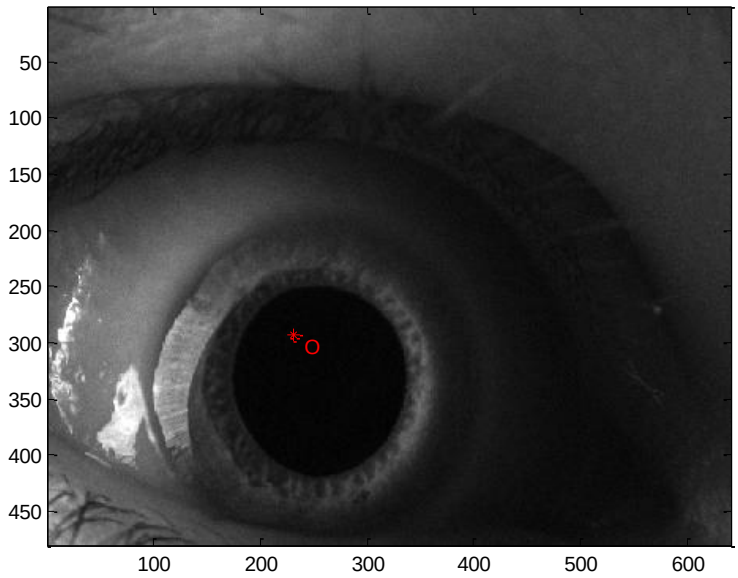



**Figure 16 : Point O trouvé à l'intérieur de la pupille**

5.3.3.1.2 Recherche des bords de la pupille

Trois points non alignés définissent un cercle. Ceci signifie que les paramètres définissants le bord de la pupille, qui est un cercle, pourront facilement être calculés si l'on arrive à trouver 3 points appartenant à la frontière pupille-iris. Les calculs à effectuer sont détaillés dans l'annexe 1.

En nous servant du point $P_0$, nous allons chercher le bord gauche de la pupille, ainsi que le bord droit et le bord du bas, comme représenté sur la figure 17.

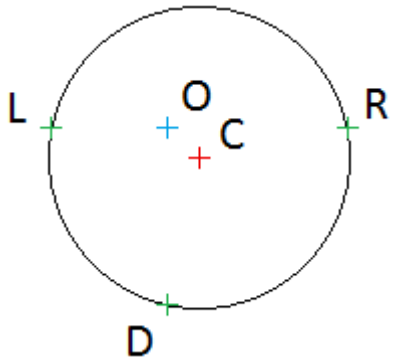



**Figure 17 : C centre de la pupille calculé à l'aide des coordonnées des points L, R et D**

- **Calcul du modèle de détection de la frontière (BDT)**

Dans l'image de l'œil, la valeur moyenne de gris de la pupille est la valeur minimale, celle de la sclérotique (tissu qui compose le blanc de l'œil) est maximale et celle de l'iris est entre les deux

valeurs précédentes. De plus la frontière pupille-iris a une intensité de bord maximale. L'algorithme proposé dans l'article [7] détecte l'intensité de bord maximale de certains points. Cependant les résultats obtenus à l'aide des outils classiques pour la détection des bords ne sont pas satisfaisants. Il y a deux raisons à cela. Premièrement les cils et les veines dans la région de l'iris produisent du bruit dans l'image. Deuxièmement les opérateurs n'utilisent qu'une seule valeur de référence pour la détection de bords. Le bruit va induire ces opérateurs en erreur. Afin de trouver les bords et de calculer les paramètres du cercle de manière exacte, un opérateur nommé Modèle de détection de la frontière ou Boundary Detection Template (BDT) a été développé. Cet opérateur est composé de deux valeurs déterminantes et est difficilement affecté par le bruit.

La première direction du BDT est horizontale (cf. figure 18), sa largeur est de *m* pixels et sa hauteur est de *n* pixels (où *m* et *n* sont des nombres impairs et $\frac{(m-1)}{2} > n$). Soit $O(x, y)$, une ligne horizontale (long axis) passant par *O* divise le BDT en deux parties : la partie haute et la partie basse, et la ligne verticale (short axis) passant par *O* divise également le BDT en deux parties : la partie gauche et la partie droite. Ainsi la direction du modèle se trouve le long de l'axe horizontal (long axis). De plus la direction du modèle doit être la même que la direction de détection.

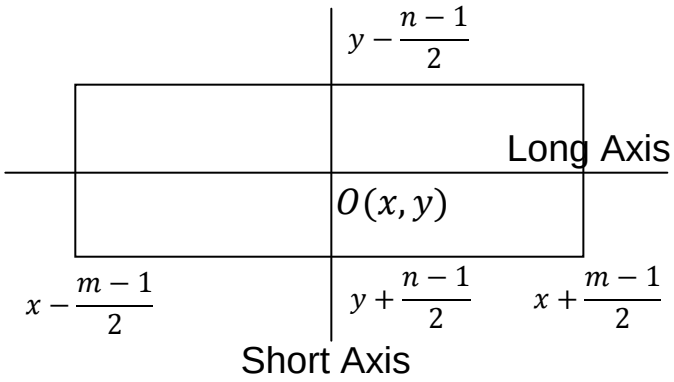



**Figure 18 : Modèle de détection de frontière - Boundary detection template (BDT)**

Dans ce paragraphe, trois problèmes seront éclaircis :

(1) Les nombres *m* et *n* sont impairs pour que, lorsque l'on considère le pixel $O(x, y)$ comme le centre du BDT, les deux régions rectangles de part et d'autre de l'axe vertical (short axis) aient le même nombre de pixels.

(2) On fixe $\frac{(m-1)}{2} > n$. Cela permet de réduire l'influence des pixels sans importance pour le résultat à calculer et permet aussi au BDT de ne pas être affecté par le bruit.

(3) La direction du BDT doit être la même que la direction de détection, c'est-à-dire que, lorsque la recherche est faite sur l'axe horizontale, le BDT est tel que sur la figure 18 et lorsque la recherche est faite sur l'axe vertical, le BDT est tel que le long axis soit vertical et le short axis soit horizontal. Cela garantit que les valeurs déterminantes (*m* et *n*) sont les mêmes quelque soit la direction de détection.

→ **Intensité de bord**

La première valeur déterminante est l'intensité de bord (ou *edge intensity)*.

Pour chaque pixel $(x, y)$ du domaine de recherche, ce pixel sera considéré comme le centre du BDT et l'intensité de bord sera calculée à l'aide des équations (3) ($dif^1$ et $dif^2$ sont respectivement utilisés pour les directions horizontales et verticales). Lorsque cette valeur est maximale, le pixel $(x, y)$ est un bord.

$$\begin{cases} dif^1(x,y) = \left| \sum_{i=x-\frac{m-1}{2}}^{x-1} \sum_{j=y-\frac{n-1}{2}}^{y+\frac{n-1}{2}} f(i,j) - \sum_{p=x+1}^{x+\frac{m-1}{2}} \sum_{q=y-\frac{n-1}{2}}^{y+\frac{n-1}{2}} f(p,q) \right| \\ dif^2(x,y) = \left| \sum_{i=x-\frac{n-1}{2}}^{x+\frac{n-1}{2}} \sum_{j=y-\frac{m-1}{2}}^{y-1} f(i,j) - \sum_{p=x-\frac{n-1}{2}}^{x+\frac{n-1}{2}} \sum_{q=y+1}^{y+\frac{m-1}{2}} f(p,q) \right| \end{cases} \quad \textbf{(3)}$$

→ **Valeur moyenne de gris**

La seconde valeur déterminante est la valeur moyenne de gris (ou *average gray value*). Elle peut être décrite comme la valeur moyenne de gris de tous les pixels contenus dans la moitié du modèle, c'est-à-dire la partie gauche du BDT si la direction est « gauche », droite si la direction est « droite » et basse si la direction est « bas ». Concernant le point de la frontière entre la pupille et l'iris, s'il est considéré comme le centre du BDT, tous les pixels dans la moitié du modèle qui est proche du centre de la pupille doivent être à l'intérieur de la pupille ; la valeur moyenne de cette moitié doit être environ égale à la valeur de gris moyenne de la pupille. Par conséquent cette mesure peut aussi être utilisée dans la détermination d'un point de la frontière.

Pour chaque pixel $(x, y)$, la deuxième valeur déterminante peut être calculée à l'aide de la formule (4) ($avg^1$,$avg^2$ et $avg^3$ sont respectivement utilisés pour les côtés gauche, droit et bas). Lorsque la valeur moyenne de gris est inférieure à la valeur de gris de référence de la pupille, le point $(x, y)$ ne sera pas considéré comme du bruit.

$$\begin{cases} avg^1(x,y) = \left[ \sum_{i=x+1}^{x+\frac{m-1}{2}} \sum_{j=y-\frac{n-1}{2}}^{y+\frac{n-1}{2}} f(i,j) \right] / \left( n * \frac{m-1}{2} \right) \\ avg^2(x,y) = \left[ \sum_{i=x-\frac{m-1}{2}}^{x-1} \sum_{j=y-\frac{n-1}{2}}^{y+\frac{n-1}{2}} f(i,j) \right] / \left( n * \frac{m-1}{2} \right) \\ avg^3(x,y) = \left[ \sum_{i=x-\frac{n-1}{2}}^{x+\frac{n-1}{2}} \sum_{j=y-\frac{m-1}{2}}^{y-1} f(i,j) \right] / \left( n * \frac{m-1}{2} \right) \end{cases} \quad \textbf{(4)}$$

- **Recherche des bords gauche, droit et bas**

Dans l'image de l'œil, la recherche commence par le point $P_0(x_0, y_0)$, trouvé dans le paragraphe 5.3.3.1.1. Lorsque le BDT est utilisé dans les directions gauche horizontale, droite horizontale et bas verticale, trois points appartenant à la frontière pupille-iris peuvent être obtenus (respectivement $P_L$, $P_R$ et $P_D$), comme l'a été montré plus haut, dans la figure 17. Ensuite les paramètres du cercle correspondant à la frontière seront calculés précisément selon le principe suivant : trois points non alignés définissent un cercle.

Les étapes de recherche de la frontière pupille-iris sont les suivantes :

(1) Détermination de l'espace de recherche.

- Dans la direction horizontale, la détection des points de la frontière se fait le long de la ligne $y = y_0$.
  - L'espace de recherche du côté gauche est : $\left[\frac{n-1}{2}, x_0 - 1\right]$
  - L'espace de recherche du côté droit est : $\left[x_0 + 1, Width - \frac{m+1}{2}\right]$

- Dans la direction verticale, la détection des points de la frontière se fait le long de la ligne $x = x_L + 20$.
  - L'espace de recherche du côté bas est : $\left[y_0 + 1, Height - \frac{n-1}{2}\right]$

(2) Déterminer la valeur de gris de référence.

La valeur de référence a été fixée à $v = 2 * v_0$. Après avoir effectué plusieurs tests sur les images de l'UFPR, il a été conclu que cette valeur est la plus pertinente pour notre étude.

(3) Utilisation du BDT pour détecter le premier point appartenant à la frontière, point se trouvant dans la zone de recherche de gauche.

Pour chaque pixel $(x, y)$, considéré comme le centre du BDT,

- L'intensité de bord *dif* est calculée (en utilisant $dif^1$ pour la direction horizontale) et un vecteur $\{dif_1, dif_2, dif_3, \dots, dif_n\}$ est obtenu. Chaque $dif_i$ correspondant à l'intensité de bord du BDT avec $P_i(x_i, y_i)$ considéré comme le centre du BDT.
- Création d'un nouveau vecteur contenant les valeurs de l'intensité de bord triées dans l'ordre décroissant.
- Dans ce nouveau vecteur, trouver la première valeur de l'intensité de bord qui est inférieure à la valeur de gris de référence.

Le point trouvé est le point du bord gauche appelé $P_L(x_L, y_L)$.

(4) L'étape (3) est répétée pour les côtés droit et bas. Elle permet de trouver les points du bord droit appelé $P_R(x_R, y_R)$ et du bord bas appelé $P_D(x_D, y_D)$.

Les trois points obtenus (gauche, droit et bas) sont représentés sur la figure 19 ci-dessous.

