

Filière Systèmes industriels

Orientation Infotronics

Diplôme 2014

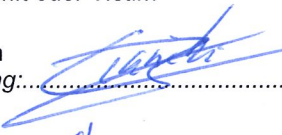

*Jérémie Mayor*

*Camera-based  
daphnia monitoring*

 Professeur  
Pierre-André Mudry  
 Expert  
Julien Pilet  
 Date de la remise du rapport  
11.07.2014

<input checked="" type="checkbox"/> FSI <input type="checkbox"/> FTV	Année académique / Studienjahr <b>2013/14</b>	No TD / Nr. DA <b>it/2014/68</b>
Mandant / Auftraggeber <input checked="" type="checkbox"/> HES—SO Valais <input type="checkbox"/> Industrie <input checked="" type="checkbox"/> Etablissement partenaire <i>Partnerinstitution</i> <b>EPFL</b>	Etudiant / Student <b>Jérémie Mayor</b>  Professeur / Dozent <b>Pierre-André Mudry</b>	Lieu d'exécution / Ausführungsort <input checked="" type="checkbox"/> HES—SO Valais <input type="checkbox"/> Industrie <input type="checkbox"/> Etablissement partenaire <i>Partnerinstitution</i>
Travail confidentiel / vertrauliche Arbeit <input type="checkbox"/> oui / ja <sup>1</sup> <input checked="" type="checkbox"/> non / nein	Expert / Experte (données complètes) <b>Julien Pilet</b> Opticode SA   Rue Neuve 6A   1020 Renens	

Titre / Titel  <p style="text-align: center;"><b>Camera-based daphnia monitoring</b></p>
Description / Beschreibung <p>Le but de ce projet en collaboration avec l'EPFL est de réaliser un système embarqué à base de Cortex STM32F4 permettant de filmer de manière autonome une daphnie et de détecter son rythme cardiaque. Le but de cette expérience est de pouvoir récolter des images, déterminer où se trouve le cœur de la daphnie de manière manuelle puis d'en extraire sa fréquence cardiaque. Il faudra également pouvoir réaliser la compression des images et transmettre les images sur PC via USB. La plateforme optique pour pouvoir réaliser les prises de vues des daphnies sera fournie.</p> Objectifs / Ziele <ul style="list-style-type: none"> <li>— Acquisition d'images sur caméra embarquée + transfert vers PC.</li> <li>— Sauvegarde des images comprimées sur carte SD dans la plate-forme embarquée.</li> <li>— Analyse des images sur PC avec Matlab, amélioration de l'algorithme développé au projet de semestre</li> <li>— Algorithme embarqué de détection de la fréquence cardiaque des daphnies.</li> </ul>

Signature ou visa / Unterschrift oder Visum  Responsable de l'orientation <i>Leiter der Vertiefungsrichtung:</i> .....   <sup>1</sup> Etudiant / Student : ..... 	Délais / Termine  Attribution du thème / Ausgabe des Auftrags: 12.05.2014  Remise du rapport / Abgabe des Schlussberichts: 11.07.2014, 12:00  Expositions / Ausstellungen der Diplomarbeiten: 27 – 29.08.2014  Défense orale / Mündliche Verfechtung: Semaine   Woche 36
--	--

<sup>1</sup> Par sa signature, l'étudiant-e s'engage à respecter strictement la directive DI.1.2.02.07 liée au travail de diplôme.  
Durch seine Unterschrift verpflichtet sich der/die Student/in, sich an die Richtlinie DI.1.2.02.07 der Diplomarbeit zu halten.



## Camera-based daphnia monitoring

Diplômant/e Jérémie Mayor

### Objectif du projet

Réaliser un système embarqué à base de Cortex STM32F4 permettant de filmer de manière autonome une daphnie et de détecter son rythme cardiaque. Le système devra également récolter les images, les compresser et les transmettre sur PC par USB.

### Méthodes | Expériences | Résultats

Les daphnies sont des petits organismes vivant dans l'eau douce. Elles mesurent quelques millimètres. Leur rythme cardiaque se modifie en fonction de l'environnement dans lequel elles évoluent, par exemple en fonction de la pollution de l'eau.

Un algorithme détectant la fréquence cardiaque a d'abord été développé sur Matlab. Il analyse la zone de l'image contenant le cœur de la daphnie en effectuant une transformée de Fourier de la moyenne des pixels de chaque image. Il est alors possible de récupérer la fréquence correspondante au rythme cardiaque, car cette opération mathématique permet d'extraire les différentes fréquences contenues dans le signal.

Un programme PC a été développé pour recevoir les images en provenance de la caméra du système embarqué. Ce programme permet également de sélectionner manuellement la zone contenant le cœur.

L'algorithme a ensuite été implémenté sur le système embarqué. Des tests ont alors été effectués à partir d'images d'une daphnie immobilisée dans un piège. Une simple lentille posée sur la caméra permet un zoom suffisant pour observer la daphnie. Les résultats des tests s'avèrent concluants, le rythme cardiaque de la daphnie est mesurable.

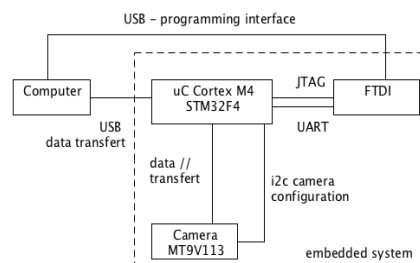
Travail de diplôme  
| édition 2014 |

Filière  
*Systèmes industriels*

Domaine d'application  
*Infotronics*

Professeur responsable  
*Pierre-André Mudry*  
*pierre-andre.mudry@hevs.ch*

Partenaire  
*EPFL*



L'image ci-dessus est le schéma bloc du système embarqué.

L'image ci-dessus est une photo du système embarqué et de la caméra.

# Table des matières

<b>1</b>	<b>Données du projet</b>	<b>1</b>
1.1	Contexte . . . . .	1
1.2	Cahier des charges . . . . .	1
1.2.1	Description . . . . .	1
1.2.2	Objectifs . . . . .	1
1.3	Réalisé durant le projet de semestre . . . . .	1
1.4	Réalisé durant le travail de diplôme . . . . .	2
<b>2</b>	<b>Introduction</b>	<b>3</b>
2.1	Qu'est-ce qu'une daphnie . . . . .	3
2.2	Schéma bloc du système embarqué . . . . .	3
2.3	Environnement de développement et de test . . . . .	4
2.3.1	Développement de l'algorithme . . . . .	4
2.3.2	Développement sur le système embarqué . . . . .	4
2.3.3	Développement de l'application PC . . . . .	4
<b>3</b>	<b>Principe de l'algorithme de détection</b>	<b>5</b>
3.1	Schéma bloc de l'algorithme . . . . .	5
3.1.1	Découpe d'une zone . . . . .	5
3.1.2	Somme . . . . .	6
3.1.3	FFT : transformée de Fourier . . . . .	7
<b>4</b>	<b>Implémentation basique de l'algorithme</b>	<b>8</b>
4.1	Description . . . . .	8
4.1.1	Traitement de l'image . . . . .	8
4.1.2	FFT . . . . .	8
4.2	Résultats . . . . .	9
4.2.1	Vidéo : B2_M2(Ephippia) . . . . .	9
4.2.2	Vidéo : Loading_t0 . . . . .	10
4.2.3	Vidéo : D1_M2 . . . . .	10
4.2.4	Vidéo : B1_M2 . . . . .	11
4.2.5	Vidéo : Loading_t10 . . . . .	12
4.3	Synthèse des résultats et commentaires . . . . .	12



<b>5</b>	<b>Implémentation de l'algorithme avec un filtre</b>	<b>14</b>
5.1	Description . . . . .	14
5.1.1	Design du filtre . . . . .	14
5.1.2	Modification de la FFT . . . . .	15
5.2	Résultats . . . . .	15
5.2.1	Vidéo : B2_M2(Ephippia) . . . . .	16
5.2.2	Vidéo : Loading_t0 . . . . .	16
5.2.3	Vidéo : D1_M2 . . . . .	17
5.2.4	Vidéo : B1_M2 . . . . .	17
5.2.5	Vidéo : Loading_t10 . . . . .	18
5.3	Synthèse des résultats et commentaires . . . . .	18
<b>6</b>	<b>Implémentation de l'algorithme avec une sliding window</b>	<b>20</b>
6.1	Description . . . . .	20
6.1.1	Taille de la fenêtre . . . . .	21
6.2	Résultats . . . . .	21
6.2.1	Vidéo : B2_M2(Ephippia) . . . . .	21
6.2.2	Vidéo : Loading_t0 . . . . .	22
6.2.3	Vidéo : D1_M2 . . . . .	22
6.2.4	Vidéo : Loading_t0 version longue . . . . .	23
6.3	Synthèse des résultats et commentaires . . . . .	23
<b>7</b>	<b>Synthèse des résultats des tests sur vidéos</b>	<b>25</b>
<b>8</b>	<b>Approche du système embarqué</b>	<b>26</b>
8.1	Description . . . . .	26
8.2	Environnement de développement . . . . .	27
8.2.1	Micro-contrôleur . . . . .	27
8.2.2	Programme PC . . . . .	28
8.3	Micro-contrôleur . . . . .	28
8.3.1	Librairie . . . . .	28
8.3.2	Système d'exploitation . . . . .	28
8.3.3	Description des tâches . . . . .	28
8.3.4	Communication entre les tâches . . . . .	29
8.3.5	Description des commandes possibles . . . . .	29
8.3.6	Description du fonctionnement de la caméra . . . . .	30
8.3.7	Format de l'image capturée . . . . .	31
8.4	Programme d'acquisition d'images sur PC . . . . .	31
8.4.1	Description de l'interface . . . . .	31
8.4.2	Environnement . . . . .	32
8.4.3	Diagramme de classes . . . . .	32
8.4.4	Fonctionnement . . . . .	33

<b>9</b>	<b>Modification : prise d'images en continu</b>	<b>35</b>
9.1	Limitation actuelle dû au taux de transfert USB	35
9.2	Modification des programmes	36
9.2.1	Principe de fonctionnement	36
9.2.2	Sélection de la zone (Qt)	38
9.2.3	Transfert des points	39
9.2.4	Réception de la position des points (micro-contrôleur)	40
9.2.5	Découpe de l'image (micro-contrôleur)	41
9.2.6	Réception et affichage de l'image découpée (Qt)	42
9.3	Premier test de capture en continu	42
9.4	Augmentation du nombre d'images par seconde	43
9.4.1	Test de la fréquence de prise d'images par la caméra	43
9.4.2	Changement des paramètres de DMA	43
9.5	Deuxième test de capture en continu	44
9.5.1	Limitation de la taille des images	44
9.5.2	Vérification de la fréquence de prise d'images	44
9.5.3	Remarque	45
<b>10</b>	<b>Mise en place du dispositif de capture d'images</b>	<b>46</b>
10.1	Piège a daphnie	46
10.1.1	Pompe	46
10.1.2	Piège	47
10.2	Choix de la lentille	48
10.3	Éclairage	49
10.3.1	Positionnement de l'éclairage	49
10.3.2	Diffuseur de lumière	50
10.3.3	Filtre	50
<b>11</b>	<b>Test de l'algorithme sur les images de daphnies capturées</b>	<b>52</b>
11.1	Première série d'image	52
11.2	Test de l'algorithme de détection sur la première série d'images	52
11.2.1	Échantillon 1 : "Test8"	52
11.2.2	Échantillon 2 : "TestCoeur"	54
11.2.3	Limitation	54
11.3	Modifications des programmes	55
11.4	Deuxième série d'images	55
11.5	Test de l'algorithme de détection sur la deuxième série	56
11.5.1	Échantillon 1 : "test1-0hz220V"	56
11.5.2	Échantillon 2 : "test2-0hz220vdiphnoir"	57
11.5.3	Échantillon 3 : "test1-2hz220v"	58
11.5.4	Échantillon 4 : "test2-4hz220v"	59
11.5.5	Synthèse et vérification	59