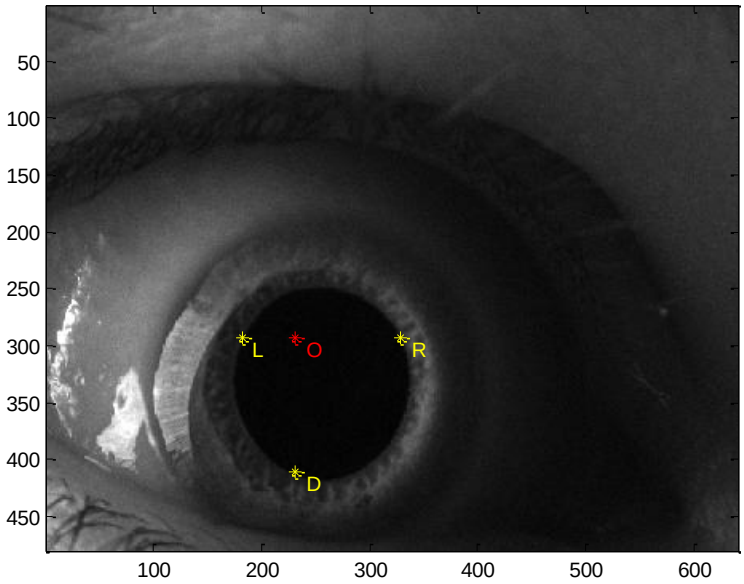



**Figure 19 : Bords gauche, droit et bas de la pupille**

5.3.3.1.3 Calcul du rayon de la pupille

Les coordonnées du centre et du rayon de la pupille sont calculées selon le principe énoncé dans les paragraphes précédents : trois points non alignés définissent un cercle. Les calculs sont détaillés dans l'annexe 1. Les formules (5) sont à appliquer pour obtenir les valeurs du rayon et des coordonnées du centre.

$$\begin{cases} x_C = \dfrac{x_L + x_R}{2} \\ y_C = \dfrac{{x_D}^2 + {y_D}^2 - x_D * (x_L + x_R) - y_R + x_R * x_L}{2 * (y_D - y_R)} \\ R = \sqrt{(x_D - x_c)^2 + (y_D - y_c)^2} \end{cases} \quad \textbf{(5)}$$

Les résultats obtenus, c'est-à-dire les coordonnées du centre C ainsi que le point dans la pupille O et les trois points L, R et D sur le cercle définissant la pupille, sont représentés sur la figure 20 ci-dessous.

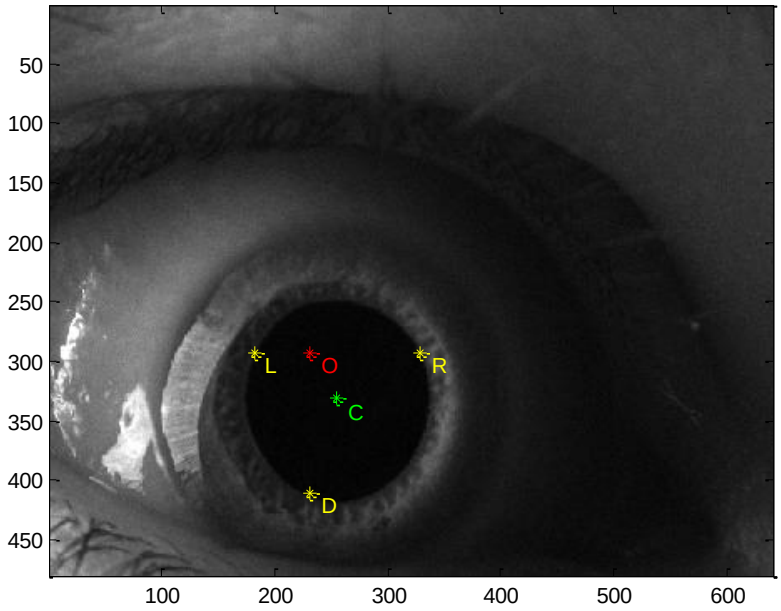



**Figure 20 : Calcul du centre C de la pupille à partir des points L, R et D trouvés**

Sur la figure 21, seul le centre C de la pupille a été représenté. Cela permet d'observer de façon plus claire l'efficacité de l'algorithme.

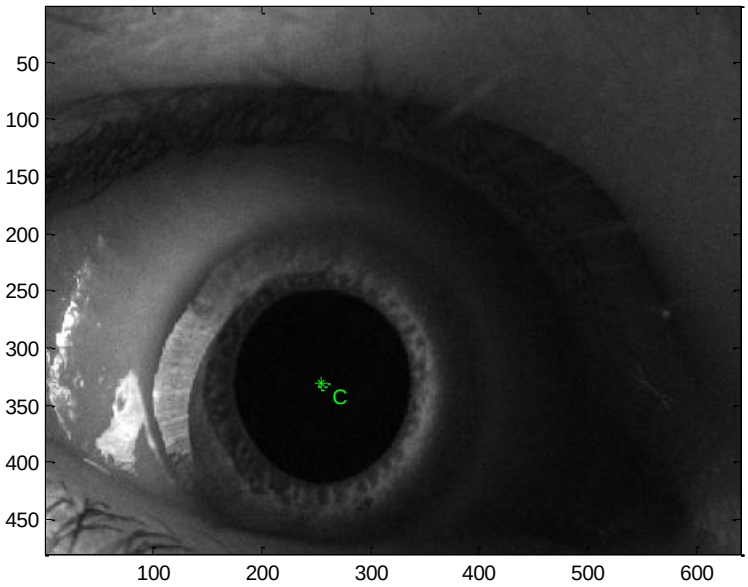



**Figure 21 : Centre C de la pupille**

#### 5.3.3.2 Algorithme codé sur MATLAB

L'algorithme décrit précédemment a été développé sur MATLAB afin de permettre une comparaison des résultats obtenus en VHDL. Cela permet néanmoins de valider le travail effectué en VHDL.

L'algorithme a été découpé en plusieurs programmes pour simplifier son utilisation. Le code a été organisé comme décrit sur l'organigramme de la figure 22.

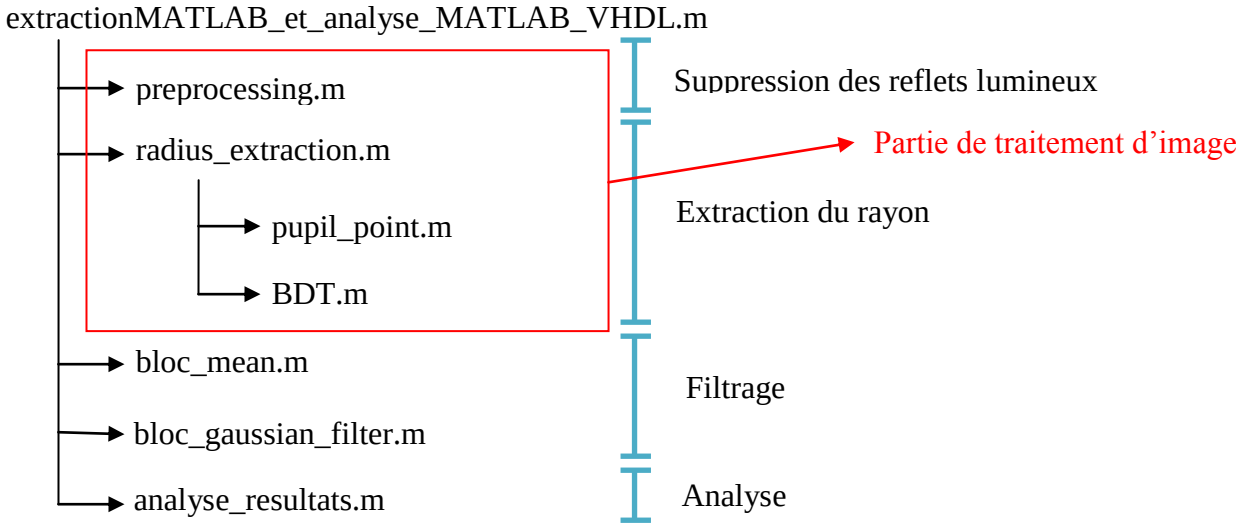



**Figure 22 : Organisation des fonctions sur MATLAB**

Les algorithmes de ces fonctions sont détaillés dans l'annexe 2.

La fonction de pré-traitement (*preprocessing*.m, annexe 2.1) est utilisée pour enlever les reflets lumineux qui peuvent apparaître sur la pupille. Il est nécessaire de les supprimer, car ils pourraient induire des erreurs dans le traitement qui suit. Il n'est utilisé que lorsqu'il y a des reflets sur l'image.

La fonction d'extraction du rayon (*radius_extraction.m*, annexe 2.2) sert à trouver les paramètres du cercle définissant la pupille. Elle est composée de plusieurs sous-fonctions. Ces sous-fonctions sont utilisées pour trouver un point dans la pupille, pour calculer les valeurs de référence et pour calculer le BDT. Ses sous-fonctions sont les suivantes :

- La fonction *pupil_point.m* (annexe 2.2.1) est utile pour trouver un premier point dans la pupille. Ce point servira ensuite de point de départ pour trouver les points du cercle.
  Les paramètres de la fenêtre de recherche sont $m = 25$ et $n = 5$ pour la recherche de la coordonnée horizontale $x_0$ et sont $m = 5$ et $n = 25$ pour la recherche de la coordonnée verticale $y_0$.
  En sortie de la fonction, nous obtenons la liste des coordonnées des points pour lesquels le GVSO, et donc la valeur de S, est minimum. Ensuite les coordonnées du point O situé dans la pupille sont calculées à l'aide des équations (6) et (7).

$$x0 = min(c) + (max(c) - min(c))/2 \quad \textbf{(6)}$$
$$y0 = min(r) + (max(r) - min(r))/2 \quad \textbf{(7)}$$

- La fonction *BDT.m* (annexe 2.2.2) est utilisée pour calculer le BDT et ses deux valeurs déterminantes. Elle permet de trouver les valeurs de l'intensité et de gris de chaque pixel du domaine de recherche.
  Les paramètres utilisés diffèrent suivant le sens de recherche du point. Lorsque la recherche est horizontale, $m = 25$ et $n = 5$. Lorsque la recherche est verticale $m = 5$ et $n = 25$.

#### 5.3.3.3 Algorithme codé en VHDL

La figure 23 est un graphique explicatif de l'algorithme. C'est Pedro Pitt, l'un des élèves travaillant sur le projet, qui a codé la partie VHDL. Ses codes ont été utilisés pour faire les tests sur le projet.

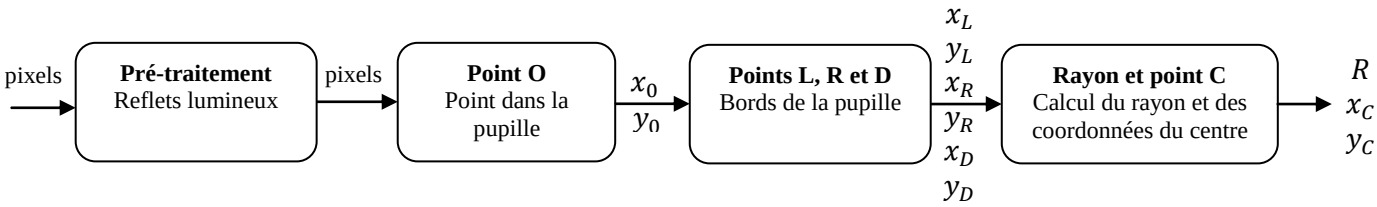



**Figure 23 : Chaîne de traitement d'image**

Les reflets lumineux sur la pupille qui dégradent la qualité de l'image sont supprimés à l'aide du bloc de « Pré-traitement » nommé *white spots* dans le code VHDL. Puis vient l'étape de recherche du point O dans la pupille ; point qui correspondra au centre de la pupille. Cette étape est effectuée par le bloc « Point O » nommé *pupil point*. Il s'agit ensuite de trouver les bords de la pupille pour pouvoir mesurer le rayon de celle-ci. C'est le bloc « Points L, R et D » nommé *Weiqi boundaries* qui se charge de cette étape. Enfin le rayon ainsi que les coordonnées du centre de la pupille sont calculés par le bloc « Rayon et point C » nommé *r_calc*. En sortie, les mesures de la variation du rayon de la pupille en fonction du temps sont enregistrées dans un vecteur. Le rayon de la pupille est mesuré en pixels.

### *5.3.4 Filtrage des données [6]*

Une fois que toutes les mesures sont enregistrées, elles sont filtrées pour supprimer les erreurs ou les éléments qui parasiteraient l'analyse des données. Le filtrage se fait en 2 parties comme présenté dans la figure 24. L'algorithme de la partie filtrage est détaillé dans l'article [6].