<b>12 Améliorations de l'algorithme Matlab</b>	<b>61</b>
12.1 Suppression du filtre . . . . .	61
12.2 Amélioration de la fenêtre glissante . . . . .	61
12.3 FFT . . . . .	62
12.3.1 Résolution et fenêtre de "Hann" . . . . .	62
12.3.2 Suppression des fréquences sans intérêt . . . . .	63
12.4 Algorithme fenêtre glissante . . . . .	63
12.5 Test du nouvel algorithme . . . . .	63
<b>13 Implémentation des fonctions basiques de l'algorithme sur le micro-contrôleur</b>	<b>66</b>
13.1 FFT sur le micro-contrôleur . . . . .	66
13.1.1 Librairie utilisée . . . . .	66
13.1.2 Fonctions et paramètres utilisés . . . . .	66
13.1.3 Fenêtre du signal . . . . .	67
13.1.4 Environnement de test . . . . .	68
13.1.5 Test . . . . .	68
13.2 Conversion des images en noir et blanc et somme des pixels . . . . .	71
13.2.1 Conversion RGB565 to grayscale . . . . .	71
13.2.2 Test . . . . .	72
<b>14 Implémentation de l'algorithme avec fenêtre glissante</b>	<b>73</b>
14.1 Diagramme de flux . . . . .	73
14.1.1 Process picture . . . . .	74
14.1.2 Algorithm heartbeat . . . . .	74
14.2 Fenêtre glissante . . . . .	75
14.2.1 Fonctionnement . . . . .	75
14.2.2 Implémentation . . . . .	76
14.3 Fenêtre de "Hann" . . . . .	76
14.4 Traitement du résultat de la FFT . . . . .	77
14.4.1 Interprétation des résultats de la FFT . . . . .	77
14.4.2 Suppression des fréquences sans intérêt . . . . .	77
14.5 Stockage de la valeur maximum . . . . .	78
14.6 Test . . . . .	78
14.6.1 Test du traitement des résultats de la FFT . . . . .	78
14.6.2 Test de l'algorithme dans son ensemble . . . . .	78
<b>15 Déroulement du travail</b>	<b>80</b>
15.1 Planning . . . . .	80
15.1.1 Modifications du planning et imprévus . . . . .	80
15.1.2 Tâches non réalisées . . . . .	82
<b>16 Améliorations</b>	<b>83</b>
16.1 Amélioration de l'affichage des battements cardiaques . . . . .	83
16.2 Amélioration de la sélection de la position du cœur . . . . .	83

<b>17 Conclusion</b>	<b>84</b>
<b>18 Bibliographie</b>	<b>85</b>
<b>A Annexe : Vidéos</b>	<b>86</b>
A.1 Choix des vidéos pour les tests . . . . .	86
A.2 Mesure des fréquences cardiaques par comptage . . . . .	87
<b>B Annexe : Résultats complets des tests</b>	<b>88</b>
B.1 Retard du filtre . . . . .	88
B.2 Résultats : algorithme de base . . . . .	89
B.3 Résultats : algorithme avec filtre . . . . .	92
B.4 Résultats : algorithme avec fenêtre glissante . . . . .	94
<b>C Annexe : Comptage des battements cardiaque</b>	<b>96</b>
C.1 Battements cardiaques de la daphnie échantillon 1, 2 et 3 . . . . .	96
<b>D Annexe : Divers</b>	<b>97</b>
D.1 Spécification lentille . . . . .	97
D.2 Documentation FFT CMSIS . . . . .	98
D.3 Planning . . . . .	100

# Chapitre 1

## Données du projet

### 1.1 CONTEXTE

Le travail diplôme est réalisé dans le cadre de formation d'ingénieur en Systèmes Industriels. Il est accompli durant 9 semaines. Ce rapport présente le travail de diplôme ainsi que le projet de semestre qui lui a servi d'introduction.

### 1.2 CAHIER DES CHARGES

#### 1.2.1 Description

Le but de ce projet en collaboration avec l'EPFL est de réaliser un système embarqué à base de Cortex STM32F4 permettant de filmer de manière autonome une daphnie et de détecter son rythme cardiaque. Le but de cette expérience est de pouvoir récolter des images, déterminer où se trouve le cœur de la daphnie de manière manuelle puis d'en extraire sa fréquence cardiaque. Il faudra également pouvoir réaliser la compression des images et transmettre les images sur PC via USB. La plateforme optique pour pouvoir réaliser les prises de vues des daphnies sera fournie.

#### 1.2.2 Objectifs

- ◆ Acquisition d'images sur caméra embarquée + transfert vers PC.
- ◆ Sauvegarde des images comprimées sur carte SD dans la plate-forme embarquée.
- ◆ Analyse des images sur PC avec Matlab, amélioration de l'algorithme développé au projet de semestre.
- ◆ Algorithme embarqué de détection de la fréquence cardiaque des daphnies.

### 1.3 RÉALISÉ DURANT LE PROJET DE SEMESTRE

Le cahier des charges ci-dessus présente la totalité du travail de diplôme. Dans le cadre du projet de semestre, les points suivants ont été abordés :

- ◆ Réalisation d'un algorithme simple de détection de la fréquence cardiaque d'une daphnie.
- ◆ Test de l'algorithme sur PC en utilisant des vidéos de daphnie fournies.
- ◆ Réflexion quant à l'implémentation de l'algorithme sur un système embarqué.



## 1.4 RÉALISÉ DURANT LE TRAVAIL DE DIPLÔME

Les points suivants ont été réalisés :

- ◆ Acquisition d'images sur caméra embarquée + transfert vers PC.
- ◆ Analyse des images sur PC avec Matlab, amélioration de l'algorithme développé au projet de semestre.
- ◆ Algorithme embarqué de détection de la fréquence cardiaque des daphnies.
- ◆ Amélioration du programme PC réceptionnant les images

Par manque de temps, la sauvegarde des images sur carte SD n'a pas pu être implémentée.

## Chapitre 2

# Introduction

### 2.1 QU'EST-CE QU'UNE DAPHNIE

Les daphnies sont des petits organismes vivants dans les eaux douces et stagnantes. Elles sont aussi appelées "puce d'eau" et mesurent quelques millimètres<sup>1</sup>.



FIGURE 2.1 – Daphnie avec dans la zone entourée le cœur.<sup>2</sup>

Elles sont souvent utilisées pour étudier la qualité de l'eau. Le rythme de leurs battements de cœur dépend en effet de la composition de l'eau. Si par exemple elle est polluée, leur fréquence cardiaque va se modifier.

La figure 2.1 montre une daphnie, sur laquelle on peut distinguer son cœur (zone entourée en rouge).

### 2.2 SCHÉMA BLOC DU SYSTÈME EMBARQUÉ

La figure 2.2 présente le schéma bloc du système.

1. Source texte : <http://fr.wikipedia.org/wiki/Daphnie>, consulté le 01.05.14

2. Source image : [http://nl.wikipedia.org/wiki/Daphnia\\_magna](http://nl.wikipedia.org/wiki/Daphnia_magna), consulté le 01.05.14

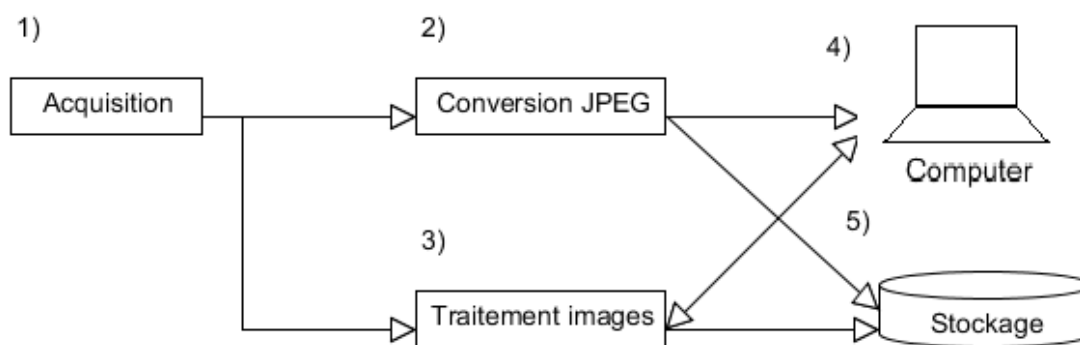


FIGURE 2.2 – Schéma bloc du système embarqué

Les fonctions des blocs sont les suivantes :

- ◆ 1) acquisition d'images au format brute "RGB565"
- ◆ 2) compression des images en "JPEG" afin de diminuer leur taille
- ◆ 3) traitement des images pour déterminer la fréquence cardiaque de la daphnie
- ◆ 4) envoi des images par USB sur le PC et sélection manuelle du cœur
- ◆ 5) stockage des images sur une carte SD formatée en "fat32"

Dans le cadre du travail de diplôme, le traitement des images (bloc 3), l'application et l'envoi par USB d'images (bloc 4) et le stockage (bloc 5) seront développés. Une partie de l'algorithme de traitement d'images a été développée sur Matlab durant le projet de semestre.

## 2.3 ENVIRONNEMENT DE DÉVELOPPEMENT ET DE TEST

### 2.3.1 Développement de l'algorithme

Matlab a été utilisé pour développer et tester l'algorithme. Il a été choisi car il permet de faire du traitement d'images, ainsi que du traitement de signaux facilement, sans avoir besoin de mettre en place un lourd environnement de développement. Le toolbox "image processing" fourni par Matlab permet en effet un traitement d'images simplifié.

Plusieurs vidéos de daphnie ont été fournies pour réaliser les tests. Elles ont été converties en images grâce à un utilitaire gratuit "Free Video to JPG Converter"<sup>3</sup> fournit par "Freestudio". Ce petit logiciel permet d'extraire toutes les images d'une vidéo très simplement.

### 2.3.2 Développement sur le système embarqué

L'environnement utilisé pour développer sur le système embarqué est Eclipse. Le micro-contrôleur est programmé en C. Plus de détails quant à sa configuration sont disponibles au chapitre 8.2.1.

### 2.3.3 Développement de l'application PC

L'application sur le PC est développée en C++ grâce à l'API Qt. L'environnement de développement utilisé est QtCreator. Plus de détails sont disponibles au chapitre 8.2.2.

3. Lien vers le fournisseur du logiciel : <http://www.dvdvideosoft.com/fr/> - section "Convertisseurs"

## Chapitre 3

# Principe de l'algorithme de détection

### 3.1 SCHÉMA BLOC DE L'ALGORITHME

La figure 3.1 présente le fonctionnement de base de l'algorithme permettant de détecter la fréquence cardiaque.

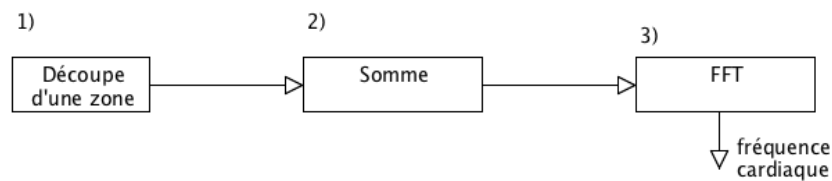


FIGURE 3.1 – Schéma bloc du système embarqué

L'explication concernant les différents blocs est donnée dans les chapitres 3.1.1 à 13.1.

#### 3.1.1 Découpe d'une zone

Cette partie de l'algorithme s'occupe de découper dans l'image la zone contenant le cœur.

En vert sur la figure 3.2 on peut voir la zone contenant beaucoup de mouvements parasites et en rouge le cœur.

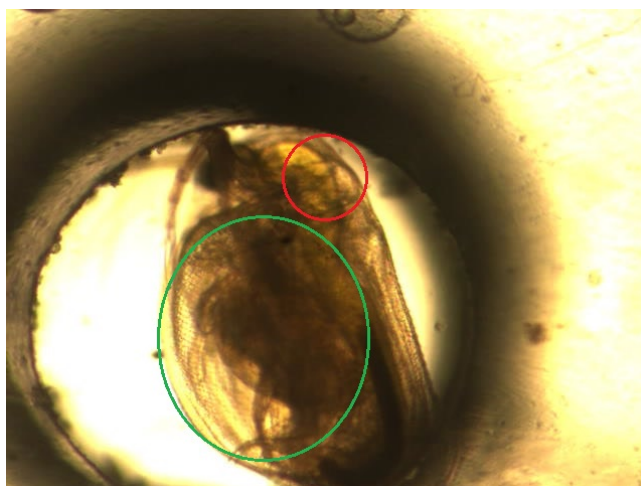


FIGURE 3.2 – Daphnie : en rouge le cœur et en vert la zone des mouvements parasites principaux

L'opération de découpe d'image permet d'éliminer les mouvements parasites. L'image à la figure 3.3 est le résultat de cette opération. Le reste du traitement s'effectue uniquement sur la zone découpée.



FIGURE 3.3 – Zone découpée : cœur de la daphnie

### Stabilité de la daphnie

Un piège pour la daphnie est en cours de développement par M.Frédéric Truffer (Adjoint scientifique à la HES-SO). Ce piège conique devrait la grader immobile et l'empêcher de tourner sur elle-même.

En attendant de pouvoir tester le bon fonctionnement du piège, les daphnies seront considérées comme stables et immobiles. Si les résultats des tests avec le piège s'avèrent peu concluants, il sera alors probablement nécessaire de rajouter un étage de traitement pour supprimer les mouvements.

### 3.1.2 Somme

L'image est transformée en noir et blanc. Lorsque l'algorithme sera implémenté sur le système embarqué, cette étape ne sera plus nécessaire car la caméra est capable de capturer les images directement en noir et blanc.

En noir et blanc, chaque pixel contient une valeur déterminant l'intensité du noir. La valeur de chaque pixels d'une image est sommée. Cette opération est répétée pour chaque image et donne ainsi un vecteur de données contenant une valeur par image. La figure 3.4 montre ce processus.