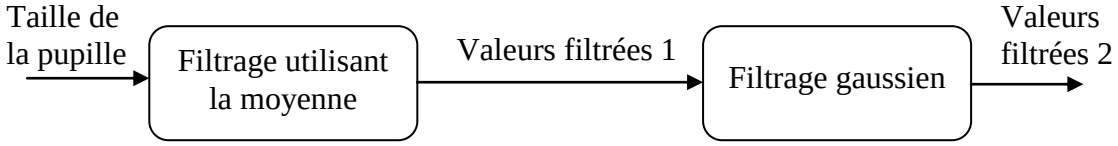



**Figure 24 : Filtrage des données**

Lorsque la pupille et l'iris ne sont pas bien visibles (clignement de l'œil, image trop sombre), la pupille n'est pas localisée correctement. La mesure du rayon de la pupille sera faussée.

Il est possible d'observer les points pour lesquels la localisation de la pupille a échoué à l'aide d'un graphique présentant le rayon de la pupille en fonction du temps. Ces points sont corrigés en appliquant les filtrages suivants :

- Le premier filtre est un filtre moyenneur
- Le second est un filtre gaussien

#### 5.3.4.1 Description de l'algorithme

##### 5.3.4.1.1 Filtre moyenneur

Il est possible, à l'aide d'un graphique du rayon de la pupille en fonction du temps (figure 25), d'observer les points pour lesquels la localisation de la pupille n'a pas fonctionné.

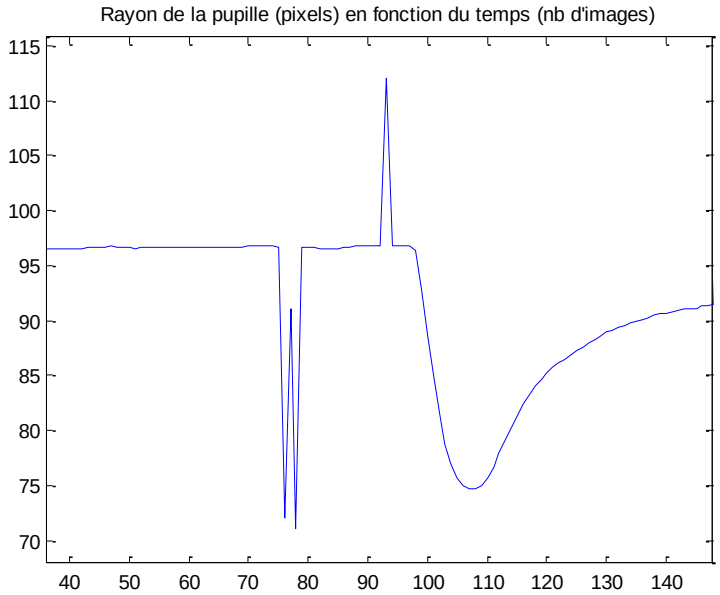



**Figure 25 : Dysfonctionnement de la localisation de la pupille**

Ces points sont corrigés à l'aide du filtre moyenneur décrit comme suit : Pour chaque point (en rouge sur la figure 26), est calculée la moyenne d'une séquence de 3 points à gauche et de 3 points à droite (en vert sur la figure 26).

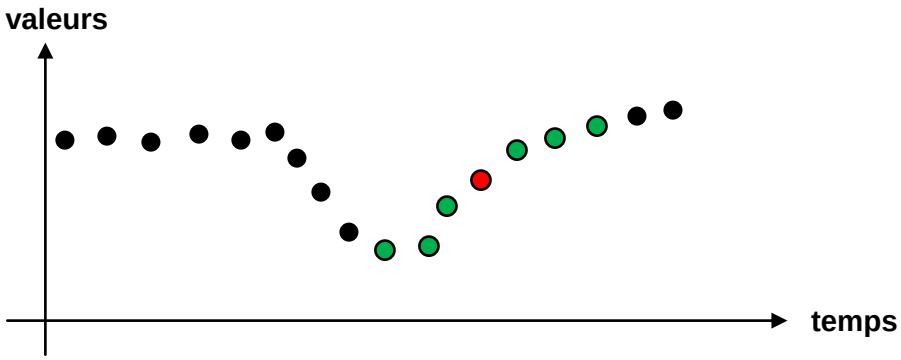



**Figure 26 : Description du filtre moyenneur**

Ensuite la différence entre les modules des moyennes est calculée. Si la différence est supérieure à 2, la valeur du point est remplacée par celle de la moyenne des 6 points. Sinon la valeur du point est gardée.

Les trois premières et trois dernières valeurs de la courbe ne sont pas filtrées. Elles sont simplement recopiées de l'entrée à la sortie du filtrage.

Pour une meilleure compréhension, l'algorithme du filtre moyenneur est détaillé ci-dessous.

```
En entrée : (R(0), R(1), …, R(N-1)) vecteur de la taille de la pupille en fonction du temps
En sortie : (S(0), S(1), …, S(N-1)) vecteur de la taille de la pupille en fonction du temps filtré 1

Début
// Bords: on copie les 3 premiers éléments dans S(i)
S(0) = R(0)
S(1) = R(1)
S(2) = R(2)

Pour i allant de 3 à N-4
    // Initialisation
    A = 0
    B = 0
    Diff_mean = 0

    // Calcul des moyennes de chaque côté du point i
    A = abs((R(i-3) + R(i-2) + R(i-1)) /3)
    B = abs((R(i+1) + R(i+2) + R(i+3)) /3)
    Diff_mean = A – B

    Si Diff_mean > 2 alors
        S(i) = (A + B)/2
    Sinon
        S(i) = R(i)
    Fin si
Fin pour

// Bords: on copie les 3 derniers éléments dans S(i)
S(N-3) = R(N-3)
S(N-2) = R(N-2)
S(N-1) = R(N-1)
Fin
```

#### 5.3.4.1.2 Filtre gaussien

Après la correction des erreurs de localisation, un filtre gaussien de fenêtre de 7 éléments et de résolution σ = 5 est appliqué aux données. L'équation du filtre est la suivante : $G(i) = \frac{1}{\sigma} * e^{-\frac{i^2}{\sigma^2}}$. Le filtre est normalisé afin d'éviter une modification des valeurs mesurées.

Les valeurs du filtre gaussien normalisé, divisé par la somme de ses coefficients égale à 1.2042434, sont les valeurs du tableau ci-dessous.

| G(-3) | G(-2) | G(-1) | G(0) | G(1) | G(2) | G(3) |
|---|---|---|---|---|---|---|
| 0.1158696 | 0.1415235 | 0.1595673 | 0.1660793 | 0.1595673 | 0.1415235 | 0.1158696 |

**Tableau 3 : Coefficients du filtre gaussien normalisé**

La figure 27 représente un graphique du rayon de la pupille en fonction du temps après filtrage.

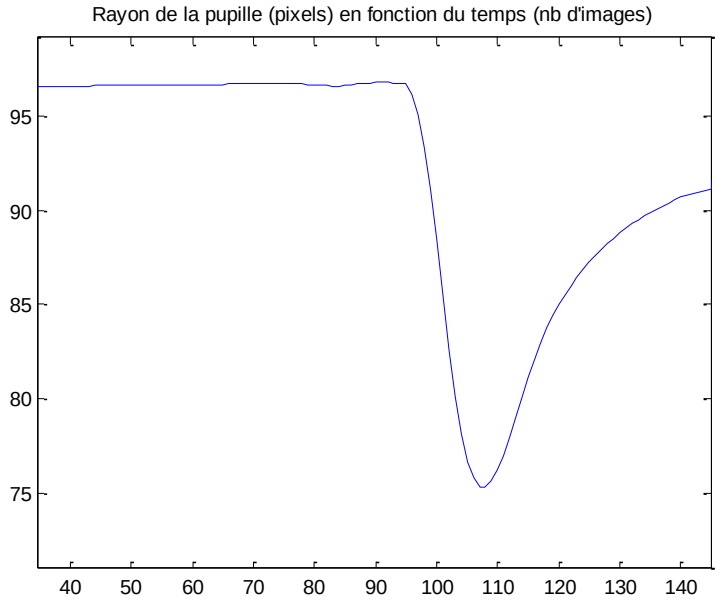



**Figure 27 : Rayon de la pupille en fonction du temps, après filtrage**

Les trois premières et trois dernières valeurs de la courbe ne sont pas filtrées. Elles sont simplement recopiées de l'entrée à la sortie du filtrage.

Pour une meilleure compréhension, l'algorithme du filtre moyenneur est détaillé ci-dessous.

En entrée : (S(0), S(1), …, S(N-1)) vecteur de la taille de la pupille en fonction du temps filtré 1
En sortie : (T(0), T(1), …, T(N-1)) vecteur de la taille de la pupille en fonction du temps filtré 2
σ = 5

Début
// Filtre gaussien normalisé arrondi et multiplié par $10^7$ pour avoir des calculs d'entiers
G = (0.1158696, 0.1415235, 0.1595673, 0.1660793, 0.1595673, 0.1415235, 0.1158696)

// Bords: on copie les 3 premiers éléments dans T(i)
T(0) = S(0)
T(1) = S(1)
T(2) = S(2)

Pour i allant de 3 à N-4
    T(i) = (S(i+3)*G(0) + S(i+2)*G (1) + S(i+1)*G (2) + S(i)*G (3) + S(i-1)*G (4) + S(i-2)*G (5)
    + S(i-3)*G (6))
Fin pour

// Bords: on copie les 3 derniers éléments dans T(i)
T(N-3) = S(N-3)
T(N-2) = S(N-2)
T(N-1) = S(N-1)
Fin

#### 5.3.4.2 Algorithme codé sur MATLAB

L'algorithme codé sur MATLAB est détaillé dans l'annexe 3. Le code MATLAB a été découpé en fonctions et organisé comme suit.

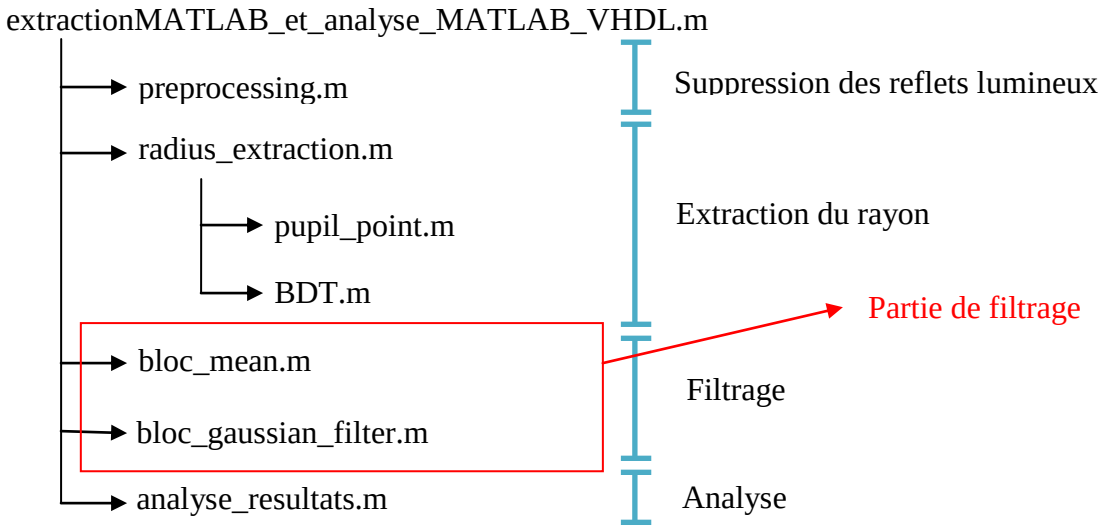



**Figure 28 : Organisation des fonctions sur MATLAB**

**Filtre moyenneur (annexe 3.1)**

Les valeurs du rayon en entrée du filtre moyenneur sont mémorisées dans un vecteur $R$ de N éléments ; celles en sortie sont mémorisées dans un vecteur $S$ de N éléments.

Les 3 premiers et derniers éléments du vecteur $(R(1), R(2), \dots, R(N))$ sont recopiés dans le vecteur $(S(1), S(2), \dots, S(N))$.

Les calculs sont faits sur toutes les valeurs du vecteur à l'exception des bords qui sont recopiés. Les opérations effectuées sont les suivantes :

- Calcul de la moyenne des 3 points précédant le point $R(i)$, c'est-à-dire sur $R(i-1)$, $R(i-2)$ et $R(i-3)$.
- Calcul de la moyenne des 3 points suivant le point $R(i)$, c'est-à-dire sur $R(i+1)$, $R(i+2)$ et $R(i+3)$.
- Calcul de la différence entre les 2 moyennes

Si cette différence est supérieure à 2, la valeur de $R(i)$ sera remplacée par la valeur de la moyenne des 6 points (des 3 points précédent et des 3 points suivant). Sinon la valeur de $R(i)$ sera recopiée dans $S(i)$.