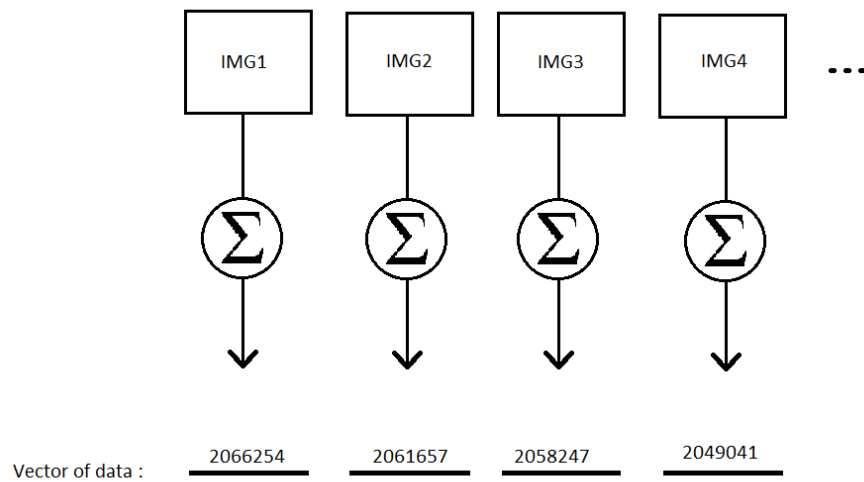


FIGURE 3.4 – Processus de somme

### 3.1.3 FFT : transformée de Fourier

Une transformée de Fourier est une opération mathématique permettant de récupérer les fréquences contenues dans un signal. Tout signal peut être construit à partir d'une somme finie ou non de sinus. Une FFT est la version rapide ("Fast") de cette opération.

La FFT est appliquée au signal obtenu au paragraphe 3.1.2. La fréquence dominante obtenue devrait, en principe, être la fréquence cardiaque de la daphnie.

## Chapitre 4

# Implémentation basique de l'algorithme

### 4.1 DESCRIPTION

La somme des pixels est réalisée pour chaque image (voir chapitre 3.1.2). Puis la FFT est effectuée sur la totalité du vecteur de somme ("vector of data" sur la figure 3.4). Le résultat obtenu est un spectre de fréquence. La totalité des scripts Matlab sont disponibles sur le CD-ROM en annexe.

#### 4.1.1 Traitement de l'image

L'image est découpée grâce à la fonction "imcrop(img,vec)" de Matlab. Elle prend comme paramètre une image (img) et un vecteur indiquant la taille ainsi que la position de la découpe.

#### 4.1.2 FFT

Le script (Listing : 4.1) montre l'implémentation d'une FFT sous Matlab. Ce script est basé sur des exemples trouvés sur le site d'aide en ligne de Matlab "Mathworks <sup>1</sup>" et sur le site de "ele.uri.edu <sup>2</sup>".

Listing 4.1 – Calcul d'une FFT avec Matlab

```
1  %compute the fft
2  Y = fft(u);
3  Y(1)=[];
4
5  n=length(Y);
6
7  % take the abs value of the FFT complex
8  power = abs(Y(1:floor(n/2))).^2;
9
10 % build the vector for the x axis
11 nyquist = 1/2;
12 freq = (1:n/2) / (n/2) *nyquist;
13 freq = freq*Fs;
14
15 plot(freq,power);
```

1. <http://www.mathworks.ch/ch/help/matlab/ref/fft.html> consulté le 06.03.14

2. <http://www.ele.uri.edu/hansen/projects/ele436/fft.pdf> consulté le 06.03.14

## 4.2 RÉSULTATS

Un tableau détaillant le choix des vidéo utilisées pour les tests est fourni en annexe A.1. Pour chaque vidéo, la fréquence cardiaque réelle a été déterminée à la main par comptage des battements du cœur (voir annexe A.2).

Pour certaines vidéos, seul les résultats des FFT sont présentés. La totalité des tests sont disponibles en annexe B.

### 4.2.1 Vidéo : B2\_M2(Ephippia)

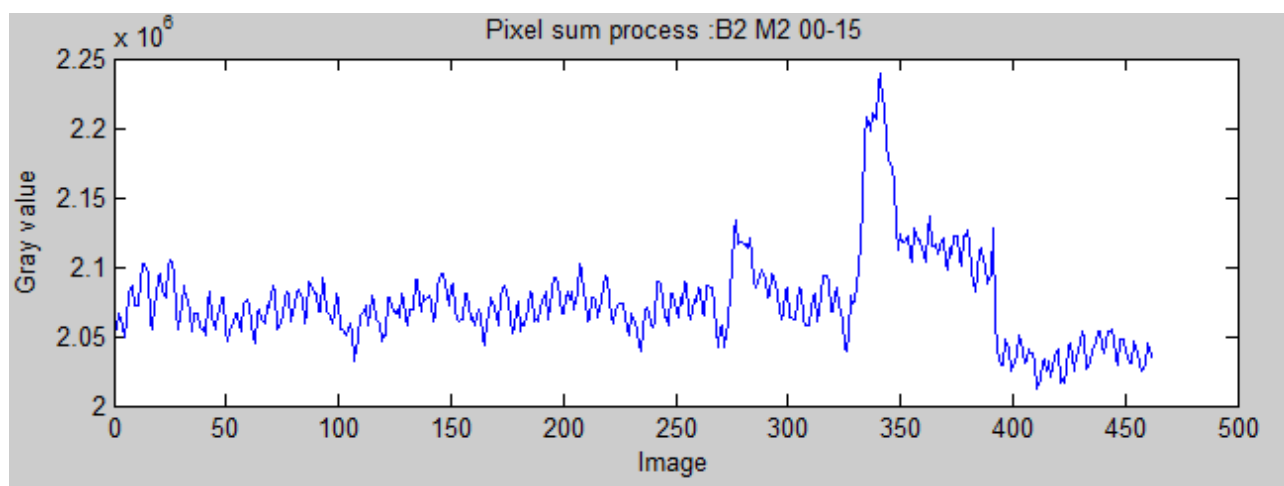


FIGURE 4.1 – Signal des pixels sommés Vidéo : B2\_M2(Ephippia)

La figure 4.1 présente le résultat de l'addition des pixels pour chaque image. En regardant ce graphique, on peut voir que le signal oscille. C'est cette oscillation que la FFT va extraire. Les fortes variations entre les images 300 à 350 correspondent à un changement brusque de luminosité sur la vidéo. Cette variation introduira probablement des pics situés dans les hautes fréquences sur la FFT. Cependant, ces pics ne seront pas problématiques car ils ne seront pas confondus avec ceux correspondant aux battements de cœur de la daphnie.

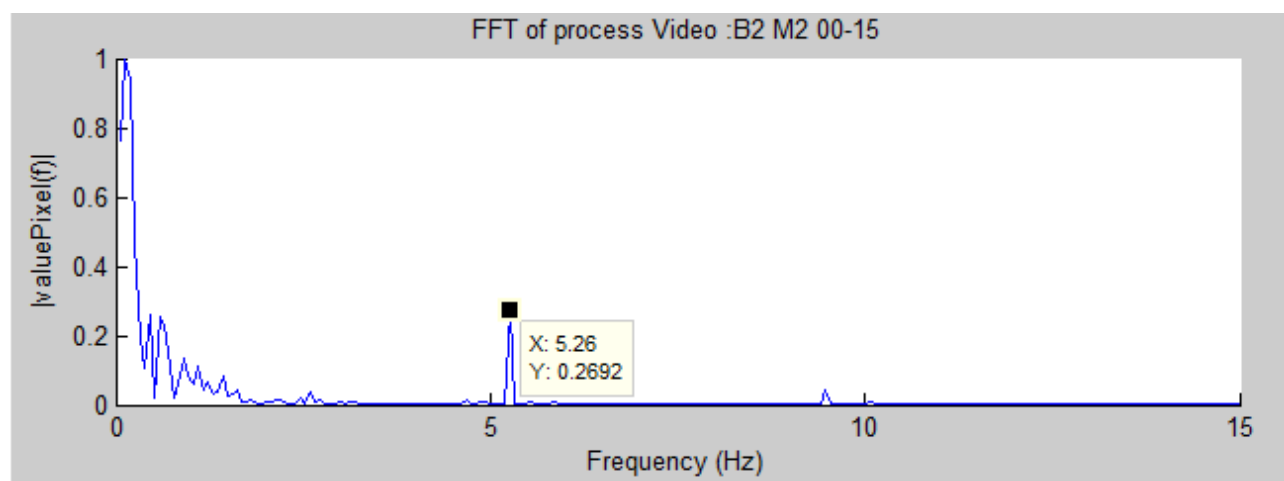


FIGURE 4.2 – FFT du signal des pixels sommés Vidéo : B2\_M2(Ephippia)

La figure 4.2 présente la FFT du signal de la figure 4.1. La valeur de la FFT a été normalisée pour une meilleure visibilité.

Sur le spectre on remarque un pic à 5.26 Hz. C'est la fréquence des battements du cœur de la daphnie. La fréquence cardiaque réelle est de 4.86 Hz (voir annexe A.2). L'erreur relative est donc égale à 8.2%.

On constate également que la FFT comporte des signaux parasites dans les fréquences <1.5Hz.

#### 4.2.2 Vidéo : Loading\_t0

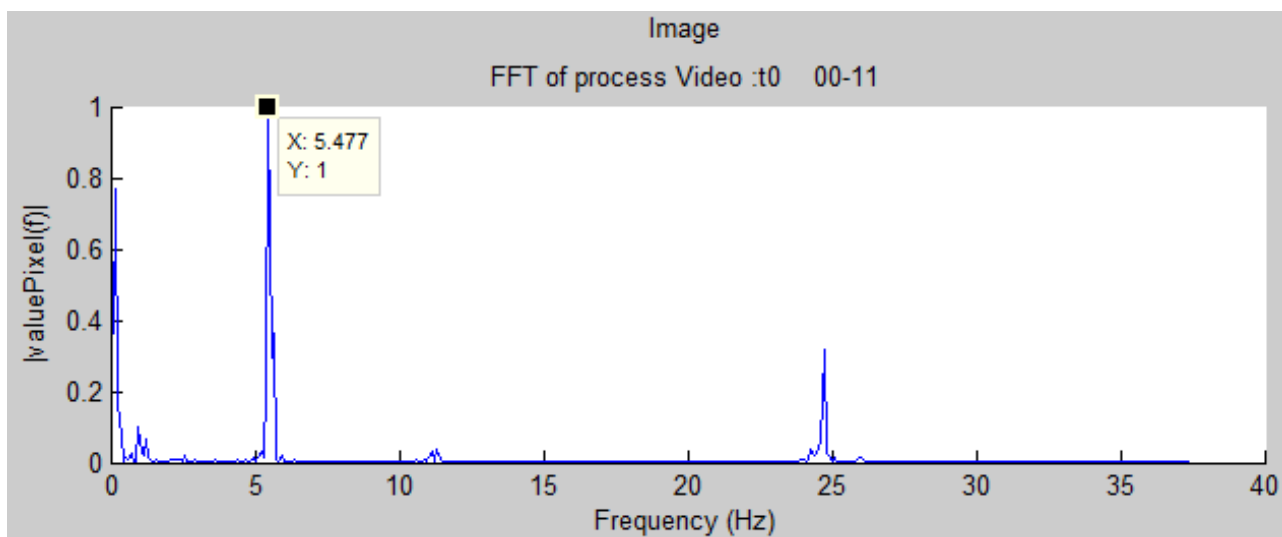


FIGURE 4.3 – FFT du signal des pixels sommés Vidéo : Loading\_t0

La figure 4.3 présente la FFT du signal de la somme des pixels pour la vidéo "Loading\_t0". Le pic correspondant à la fréquence des battements de cœur est bien plus marqué que dans la première vidéo. Il se situe à 5.48 Hz. La vraie fréquence cardiaque étant de 5.65 Hz, l'erreur relative est de -3%.

Il subsiste toujours du bruit dans les basses fréquences, mais d'intensité plus basse que dans la vidéo précédente. On peut également constater la présence d'un pic parasite proche de 25Hz. Ce dernier ne devrait pas poser de problème car il pourra être filtré facilement.