**Filtre gaussien (annexe 3.2)**

Les valeurs du rayon en entrée du filtre moyenneur sont mémorisées dans un vecteur $S$ de N éléments ; celles en sortie sont mémorisées dans un vecteur $T$ de N éléments.

Les 3 premiers et derniers éléments du vecteur $(S(1), S(2), \dots, S(N))$ sont recopiés dans le vecteur $(T(1), T(2), \dots, T(N))$.

Les calculs sont faits sur toutes les valeurs du vecteur à l'exception des bords qui sont recopiés. Les opérations effectuées sont les suivantes :

- Calcul de la convolution entre les points du vecteur et les coefficients du filtre

La formule de la convolution est $T(i) = S * G(i) = \sum_{k=-3}^{3} S(i-k) * G(k)$, une fenêtre de 7 éléments est utilisée.

#### 5.3.4.3 Algorithme codé en VHDL

L'algorithme codé en VHDL est détaillé dans l'annexe 4.

**Filtre moyenneur (annexe 4.1)**

De même que pour le code sur MATLAB, les valeurs du rayon en entrée du filtre moyenneur sont mémorisées dans un vecteur $R$ de N éléments ; celles en sortie sont mémorisées dans un vecteur $S$ de N éléments.

Les 3 premiers et derniers éléments du vecteur $(R(0), R(1), \dots, R(N-1))$ sont recopiés dans le vecteur $(S(0), S(1), \dots, S(N-1))$.

- *Arrondissement du résultat*

L'algorithme de base a été modifié pour répondre aux contraintes liées au FPGA. En effet les valeurs utilisées sur le FPGA sont des nombres entiers. Cela implique que lorsque le résultat est un nombre décimal, le FPGA n'enregistrera que la partie entière du résultat dans le nouveau vecteur. Exemple : 82,6 sera enregistré comme un résultat égal à 82.

Il est évident que cela entraînera des modifications significatives dans le résultat final. La méthode adoptée est donc d'arrondir le résultat à l'entier supérieur si la décimale est supérieure à 5 (pour un nombre à un chiffre après la virgule). Les valeurs du vecteur sont multipliées par 10 pour que le FPGA travaille avec des entiers et pour garder de manière virtuelle le chiffre après la virgule. Ainsi une moyenne égale à 94,3 sera vue par le FPGA comme un entier égal à 943. Nous appellerons cette valeur le résultat précis.

Il s'agit ensuite de comparer le résultat décimal au résultat entier trouvé par le FPGA. Pour cela la moyenne est calculée puis est multipliée par 10. Si l'on reprend l'exemple précédent, la moyenne calculée par le FPGA sera égale à 94, puis à 940 lorsqu'elle sera multipliée par 10. Nous appellerons cette valeur le résultat imprécis. Pour savoir si le résultat doit être arrondi à l'entier supérieur ou non, la différence entre le résultat précis et le résultat imprécis est calculée. Si cette différence, appelée précision dans l'algorithme, est supérieure ou égale à 5, le résultat sera arrondi à l'entier supérieur. Sinon nous garderons le résultat trouvé par le FPGA. Le résultat enregistré dans $S(i)$ sera, selon notre exemple, 94.

- *Optimisation du code VHDL*

L'algorithme a été optimisé pour diminuer le temps de calcul du FPGA. Les multiplications ou divisions sont des opérations lourdes pour le FPGA. Pour simplifier les calculs, les variables précédemment multipliées ou divisées par $10^n$ le seront par $2^n$. Le choix de la puissance est fait selon deux critères : la valeur de $2^n$ doit être supérieure à celle de la puissance de 10 et la valeur choisie sera la valeur la plus proche de la puissance de 10. Par exemple, pour une multiplication par 10, la variable sera multipliée par 16, c'est-à-dire $2^4$.

Ci-dessous un extrait de l'algorithme contenant les multiplications par 10 et un extrait de l'algorithme contenant les multiplications par $2^4$.

Algorithme travaillant avec des multiplications par 10

```
// Calcul des moyennes de chaque côté du point i
A = abs((R(i-3) + R(i-2) + R(i-1))*10/3)
B = abs((R(i+1) + R(i+2) + R(i+3))*10/3)
Diff_mean = A – B
```

Algorithme optimisé travaillant avec des multiplications de puissances de 2

```
// Calcul des moyennes de chaque côté du point i
A = abs((R(i-3) + R(i-2) + R(i-1))*16/3)
B = abs((R(i+1) + R(i+2) + R(i+3))*16/3)
Diff_mean = A – B
```

## Filtre gaussien (annexe 4.2)

De même que pour le code MATLAB, les valeurs du rayon en entrée du filtre moyenneur sont mémorisées dans un vecteur $S$ de N éléments ; celles en sortie sont mémorisées dans un vecteur $T$ de N éléments.

Les 3 premiers et derniers éléments du vecteur $(S(0), S(1), \dots, S(N-1))$ sont recopiés dans le vecteur $(T(0), T(1), \dots, T(N-1))$.

- *Filtre gaussien*

La formule du filtre est $G(i) = \frac{1}{\sigma} * e^{-\frac{i^2}{\sigma^2}}$ avec σ = 5.

Le filtre gaussien est le suivant :

| G(0) | G(1) | G(2) | G(3) | G(4) | G(5) | G(6) |
|---|---|---|---|---|---|---|
| 0.1395352 | 0.1704287 | 0.1921578 | 0.2 | 0.1921578 | 0.1704287 | 0.1395352 |

**Tableau 4 : Coefficients du filtre gaussien**

Le filtre gaussien normalisé, divisé par la somme de ses coefficients égale à 1.2042434, est le suivant :

| G(0) | G(1) | G(2) | G(3) | G(4) | G(5) | G(6) |
|---|---|---|---|---|---|---|
| 0.1158696 | 0.1415235 | 0.1595673 | 0.1660793 | 0.1595673 | 0.1415235 | 0.1158696 |

**Tableau 5 : Coefficients du filtre gaussien normalisé**

Pour travailler avec des entiers, les coefficients du filtre sont multipliés par $10^7$, ce qui donne le filtre suivant :

| G(0) | G(1) | G(2) | G(3) | G(4) | G(5) | G(6) |
|---|---|---|---|---|---|---|
| 1158696 | 1415235 | 1595673 | 1660793 | 1595673 | 1415235 | 1158696 |

**Tableau 6 : Coefficients du filtre gaussien normalisé multipliés par $10^7$**

Pour des questions d'optimisation et puisque le filtre est symétrique, seule la première partie de celui-ci sera définie dans le code. Le filtre final est donc le suivant :

| G(3) | G(2) | G(1) | G(0) |
|---|---|---|---|
| 1158696 | 1415235 | 1595673 | 1660793 |

**Tableau 7 : Coefficients du filtre gaussien normalisé optimisé**

- *Opérations effectuées pour le calcul du filtre*

La formule de la convolution est $T(i) = S * G(i) = \sum_{k=-3}^{3} S(i-k) * G(k)$, une fenêtre de 7 éléments est utilisée. Pour répondre aux contraintes du FPGA, les coefficients du filtre dans la formule correspondront aux coefficients suivants :

| Coefficient de la formule | G(-3) | G(-2) | G(-1) | G(0) | G(1) | G(2) | G(3) |
|---|---|---|---|---|---|---|---|
| Coefficient dans le code VHDL | G(0) | G(1) | G(2) | G(3) | G(4) | G(5) | G(6) |
| Coefficient dans le code VHDL optimisé | G(3) | G(2) | G(1) | G(0) | G(1) | G(2) | G(3) |

**Tableau 8 : Coefficients du filtre gaussien en VHDL**

La convolution sera développée dans le code comme ci-dessous.
(S(i+3)+ S(i-3))*G (3) + (S(i+2)+ S(i-2))*G (2) + (S(i+1)+ S(i-1))*G (1) + S(i)*G (0)

- *Arrondissement du résultat*

De même que pour le filtre précédent, l'algorithme de base a été modifié pour répondre aux contraintes liées au FPGA concernant les nombres entiers. Le filtre a déjà été multiplié par $10^7$, le résultat sera ensuite divisé par $10^7$ pour retrouver un résultat correct.

De même que précédemment, les résultats précis et imprécis sont calculés pour ensuite arrondir le résultat au nombre entier le plus proche du nombre décimal. Pour calculer le résultat précis, il suffit de diviser le résultat de la convolution par $10^6$. Pour calculer le résultat imprécis, le résultat de la convolution est divisé par $10^7$ puis multiplié par 10. Ainsi le chiffre après la virgule sera remplacé par un 0. Soit le résultat de la convolution égal à 107.8, le résultat précis sera égal à 1078 et le résultat imprécis sera égal à 1070.

Pour savoir si le résultat doit être arrondi à l'entier supérieur ou non, la différence entre le résultat précis et le résultat imprécis est calculée. Si cette différence, appelée précision dans l'algorithme, est supérieure ou égale à 5, le résultat sera arrondi à l'entier supérieur. Sinon nous garderons le résultat trouvé par le FPGA. Le résultat enregistré dans T(i) sera, selon notre exemple, 108.

- *Optimisation du code VHDL*

L'algorithme a été optimisé pour diminuer le temps de calcul du FPGA. Les multiplications ou divisions sont des opérations lourdes pour le FPGA. Pour simplifier les calculs, les variables précédemment multipliées ou divisées par $10^n$ le seront par $2^n$. Le choix de la puissance est fait selon deux critères : la valeur de $2^n$ doit être supérieure à celle de la puissance de 10 et la valeur choisie sera la valeur la plus proche de la puissance de 10. Par exemple, pour une multiplication par 10, la variable sera multipliée par 16, c'est-à-dire $2^4$.

Ci-dessous un extrait de l'algorithme contenant les divisions par $10^7$ et un extrait de l'algorithme contenant les divisions par $2^n$.

Algorithme travaillant avec des multiplications par 10

```
// Résultat arrondi à l'entier supérieur ou non
Resultat_precis = ((S(i+3)+ S(i-3))*G (3) + (S(i+2) + S(i-2))*G (2)
+ (S(i+1) + S(i-1))*G (1) + S(i)*G (0))/1000000
Resultat_imprecis = (((S(i+3)+ S(i-3))*G (3) +(S(i+2)+ S(i-2))*G (2)
+ (S(i+1) +S(i-1))*G (1) + S(i)*G (0) )/10000000)*10
Precision = Resultat_precis – Resultat_imprecis
```

Algorithme optimisé travaillant avec des multiplications de puissances de 2

**$2^4$ = 16**
**$2^{20}$ = 1048576**
**$2^{24}$ = 16777216**

```
// Résultat arrondi à l'entier supérieur ou non
Resultat_precis = ((S(i+3)+ S(i-3))*G (3) + (S(i+2) + S(i-2))*G (2)
+ (S(i+1)+ S(i-1))*G (1) + S(i)*G (0))/ 1048576
Resultat_imprecis = (((S(i+3)+ S(i-3))*G (3) +(S(i+2)+ S(i-2))*G (2)
+ (S(i+1) +S(i-1))*G (1) + S(i)*G (0) ) /16777216)*16
Precision = Resultat_precis – Resultat_imprecis
```

## 5.4 Test

Les différents tests ont été faits pour tester les blocs codés ou en VHDL ou sur MATLAB.

Le premier test effectué a eu pour but de vérifier le bon fonctionnement des blocs de filtrage en VHDL. Il a d'abord été testé avec un nombre limité de valeurs de rayon. Ce test a permis de vérifier le bon fonctionnement du bloc moyenneur et du bloc du filtre gaussien.

Par la suite, ces blocs ont été testés avec des valeurs obtenues lors d'un projet antérieur. Les valeurs en fin de traitement d'image avant et après filtrage ont été reprises de ce projet. Les valeurs avant filtrage ont été utilisées pour faire une comparaison des valeurs obtenues après filtrage en VHDL. Cela a permis de vérifier l'efficacité des blocs VHDL et de les valider.

Pour finir, les images d'une vidéo faite à l'UFPR par Renan Wille ont été utilisées. La vidéo contient environ 95 images par seconde. Seule la partie intéressante de la vidéo a été gardée et seulement une image sur 5 a été utilisée ; cela permet d'observer le changement de taille de la pupille. Cette vidéo a permis de tester les blocs de traitement d'image codés par Pedro Pitt ainsi que les blocs de filtrage avec des images réelles. Les temps de simulation étant longs, les blocs de traitement d'image ont été testés en envoyant une seule image par simulation. Après avoir simulé toutes les images, soit un total de 42 images, les valeurs résultantes ont été introduites dans les blocs de filtrage.