#### 4.2.3 Vidéo : D1\_M2

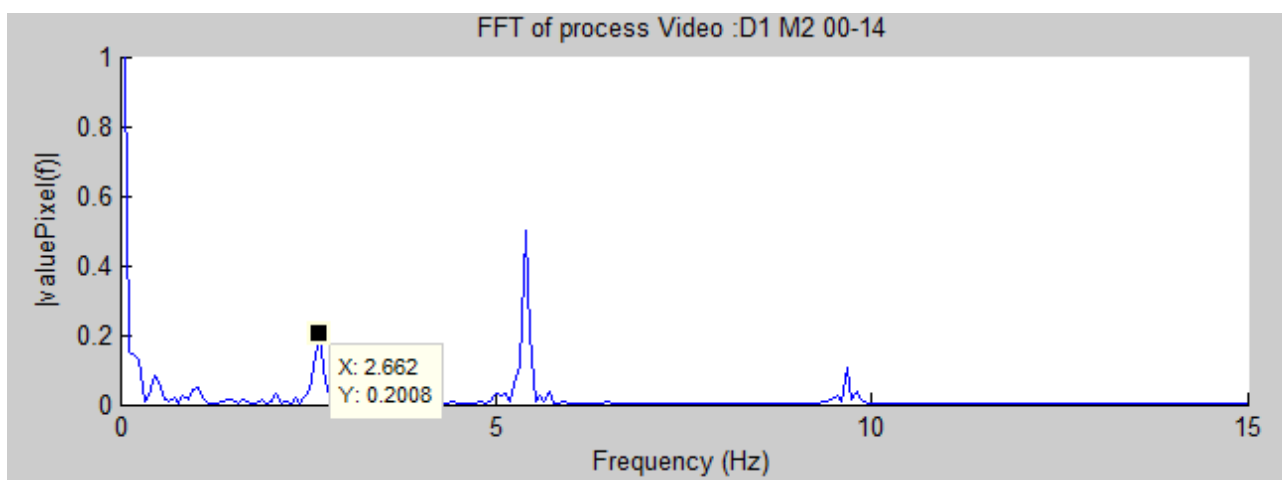


FIGURE 4.4 – FFT du signal des pixels sommés Vidéo : D1\_M2

La figure 4.4 présente la FFT du signal de la somme des pixels pour la vidéo "D1\_M2". On constate que deux pics sont présents, un à 2.66 Hz et un autre à 5.39 Hz. La vraie fréquence cardiaque est de 2.59 Hz (voir annexe A.2), ce qui correspond au premier pic dans le spectre de la vidéo. Lors de l'implémentation future du filtre, il faudra veiller à ce que le pic parasite (5.39 Hz) disparaisse, ou du moins diminue, face à celui de 2.66 Hz.

L'origine du parasite à 5.39 Hz est difficile à expliquer. Idéalement il faudrait pouvoir analyser une autre vidéo dans laquelle la fréquence cardiaque serait proche de 2.5 Hz, afin de voir si un pic comparable à celui de 5.39 Hz est présent. Il est possible que ce pic provienne d'un "aliasing".

#### 4.2.4 Vidéo : B1\_M2

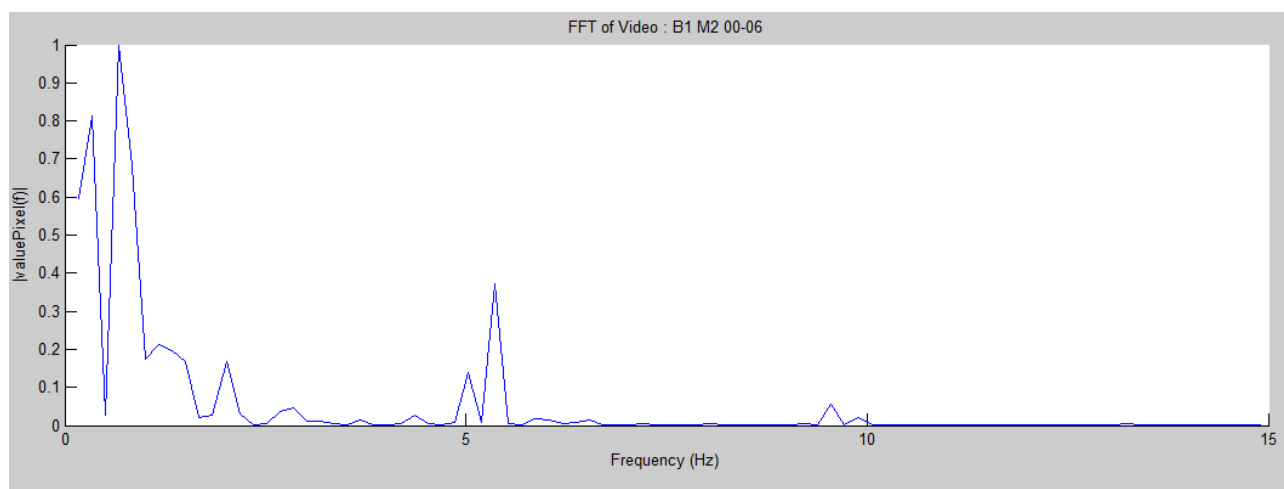


FIGURE 4.5 – FFT du signal des pixels sommés Vidéo : B1\_M2

La figure 4.5 présente la FFT du signal de la somme des pixels pour la vidéo "B1\_M2". Le résultat obtenu par cette FFT est peu concluant. Beaucoup de bruit est présent dans les basses fréquences. De plus, deux pics apparaissent autour de la fréquence des battements cardiaques. Le pic dominant se situe à 5.46 Hz. La vraie fréquence cardiaque est de 5.63 Hz, l'erreur relative est donc de -3.0%.

La courbe n'est pas très lisse, elle manque de points. Avec la fonction FFT de Matlab, la résolution dépend de la taille du signal. Plus il est grand et plus la résolution sera bonne. Dans cette vidéo, le signal est relativement court, il comporte 120 points. Une modification de la FFT devra donc être faite pour conserver une résolution convenable (voir chapitre 5.1.2).



## 4.2.5 Vidéo : Loading\_t10

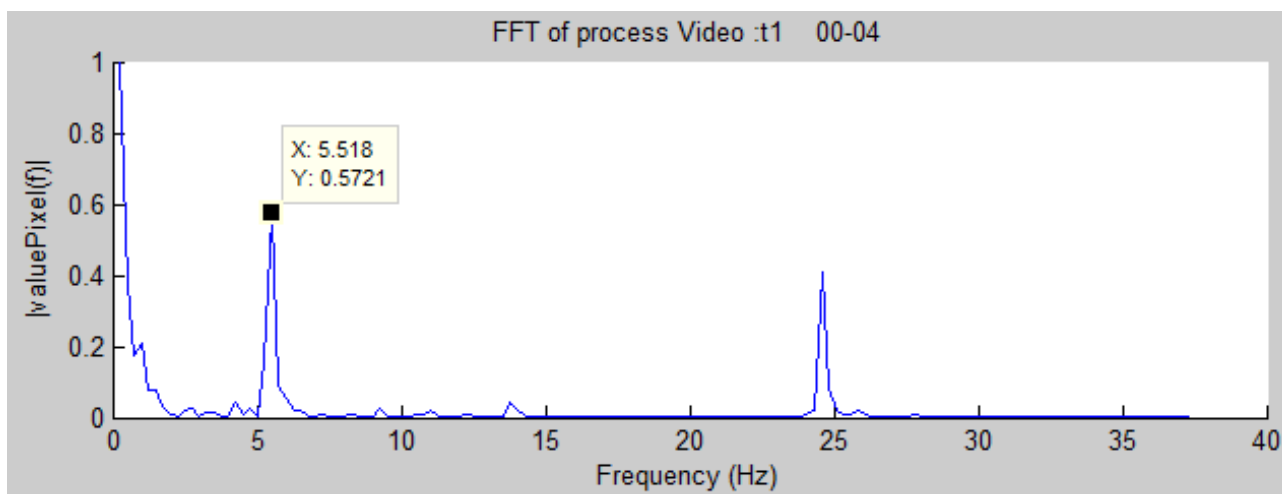


FIGURE 4.6 – FFT du signal des pixels sommés Vidéo : Loading\_t10

La figure 4.6 présente la FFT du signal de la somme des pixels pour la vidéo "Loading\_t10". Le pic indiquant la fréquence cardiaque se situe à 5.52 Hz. La fréquence cardiaque correcte est de 6.75 Hz. L'erreur relative est égale à -18.2 %. Le pic mesuré n'est donc pas correct.