Le code MATLAB a été testé en utilisant les images de la vidéo. Les résultats obtenus sur MATLAB et en VHDL avec le test précédent ont été comparés.

## 5.5 Validation

La validation des tests se fait par la comparaison des résultats VHDL et MATLAB. Les résultats obtenus sur MATLAB sont considérés comme précis. Ils seront donc considérés comme des

« valeurs de référence » permettant d'avoir un regard critique sur le travail effectué en VHDL. Ces valeurs serviront à valider les résultats obtenus en VHDL et donc à valider le code écrit en VHDL.

La comparaison des résultats se fait par l'intermédiaire des courbes suivantes :

- Comparaison des valeurs en entrée des blocs de filtrage, obtenues en VHDL et sur MATLAB
- Comparaison des valeurs en sortie des blocs de filtrage, obtenues en VHDL et sur MATLAB
- Comparaison des valeurs en entrée et en sortie des blocs de filtrage pour le VHDL
- Comparaison des valeurs en entrée et en sortie des blocs de filtrage pour MATLAB
- Erreur obtenue entre les valeurs de référence (MATLAB) et les valeurs VHDL en entrée des blocs de filtrage

  La formule de l'erreur est la suivante :

$$Erreur_{initiale} = valeurs_{initiales}\ MATLAB - valeurs_{initiales}\ VHDL \quad \textbf{(8)}$$

- Erreur obtenue entre les valeurs de référence (MATLAB) et les valeurs VHDL en sortie des blocs de filtrage

  La formule de l'erreur est la suivante :

$$Erreur_{finale} = valeurs_{finales}\ MATLAB - valeurs_{finales}\ VHDL \quad \textbf{(9)}$$

La validation se fait aussi à l'aide du pourcentage d'erreur moyen sur les valeurs d'entrée et de sortie des blocs de filtrage. Le pourcentage d'erreur moyen se calcule à l'aide de la formule (10) ; la valeur théorique correspond aux valeurs de référence et la valeur expérimentale correspond aux valeurs obtenues en VHDL. Pour calculer le pourcentage d'erreur moyen, il suffit de calculer la moyenne des pourcentages d'erreur obtenus pour chaque valeur.

$$Pourcentage\ d'erreur\ = \frac{|valeur\ théorique - valeur\ expérimentale|}{valeur\ théorique} * 100 \quad \textbf{(10)}$$

## 5.6 Résultats

Seront présentés dans ce paragraphe, les résultats obtenus lors du test de la partie de filtrage et les résultats obtenus lors du test du projet complet.

Le code VHDL a été simulé à l'aide de ModelSim. Le projet n'a pas été testé sur le FPGA par manque de temps. Les images testées proviennent de vidéos enregistrées à l'UFPR ; elles ont été utilisées lors des simulations du projet complet. Les valeurs utilisées pour ne tester que la partie de filtrage proviennent d'un projet antérieur réalisé à l'UFPR.

### *5.6.1 Test de la partie filtrage*

Pour une meilleure observation des courbes, seule la partie intéressante a été gardée.

Cette comparaison permet de visualiser l'écart entre les valeurs dites de référence et les valeurs obtenues en VHDL.

Le FPGA a été programmé pour recevoir des valeurs entières en entrée des blocs de filtrage. Les valeurs de référence en entrée du filtrage ont donc été converties en valeurs entières selon le principe suivant : si le premier chiffre après la virgule est supérieur ou égal à 5, le nombre est arrondi à l'entier supérieur. Ceci peut être observé sur la figure 29 du haut. Les valeurs VHDL (en

rouge) sont des nombres entiers, d'où la présence de paliers. La courbe de référence est lisse, car les valeurs de référence sont des nombres réels. L'écart entre la valeur de référence et la valeur d'entrée sera au maximum égal à 0,5.

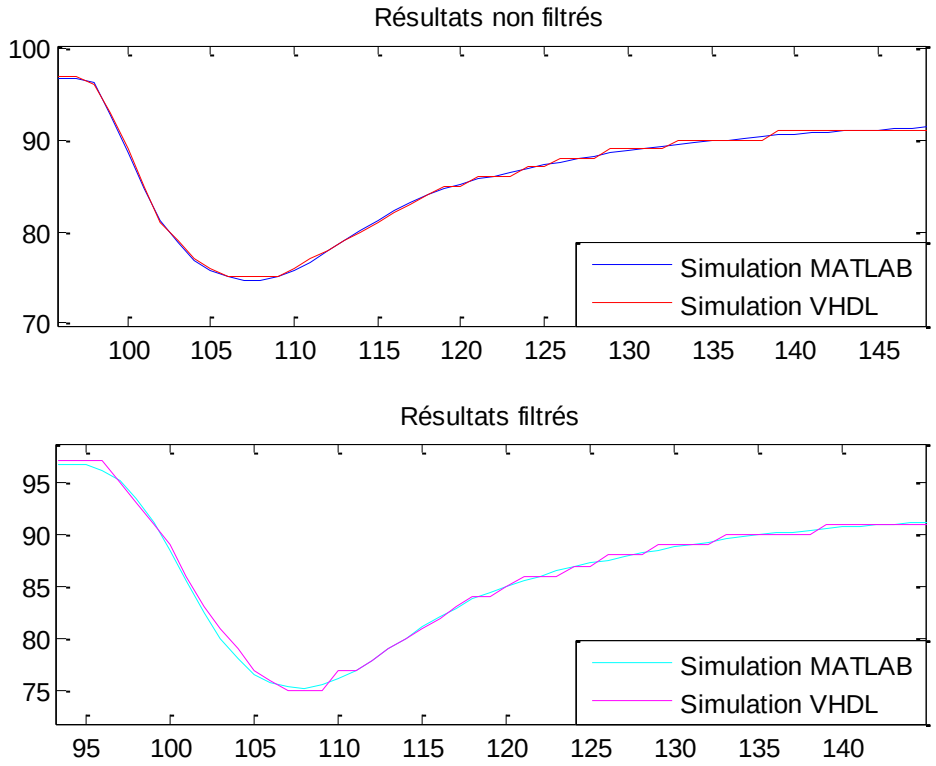



**Figure 29 : Comparaison entre les mesures de référence et les mesures VHDL**

Les résultats obtenus en sortie du filtre ne diffèrent que très peu des valeurs de référence. Ceci est très encourageant quant à l'efficacité de la partie de filtrage. Les paliers correspondant au passage d'un entier à un autre sont facilement observables.

La comparaison suivante est une comparaison entre les valeurs d'entrée et de sortie de la partie de filtrage. Ceci permet de vérifier le bon fonctionnement des filtres.

La forme des courbes des deux graphiques est très ressemblante. La seule différence flagrante est le changement d'un entier à un autre représenté par les paliers sur la courbe du bas de la figure 30.

Il est facile d'observer qu'il n'y a pas eu d'erreur de détection de la pupille (à cause d'un œil fermé par exemple), car les courbes non filtrées ne présentent pas de point n'appartenant pas à la courbe. Nous pouvons donc en déduire que le filtre moyenneur n'a pas eu de grande influence sur la courbe finale.

La courbe a été lissée à l'aide du filtre gaussien. Ceci peut être observé de l'image 95 à l'image 110, ce qui correspond, en temps, de 1 seconde à 1,16 seconde.

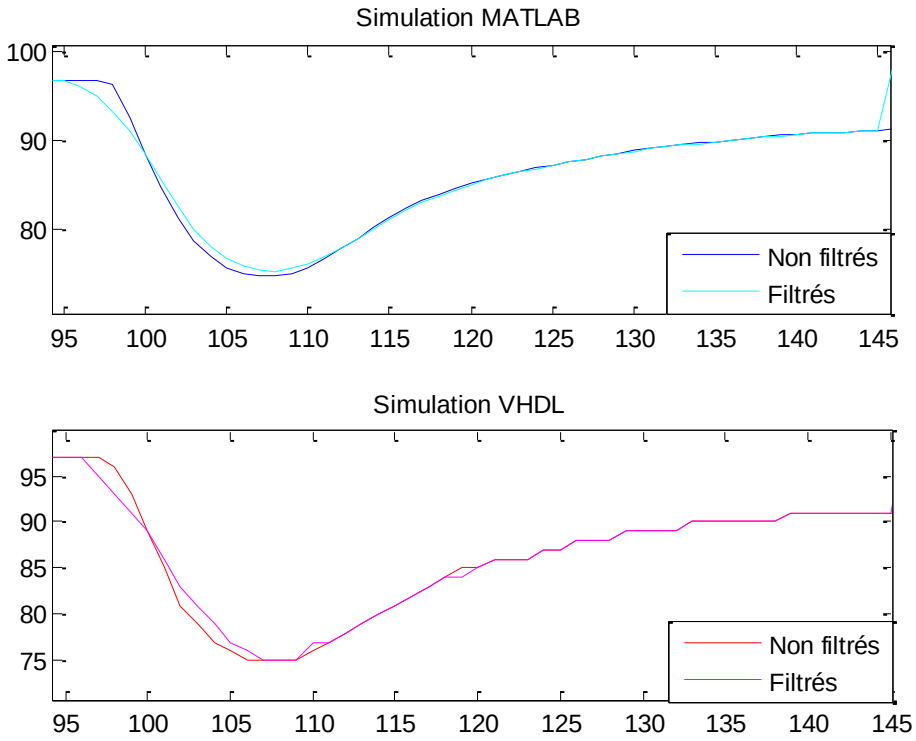



**Figure 30 : Comparaison entre les valeurs d'entrée et de sortie**

Les courbes d'erreur servent à vérifier la précision du code VHDL. En effet s'il n'est pas assez précis, cela signifie que travailler avec des nombres entiers n'est pas intéressant.

Les deux courbes d'erreur en entrée et en sortie des filtres sont très ressemblantes. Les valeurs sont comprises entre -0,5 et +0,5 pixel.

Cela signifie que l'arrondissement à l'entier supérieur des valeurs d'entrée ne modifiera pas de façon significative les valeurs de sortie.

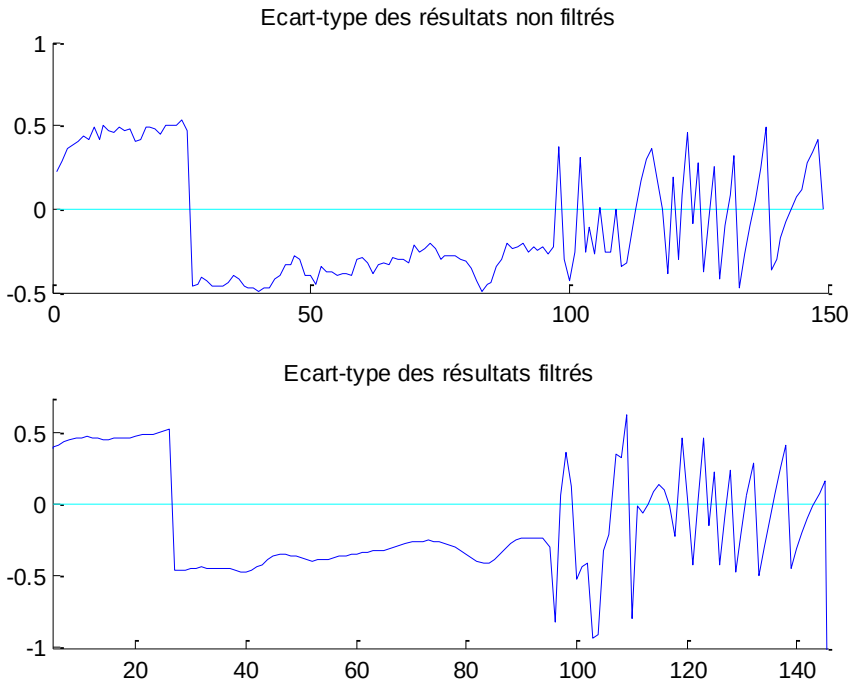



**Figure 31 : Erreur due au FPGA**

Pour avoir une vision plus claire de la précision, le pourcentage d'erreur moyen a été calculé sur les valeurs d'entrée et de sortie. Nous disposons de deux séries de mesures provenant d'un projet de l'UFPR. Le tableau suivant récapitule les résultats obtenus.

| | 1ère série de mesures | 2ème série de mesures |
|---|---|---|
| Pourcentage d'erreur moyen des valeurs non filtrées (en entrée de la partie filtrage) | 0.0023 % | 0.0003876 % |
| Pourcentage d'erreur moyen des valeurs filtrées (en sortie de la partie filtrage) | 0.0075 % | 0.0053 % |

**Tableau 9 : Pourcentage d'erreur moyen**

Que ce soit avant ou après filtrage, le pourcentage d'erreur est très faible. Tous ces résultats nous permettent donc de valider la partie filtrage codée en VHDL.

### *5.6.2 Test du projet complet*

Les résultats suivants permettent de valider ou non le projet complet, c'est-à-dire le projet contenant les parties de pré-traitement des images, de traitement d'image ainsi que la partie de filtrage. Les courbes du haut de la figure 32 représentent les résultats obtenus après l'application des parties de pré-traitement et de traitement des images. La forme des courbes du paragraphe précédent est toujours présente. Cela correspond à la variation de la taille de la pupille et, plus précisément, au moment où la pupille va se contracter à cause du flash de lumière puis se dilater de nouveau.

Il est facile d'observer que le filtrage a un effet important sur l'allure des courbes. En effet, si l'on compare les courbes du haut et du bas de la figure 32, on s'aperçoit que le filtrage élimine les points éloignés de la courbe (entre les images 45 et 55 de la figure 32) et la lisse pour obtenir des valeurs exploitables.

De plus, nous pouvons noter que les bords des valeurs MATLAB et des valeurs VHDL, c'est-à-dire les 3 premières et dernières valeurs du vecteur, sont différents. Ces valeurs ne sont pas à prendre en compte, car elles ont simplement été recopiées.

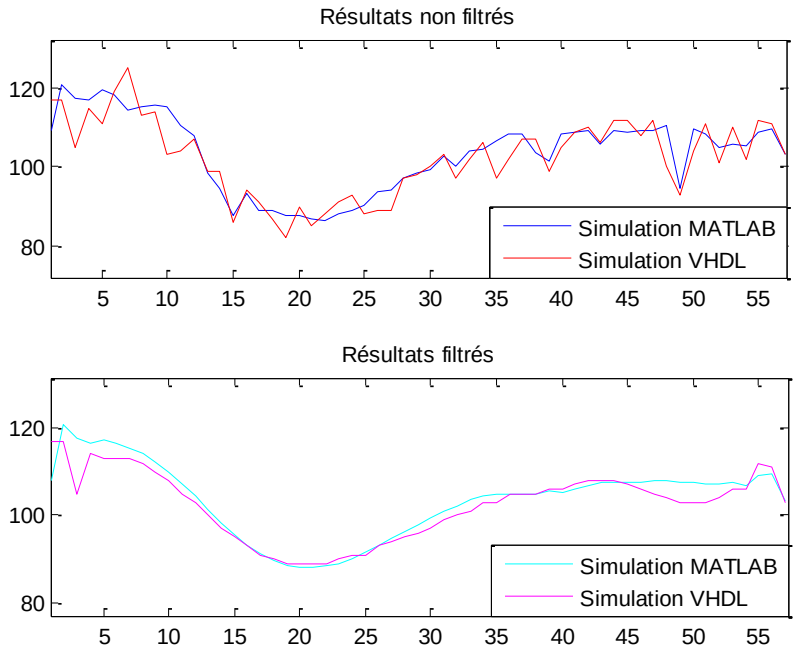



**Figure 32 : Comparaison entre les simulations MATLAB et VHDL**

La figure suivante est une comparaison des valeurs brutes, non filtrées, et des valeurs finales, filtrées. Elle permet d'avoir une idée précise de l'efficacité des algorithmes choisis sur MATLAB et en VHDL. Dans les deux cas, nous nous apercevons que les valeurs obtenues à l'aide de l'algorithme de traitement d'image sont plus ou moins précises. Cela nous permet aussi de constater, une fois de plus, l'efficacité de la partie de filtrage et ainsi de la valider.

Une autre observation peut être faite sur la figure 33 : l'algorithme utilisé sur MATLAB donne des résultats plus précis qu'en VHDL. Cela est simplement du au fait que l'on travaille avec des entiers en VHDL et avec des nombres réels sur MATLAB. De même pour les résultats filtrés.

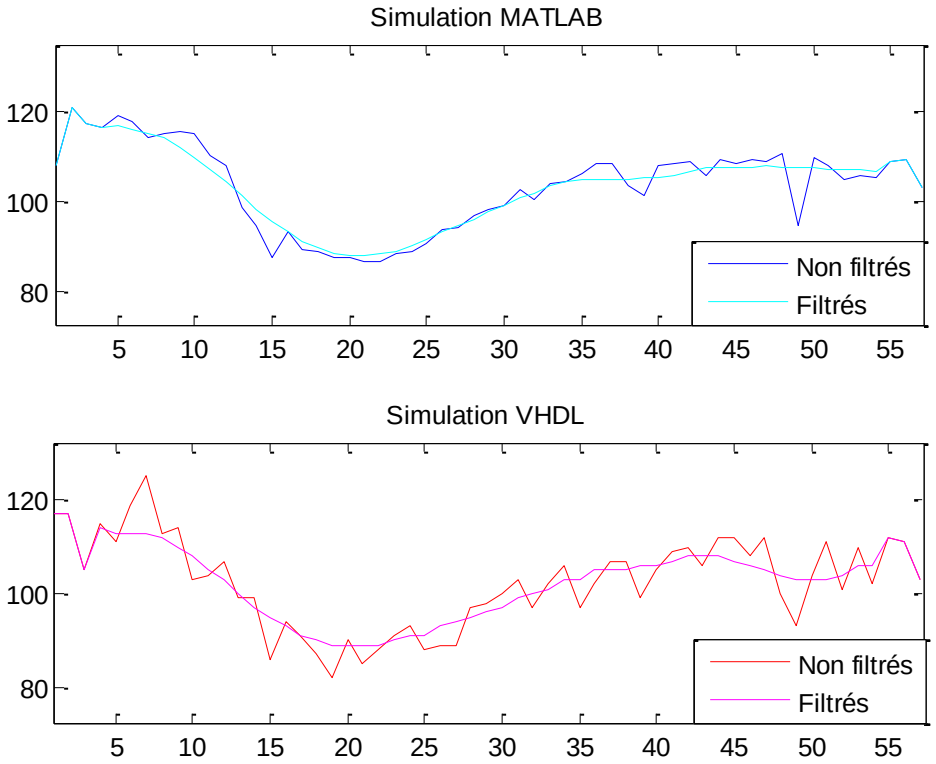



**Figure 33 : Comparaison entre les valeurs d'entrée et de sortie de la partie de filtrage**

Les erreurs des données non filtrées calculées entre les résultats MATLAB et VHDL (figure 34) sont plus élevées que celles obtenues dans le test de la partie de filtrage. Pour le filtrage, nous avons arrondi les résultats provenant d'un autre projet pour les envoyer en entrée du filtre. Dans le cas présent, l'algorithme de traitement d'image a calculé lui-même la taille de la pupille. Cela implique inévitablement des changements au niveau des résultats d'entrée de la partie de filtrage. Ils ne seront plus aussi précis que précédemment.

Les erreurs obtenues sur les données filtrées sont très satisfaisantes, car elles ont été réduites considérablement par le filtrage. Ceci est très encourageant et permet de se fier aux résultats obtenus.

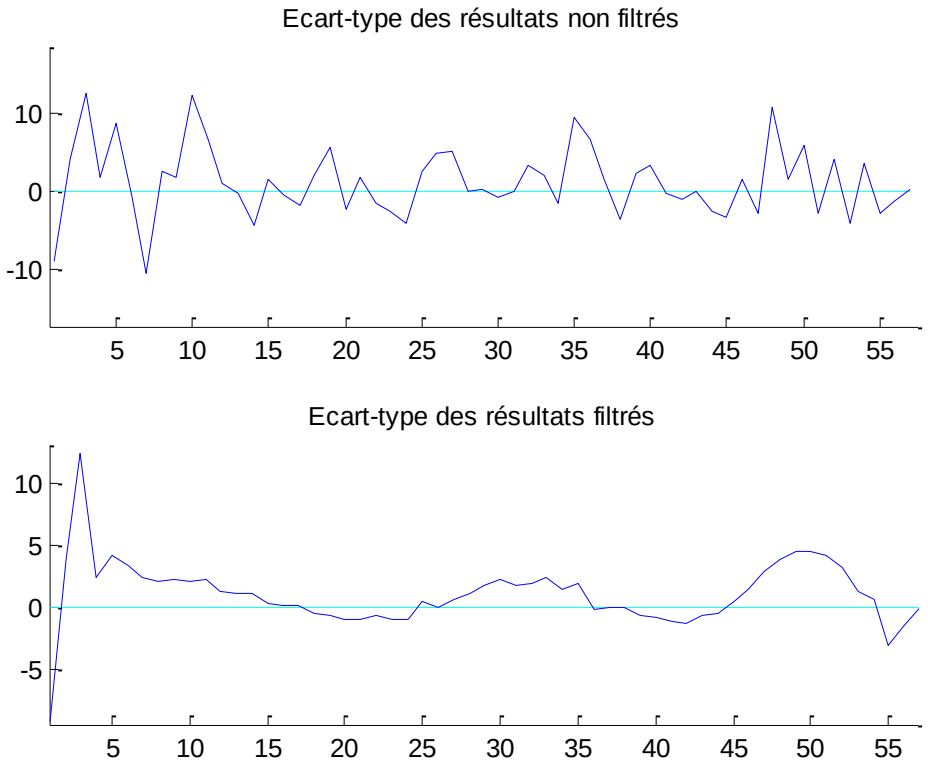



**Figure 34 : Erreur due au FPGA**

Le tableau suivant donne les valeurs minimale et maximale de l'erreur avant et après filtrage. L'unité de mesure est le pixel. L'erreur n'a pas été calculée pour les valeurs aux bords, car elles sont recopiées lors du filtrage et ne sont pas modifiées.

| | Avant filtrage | Après filtrage |
|---|---|---|
| Valeur maximale de l'erreur | 12,2406 | 4,5785 |
| Valeur minimale de l'erreur | -10,4867 | -1,2170 |

**Tableau 10 : Extrema**

La différence entre l'erreur avant et après le filtrage est flagrante lorsqu'on observe les valeurs minimum et maximum. Avant filtrage, nous obtenons une différence de 22,7273 pixels entre le maximum et le minimum de la courbe. Après filtrage nous obtenons une différence de 5,7955 pixels. Le filtrage l'a divisé par 4.

Pour avoir une vision plus claire de la précision, le pourcentage d'erreur moyen a été calculé sur les valeurs d'entrée et de sortie. Le tableau suivant récapitule les résultats obtenus.

| | |
|---|---|
| Pourcentage d'erreur moyen des valeurs non filtrées (en entrée de la partie filtrage) | 0,0592 % |
| Pourcentage d'erreur moyen des valeurs filtrées (en sortie de la partie filtrage) | 0,0201 % |

**Tableau 11 : Pourcentage d'erreur moyen**

Lors de la simulation du projet complet, le pourcentage d'erreur en entrée du filtrage est plus élevé que précédemment et celui en sortie est toujours aussi bas. La valeur la plus importante reste celle du pourcentage d'erreur en sortie du projet, car ce sont les valeurs obtenues après filtrage qui nous intéressent. Cela est très encourageant quant à la suite du projet. Ces résultats nous permettent de valider l'ensemble du projet en simulation.

# PARTIE N°6. CONCLUSION

Lors du développement de ce projet, plusieurs étapes clés ont été menées à bien. La première étant la validation du filtrage des données. L'efficacité des programmes VHDL a été vérifiée en comparant ces résultats avec les résultats obtenus lors d'un projet précédent. Les résultats étant concluants, l'étape suivante fut traitée : valider le projet complet de traitement d'image et du signal. Chaque image de la vidéo a été analysée afin de trouver la taille de la pupille, puis les valeurs obtenues ont été filtrées. Les résultats obtenus à l'aide des programmes VHDL ont été comparés à ceux obtenus sur MATLAB. En fin de projet, la réaction de la pupille à une stimulation lumineuse a pu être observée. Les résultats obtenus par la simulation du projet sur le FPGA sont concluants et ont permis de valider le travail effectué au cours du projet. Les objectifs de simulation du projet ont été atteints. La prochaine étape sera donc l'implémentation et les tests du projet complet sur le FPGA. Le projet pourra ensuite être testé à l'aide du pupillomètre et d'une caméra fonctionnelle. Ces deux dernières étapes n'ont pas été effectuées par manque de temps ou de moyens. Les programmes de traitement d'image et du signal pourront être réutilisés dans un projet futur.

Ce Projet de Fin d'Etudes a permis l'acquisition de connaissances dans le domaine de la biométrie, domaine en plein essor. Différentes méthodes de détection de la pupille et de l'iris ainsi que l'utilisation qui peut en être faite (identification d'individu, dépistage de maladies, etc) ont été explorées avant de choisir celle qui serait utilisée, cela permettant d'avoir une idée plus large des algorithmes existants. L'électronique allié au domaine médical est très attractif et mérite que l'on s'y intéresse.

De plus, le développement du projet dans deux langages informatiques différents donne une meilleure idée des avantages et des inconvénients de chaque langage et de leur utilité propre.

Cet échange à l'étranger a été l'occasion d'apprendre une nouvelle langue, le portugais, ce qui donne un atout supplémentaire lors de la recherche d'emploi.

Pour finir, ce stage a permis de constater qu'un projet peut évoluer différemment de ce qu'on avait projeté selon les moyens à disposition.

# PARTIE N°7. GLOSSAIRE

| | |
|---|---|
| ASIC | Application-Specific Integrated Circuit |
| BDT | Boundary Detection Template |
| CIEL | Centro de Instrumentação Eletrônica |
| COPEL | COmpania Paranaense de Energia |
| DDD | Discagem Direta a Distância |
| FPGA | Field-Programmable Gate Array |
| GICS | Group of Integrated Circuits and Systems |
| GVSO | Gray Value Summing Operator |
| HDL | Hardware Description Language |
| LAMMI | Lab. Microeletrônica, Medidas e Instrumentação |
| LASICO | Lab. De Simulação, Controle e Otimização |
| LED | Light Emitting Diode |
| LIEC | Lab. Instrumentação Eletrônica e Comunicação |
| MATLAB | MATrix LABoratory |
| NFC | Near Field Communication |
| TELEBRÁS | TELEfonia do BRASil |
| TELEPAR | TELEfonia do PARaná |
| UFPR | Universidade Federal do Paraná (PR) |
| VHDL | VHSIC Hardware Description Language |
| VHSIC | Very High Speed Integrated Circuits |
| VLSI | Very-Large-Scale Integration |

**Tableau 12 : Glossaire**

# PARTIE N°8. BIBLIOGRAPHIE

[1] *Biométrie (définition, intérêt) : Définition*, http://www.vulgaris-medical.com/encyclopedie/biometrie-definition-interet-9322.html

[2] *Iris (anatomie)*, http://fr.wikipedia.org/wiki/Iris_(anatomie)

[3] *Pupille*, http://fr.wikipedia.org/wiki/Pupille

[4] *Apresentação*, http://www.ufpr.br/portalufpr/a-universidade-institucional/

[5] *Departamento de Engenharia Elétrica – Universidade Federal do Paraná – História*, http://www.eletrica.ufpr.br/p/departamento:historia

[6] Vitor Yano, Alessandro Zimmer, Giselle Ferrari, "Utilização de pupilometria dinâmica para identificação pessoal baseada em reflexos humanos", *Universidade Federal do Paraná, Departamento de Engenharia Elétrica*.

[7] Weiqi, Yuan, Lu. Xu, Zhonghua, Lin, "A novel iris localization algorithm based on the gray distributions of eye images", *Computer Vision Group, Shenyang University of Technology*.

[8] G. L. Ferrari, "Pupilometria Dinâmica: Aplicação na Detecção e Avaliação da Neuropatia Autonômica Diabética e Estudo da Correlação entre a Resposta Temporal da Pupila ao Estímulo Visual e a Glicemia", Tese de Doutorado, *Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial*, UTFPR, Curitiba, Fev. 2008.

# PARTIE N°9. ANNEXES

## 9.1 Annexe 1 : Calcul des paramètres d'un cercle connaissant trois points lui appartenant

Ci-dessous, voici le détail du calcul effectué lors du traitement d'image. Il a servi à l'obtention du centre et du rayon du cercle correspondant à la frontière pupille-iris.

Les trois points appartenant au cercle seront appelés $P_L$, $P_R$ et $P_D$ (voir figure 17). Pour chaque point, nous pouvons écrire l'équation d'un cercle, nous obtenons les équations suivantes :

$$\begin{cases} (x_L - x_0)^2 + (y_L - y_0)^2 = R^2 & \textbf{(1)} \\ (x_R - x_0)^2 + (y_R - y_0)^2 = R^2 & \textbf{(2)} \\ (x_D - x_0)^2 + (y_D - y_0)^2 = R^2 & \textbf{(3)} \end{cases}$$

La coordonnée horizontale du centre du cercle est simple à trouver, car nous avons les coordonnées horizontales de deux points de part et d'autre du cercle. Nous obtenons l'équation suivante :

$$\boxed{x_0 = \frac{x_L + x_R}{2}}$$

Pour trouver la coordonnée verticale du centre du cercle, nous allons soustraire l'équation (3) à l'équation (2). Nous remplacerons également la coordonnée $x_0$ par le résultat trouvé ci-dessus.

$$(3) - (2) \Leftrightarrow (x_D - x_0)^2 + (y_D - y_0)^2 - (x_R - x_0)^2 - (y_R - y_0)^2 = 0$$
$$\Leftrightarrow {x_D}^2 - 2 * x_D * x_0 + {x_0}^2 + {y_D}^2 - 2 * y_D * y_0 + {y_0}^2 - {x_R}^2 + 2 * x_R * x_0 - {x_0}^2 - {y_R}^2 + 2 * y_R * y_0 - {y_0}^2 = 0$$
$$\Leftrightarrow {x_D}^2 - 2 * x_D * \left(\frac{x_L + x_R}{2}\right) + {y_D}^2 - 2 * y_D * y_0 - {x_R}^2 + 2 * x_R * \left(\frac{x_L + x_R}{2}\right) - {y_R}^2 + 2 * y_R * y_0 = 0$$
$$\Leftrightarrow {x_D}^2 + {y_D}^2 - {x_R}^2 - {y_R}^2 - x_D * (x_L + x_R) + x_R * (x_L + x_R) = 2 * y_D * y_0 - 2 * y_R * y_0$$
$$\Leftrightarrow {x_D}^2 + {y_D}^2 - {x_R}^2 - {y_R}^2 - x_D * (x_L + x_R) + x_R * x_L + {x_R}^2 = 2 * y_0 * (y_D - y_R)$$
$$\Leftrightarrow {x_D}^2 + {y_D}^2 - {y_R}^2 - x_D * (x_L + x_R) + x_R * x_L = 2 * y_0 * (y_D - y_R)$$

$$\boxed{\Rightarrow y_0 = \frac{{x_D}^2 + {y_D}^2 - x_D * (x_L + x_R) - {y_R}^2 + x_R * x_L}{2 * (y_D - y_R)}}$$

Nous utiliserons l'équation (3) afin de calculer le rayon :

$$R = \sqrt{(x_D - x_0)^2 + (y_D - y_0)^2}$$

## 9.2 Annexe 2 : Algorithmes de traitement d'image sur MATLAB

### 9.2.1 Pré-traitement

```
Début
fonction preprocessing prenant I1 (image d'entré)e et r  en entrée et I2 (image de sortie)  en sortie

[height, width] = taille de I1
I1 = I2

Pour j allant de r+1 à width-r
   Pour i allant de r+1 à height-r
      // Calcul de Vm et Vb
      Vm = (I1(i,j) + I1(i,j-1) + I1(i+1,j) + I1(i,j+1) + I1(i-1,j))/5
            Vb = (I1(i,j-r) + I1(i+r,j) + I1(i,j+r) + I1(i-r,j))/4
      Si (Vm > 2*Vb) alors
                        // Ligne 0
                     Pour d allant de 1 à r
                        I2(i,j+(d-7)) = Vb
                     Fin pour
                     // Ligne -1 et +1
                     Pour d allant de 1 à r-2
                           I2(i-1,j+(d-6)) = Vb
                           I2(i+1,j+(d-6)) = Vb
                     Fin pour
                     // Ligne -2 et +2
                     Pour d allant de 1 à r-2
                           I2(i-2,j+(d-6)) = Vb
                           I2(i+2,j+(d-6)) = Vb
                     Fin pour
                     // Ligne -3 et +3
                     Pour d allant de 1 à r-4
                           I2(i-3,j+(d-5)) = Vb
                           I2(i+3,j+(d-5)) = Vb
                     Fin pour
                     // Ligne -4 et +4
                     Pour d allant de 1 à r-6
                           I2(i-4,j+(d-4)) = Vb
                           I2(i+4,j+(d-4)) = Vb
                     Fin pour
                     // Ligne -5 et +5
                     Pour d allant de 1 à r-8
                           I2(i-5,j+(d-3)) = Vb
                           I2(i+5,j+(d-3)) = Vb
                     Fin pour
                     // Ligne -6 et +6
                     I2(i-6,j-1) = Vb
                     I2(i+6,j-1) = Vb
      Fin si
   Fin pour
```

Fin pour
Fin

### 9.2.2 Extraction de la valeur du rayon

Début
fonction radius_extraction prenant *Img* en entrée et le *rayon, les coordonnées du centre* et *la valeur de gris de l'image* en sortie

[height, width] = taille de Img
// Algorithme de Weiqi : Trouver un point dans la pupille
// Paramètres de la fenêtre de recherche
m = 25
n = 5
Utilisation de pupil_point.m pour trouver la coordonnée horizontale x0
// Paramètres de la fenêtre de recherche
m = 5
n = 25
Utilisation de pupil_point.m pour trouver la coordonnée verticale y0

// Algorithme de Weiqi : Trouver les limites de l'iris et la pupille
m = 25
n = 5

v0 = Img(y0,x0)
v = 2*v0 // Valeur de gris de référence

// Domaine de recherche gauche : [(m-1)/2;x0-1]
Dif_m = 0
yL = y0
xL = x0-1
Pour j allant de (m-1)/2+1 à x0-1
  Utilisation de BDT.m avec yL fixé pour calculer les valeurs du bord et de gris
  Si (Val_gris ≤ V) alors
    Si (Val_bord > Dif_m) alors
      Dif_m = val_bord
      xL = j
    Fin si
  Fin si
Fin pour

// Domaine de recherche droit : [x0+1;width-(m+1)/2]
Dif_m = 0
yR = y0
xR = x0+1
Pour j allant de x0+1 à width-(m+1)/2
  Utilisation de BDT.m avec yR fixé pour calculer les valeurs du bord et de gris
  Si (Val_gris ≤ V) alors
    Si (Val_bord > Dif_m) alors
      Dif_m = val_bord
      xR = j

```
      Fin si
   Fin si
Fin pour

// Domaine de recherche bas : [y0+1;height-(n-1)/2]
Dif_m = 0
xD = x0
yD = y0+1
Pour i allant de y0+1 à height-(n-1)/2
   Utilisation de BDT.m avec xD fixé pour calculer les valeurs du bord et de gris
   Si (Val_gris ≤ V) alors
      Si (Val_bord > Dif_m) alors
         Dif_m = val_bord
         yD = j
      Fin si
   Fin si
Fin pour

// Calcul du rayon
xp = (xr+xl)/2
yp = (xd^2+yd^2-xd*(xr+xl)-yr^2+xr*xl)/(2*(yd-yr))
Rp = sqrt((xp-xd)^2+(yp-yd)^2)
Fin
```

#### 9.2.2.1 Recherche d'un point dans la pupille

Début
fonction pupil_point prenant *Img*, *m* et *n* (paramètres de l'aire de recherche) en entrée et *les lignes et les colonnes correspondant à une valeur minimale* en sortie

[height, width] = taille de Image_finale

```
Pour j allant de (n-1)/2+1 à width-(n-1)/2-1
   Pour i allant de (m-1)/2+1 à height-(m-1)/2-1

      Pour jj allant de j-(n-1)/2 à j+(n-1)/2
         Pour ii allant de i-(m-1)/2 à i+(m-1)/2
            S(i-(m-1)/2,j-(n-1)/2) = S(i-(m-1)/2,j-(n-1)/2) + Img(ii,jj)
         Fin pour
      Fin pour
   Fin pour
Fin pour
```

Smin = min(S)
[r,c] = ligne et colonne lorsque (Smin==S)
Fin

Ensuite les coordonnées du point dans la pupille sont calculées de la façon suivante :

$$x0 = min(c) + (max(c) - min(c))/2 \quad \textbf{(11)}$$
$$y0 = min(r) + (max(r) - min(r))/2 \quad \textbf{(12)}$$

#### 9.2.2.2 BDT

Début
fonction BDT prenant *Img*, *x* et *y* (pixel de l'image) et la *direction_dif* de la 1ère valeur déterminante et la *direction_avg* de la 2ème valeur en entrée et les deux *valeurs déterminantes* en sortie

```
// Direction horizontale
Si direction est horizontale alors
   m = 25
   n = 5
   val_ij(y,x) = 0
   val_pq(y,x) = 0
   Pour jj allant de x-(m-1)/2 à x-1
      Pour ii allant de y-(n-1)/2 à y+(n-1)/2
         val_ij(y,x) = val_ij(y,x) + Img(ii,jj)
     Fin pour
   Fin pour
   Pour q allant de x+1 à x+(m-1)/2
      Pour p allant de y-(n-1)/2 à y+(n-1)/2
         val_pq(y,x) = val_pq(y,x) + Img(p,q)
     Fin pour
   Fin pour
   val_dif = abs(val_ij(y,x)-val_pq(y,x))
   Si direction est gauche alors
      val_avg = val_pq(y,x)/(n*(m-1)/2)
   Sinon si direction est droite alors
      val_avg = val_ij(y,x)/(n*(m-1)/2)
   Sinon
      Afficher message d'erreur
   Fin si
Sinon si direction est verticale alors
   m = 5
   n = 25
   val_ij(y,x) = 0
   val_pq(y,x) = 0
   Pour jj allant de x-(n-1)/2 à x+(n-1)/2
      Pour ii allant de y-(m-1)/2 à y-1
         val_ij(y,x) = val_ij(y,x) + Img(ii,jj)
     Fin pour
   Fin pour
   Pour q allant de x-(n-1)/2 à x+(n-1)/2
      Pour p allant de y+1 à y+(m-1)/2
         val_pq(y,x) = val_pq(y,x) + Img(p,q)
     Fin pour
   Fin pour
   val_dif = abs(val_ij(y,x)-val_pq(y,x))
   val_avg = val_ij(y,x)/(n*(m-1)/2)
Sinon
   Afficher message d'erreur
Fin si
```

Fin

## 9.3 Annexe 3 : Algorithmes du filtrage sur MATLAB

### 9.3.1 Filtre moyenneur

```
Début
fonction bloc_mean prenant vect_radius en entrée et vect_radius_filtre1 en sortie

nb_img = taille de vect_radius

// Bords: on copie les 3 premiers éléments dans vect_radius_filtre1(i)
vect_radius_filtre1(1) = vect_radius(1)
vect_radius_filtre1(2) = vect_radius(2)
vect_radius_filtre1(3) = vect_radius(3)

Pour i allant de 4 à nb_img-3
   // Calcul des moyennes de chaque côté du point i
   mean_A = (vect_radius(i+1) + vect_radius(i+2) + vect_radius(i+3))/3
   mean_B = (vect_radius(i-3) + vect_radius(i-2) + vect_radius(i-1))/3
   Dif_mean = mean_A - mean_B

   Si (Dif_mean > 2) alors
      vect_radius_filtre1(i) = (mean_A + mean_B)/2
   Sinon
      vect_radius_filtre1(i) = vect_radius(i);
   Fin si
Fin pour

// Bords: on copie les 3 derniers éléments dans vect_radius_filtre1 (i)
vect_radius_filtre1(nb_img-2) = vect_radius(nb_img-2)
vect_radius_filtre1(nb_img-1) = vect_radius(nb_img-1)
vect_radius_filtre1(nb_img) = vect_radius(nb_img)
Fin
```

### 9.3.2 Filtre gaussien

```
Début
fonction bloc_gaussian_filter prenant  vect_radius_filtre1 en entrée et  vect_radius_filtre2 en sortie

nb_img = taille de vect_radius_filtre1

// Filtre gaussien
σ = 5
Pour ii allant de -3 à 3
   G(ii+4) = 1/σ * exp^(-ii²/σ²)
Fin pour
somme = somme de G
Pour ii allant de -3 à 3
```

G(ii+4) = $\frac{G(ii+4)}{somme}$
Fin pour

vect_radius_filtre2(1) = vect_radius_filtre1(1)
vect_radius_filtre2(2) = vect_radius_filtre1(2)
vect_radius_filtre2(3) = vect_radius_filtre1(3)

Pour ii allant de 4 à nb_img-3
  vect_radius_filtre2(i) = 0;
  Pour k allant de -3 à 3
    vect_radius_filtre2(i) = vect_radius_filtre2(i) + vect_radius_filtre1(i-k)*G(4+k);
  Fin pour
Fin pour
vect_radius_filtre2(nb_img-2) = vect_radius_filtre1(nb_img-2)
vect_radius_filtre2(nb_img-1) = vect_radius_filtre1(nb_img-1)
vect_radius_filtre2(nb_img) = vect_radius_filtre1(nb_img)
Fin

## 9.4 Annexe 4 : Algorithmes du filtrage en VHDL

### 9.4.1 Filtre moyenneur

9.4.1.1.1 Algorithme modifié pour travailler avec des entiers

```
Début
// Bords: on copie les 3 premiers éléments dans S(i)
S(0) = R(0)
S(1) = R(1)
S(2) = R(2)

Pour i allant de 3 à N-4
    // Initialisation
    A = 0
    B = 0
    Diff_mean = "00000000"
    Resultat_precis = 0
    Resultat_imprecis = 0
    Precision = "0000"

    // Calcul des moyennes de chaque côté du point i
    A = abs((R(i-3) + R(i-2) + R(i-1))*10/3) // Multiplication par 10 pour travailler avec des entiers
    B = abs((R(i+1) + R(i+2) + R(i+3))*10/3)
    Diff_mean = A – B

    Si Diff_mean > "00010100" alors // 20 en binaire (2 multiplié par 10)
      // Résultat arrondi à l'entier supérieur ou non
      Resultat_precis = ((R(i+1) + R(i+2) + R(i+3) + R(i-3) + R(i-2) + R(i-1)) *10) /3/2
      Resultat_imprecis = ((R(i+1) + R(i+2) + R(i+3) + R(i-3) + R(i-2) + R(i-1)) /3/2) *10
      Precision = Resultat_precis – Resultat_imprecis
```

```
        Si Precision >= "0101" alors // 5 en binaire
            S(i) = ( abs(R(i+1) + R(i+2) + R(i+3) + R(i-3) + R(i-2) + R(i-1)))/6 + 1
        Sinon
            S(i) = ( abs(R(i+1) + R(i+2) + R(i+3) + R(i-3) + R(i-2) + R(i-1)))/6
        Fin si
    Sinon
        S(i) = R(i)
    Fin si
Fin pour

// Bords: on copie les 3 derniers éléments dans S(i)
S(N-3) = R(N-3)
S(N-2) = R(N-2)
S(N-1) = R(N-1)
Fin
```

9.4.1.1.2 Algorithme du code optimisé

**$2^4$ = 16**

```
Début
// Bords: on copie les 3 premiers éléments dans S(i)
S(0) = R(0)
S(1) = R(1)
S(2) = R(2)

Pour i allant de 3 à N-4
    // Initialisation
    A = 0
    B = 0
    Diff_mean = " 0000000000"
    Resultat_precis = 0
    Resultat_imprecis = 0
    Precision = " 00000"

    // Calcul des moyennes de chaque côté du point i
    A = abs((R(i-3) + R(i-2) + R(i-1))*16/3) // Multiplication par 16 pour travailler avec des entiers
    B = abs((R(i+1) + R(i+2) + R(i+3))*16/3)
    Diff_mean = A – B

    Si Diff_mean > " 0000100000" alors // 32 en binaire (2 multiplié par 16)
        // Résultat arrondi à l'entier supérieur ou non
        Resultat_precis = ((R(i+1) + R(i+2) + R(i+3) + R(i-3) + R(i-2) + R(i-1)) *16) /3/2
        Resultat_imprecis = ((R(i+1) + R(i+2) + R(i+3) + R(i-3) + R(i-2) + R(i-1)) /3/2) *16
        Precision = Resultat_precis – Resultat_imprecis

        Si Precision >= " 01000" alors // 8 en binaire (0,5 multiplié par 16)
            S(i) = ( abs(R(i+1) + R(i+2) + R(i+3) + R(i-3) + R(i-2) + R(i-1)))/6 + 1
```

```
        Sinon
            S(i) = ( abs(R(i+1) + R(i+2) + R(i+3) + R(i-3) + R(i-2) + R(i-1)))/6
        Fin si
    Sinon
        S(i) = R(i)
    Fin si
Fin pour

// Bords: on copie les 3 derniers éléments dans S(i)
S(N-3) = R(N-3)
S(N-2) = R(N-2)
S(N-1) = R(N-1)
Fin
```

### 9.4.2 Filtre gaussien

9.4.2.1.1 Algorithme modifié pour travailler avec des entiers

```
Début
// Filtre gaussien normalisé arrondi et multiplié par 10^7 pour avoir des calculs d'entiers
G = (1158696, 1415235, 1595673, 1660793)

// Bords: on copie les 3 premiers éléments dans T(i)
T(0) = S(0)
T(1) = S(1)
T(2) = S(2)

Pour i allant de 3 à N-4
    // Initialisation
    Resultat_precis = 0
    Resultat_imprecis = 0
    Precision = "0000"

    // Résultat arrondi à l'entier supérieur ou non
    Resultat_precis = ((S(i+3)+ S(i-3))*G (3) + (S(i+2) + S(i-2))*G (2) + (S(i+1)+ S(i-1))*G (1)
    + S(i)*G (0))/1000000

    Resultat_imprecis = (((S(i+3)+ S(i-3))*G (3) +(S(i+2)+ S(i-2))*G (2) + (S(i+1) +S(i-1))*G (1)
    + S(i)*G (0) )/10000000)*10

    Precision = Resultat_precis – Resultat_imprecis

    Si Precision >= "0101" alors // 5 en binaire
        T(i) = ((S(i+3)+ S(i-3))*G (3) + (S(i+2)+ S(i-2))*G (2) + (S(i+1)+ S(i-1))*G (1)
        + S(i)*G (0))/10000000 + 1
    Sinon
        T(i) = ((S(i+3)+ S(i-3))*G(3) + (S(i+2)+ S(i-2))*G (2) + (S(i+1)+ S(i-1))*G (1)
        + S(i)*G (0))/10000000
    Fin si
Fin pour
```

```
// Bords: on copie les 3 derniers éléments dans T(i)
T(N-3) = S(N-3)
T(N-2) = S(N-2)
T(N-1) = S(N-1)
Fin
```

#### 9.4.2.1.2 Algorithme du code optimisé

**$2^4$ = 16**
**$2^{20}$ = 1048576**
**$2^{24}$ = 16777216**

```
Début
// Filtre gaussien normalisé arrondi et multiplié par 16777216 = 2^24 pour avoir des calculs d'entiers
G = (1943969, 2374369, 2677094, 2786348)

// Bords: on copie les 3 premiers éléments dans T(i)
T(0) = S(0)
T(1) = S(1)
T(2) = S(2)

Pour i allant de 3 à N-4
    // Initialisation
    Resultat_precis = 0
    Resultat_imprecis = 0
    Precision = "0000000"

    // Résultat arrondi à l'entier supérieur ou non
    Resultat_precis = ((S(i+3)+ S(i-3))*G (3) + (S(i+2) + S(i-2))*G (2) + (S(i+1)+ S(i-1))*G (1)
    + S(i)*G (0))/ 1048576

    Resultat_imprecis = (((S(i+3)+ S(i-3))*G (3) +(S(i+2)+ S(i-2))*G (2) + (S(i+1) +S(i-1))*G (1)
    + S(i)*G (0) ) /16777216)*16

    Precision = Resultat_precis – Resultat_imprecis

    Si Precision >= "0001000" alors // 8 en binaire (0,5 multiplié par 16)
        T(i) = ((S(i+3)+ S(i-3))*G (3) + (S(i+2)+ S(i-2))*G (2) + (S(i+1)+ S(i-1))*G (1)
        + S(i)*G (0)) /16777216 + 1
    Sinon
        T(i) = ((S(i+3)+ S(i-3))*G(3) + (S(i+2)+ S(i-2))*G (2) + (S(i+1)+ S(i-1))*G (1)
        + S(i)*G (0)) /16777216
    Fin si
Fin pour

// Bords: on copie les 3 derniers éléments dans T(i)
T(N-3) = S(N-3)
T(N-2) = S(N-2)
T(N-1) = S(N-1)
Fin
```