Ce mauvais résultat est difficile à expliquer. Cependant, en regardant plus attentivement les images on remarque des petits points qui se déplacent sur l'image du cœur. Certains peuvent être observés, entourés en rouge à la figure 4.7.

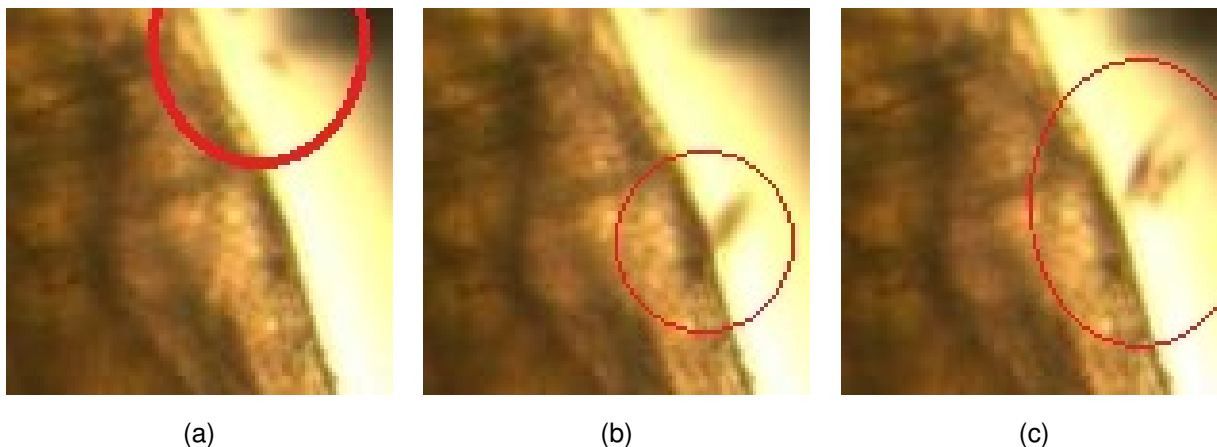


FIGURE 4.7 – (a) img60 (b) img80 (c) img81

Ces parasites se déplacent et apparaissent à plusieurs reprises dans cette vidéo. Il est difficile de déterminer s'ils sont la réelle raison de l'erreur de mesure, mais aucun autre élément ne semble pouvoir l'expliquer.

## 4.3 SYNTHÈSE DES RÉSULTATS ET COMMENTAIRES

Le tableau à la figure 4.8 présente les résultats des tests.

Vidéo	Fréquence réel [Hz]	Fréquence mesurée [Hz]	Erreur relative [%]
B2_M2(Ephippia)	4.86	5.26	8.2
Loading_t0	5.65	5.48	-3.0
D1_M2	2.59	2.66	2.7
B1_M2	5.63	5.46	-3.0
Loading_t10	6.75	5.52	-18.2

FIGURE 4.8 – Résultat avec l'algorithme basique

Les résultats obtenus pour les vidéos "Loading\_t0", "D1\_M2" et "B1\_M2" sont bons, l'erreur ne dépassant pas les 3%. Pour la vidéo "B2\_M2(Ephippia)" l'erreur de 8.2 %, est considérée comme acceptable.

Concernant la vidéo "Loading\_t10" l'erreur de -18,2% n'est pas admissible.

Les premiers résultats obtenus avec cet algorithme de base démontrent que la fréquence cardiaque peut être observée dans le spectre des fréquences. Il reste toutefois beaucoup de pics parasites dans les basses fréquences (<2Hz) et certains dans des fréquences plus hautes (>7 Hz). La prochaine étape consistera à réaliser un filtre passe-bande.

## Chapitre 5

# Implémentation de l'algorithme avec un filtre

### 5.1 DESCRIPTION

Comme vu dans le chapitre 4.3, l'implémentation d'un filtre est nécessaire avant la FFT. La figure 5.1 présente le nouveau schéma bloc avec le filtre.

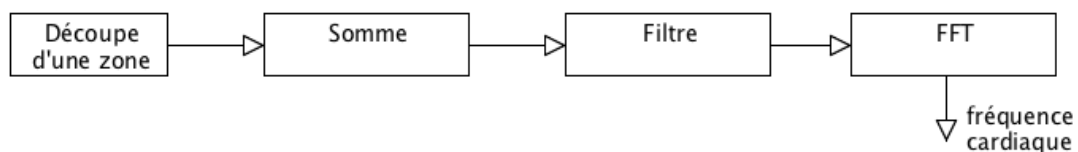


FIGURE 5.1 – Schéma bloc du système embarqué

#### 5.1.1 Design du filtre

Le filtre choisi est du type Butterworth. Il s'agit d'un bon compromis entre les caractéristiques d'un filtre Bessel (bande passante plate et pas d'oscillation) et celles d'un filtre Chebyshev (pente de coupure élevée et grande oscillation).

Le but de ce filtre est de couper le bruit et les parasites dans les basses et hautes fréquences. C'est la raison pour laquelle un filtre passe-bande est implémenté. Les fréquences de coupures sont de 2 Hz (pour la partie passe-haut) et de 5 Hz (pour la partie passe-bas). L'utilisation d'un filtre introduit un déphasage dans le signal filtré. Des tests ont permis de déterminer que le déphasage pour ce filtre correspond à un retard maximum de 100 images (voir annexe B.1). Le filtre utilisé est d'ordre 2.

Le script Listing 5.1 montre l'implémentation du filtre sous Matlab.

Listing 5.1 – Filtre passe-bande Butterworth ordre 2 bande-passante 2 à 5Hz

```

1  fcLP = 2 % Hz
2  fcHP = 5 % Hz
3
4  % Fs = sampling frequency
5  w = [2*fcLP/Fs 2*fcHP/Fs];
6
7  % compute coefficient of filter
8  [b,a]=butter(orderFilter,w,'bandpass');
9
10 % apply the filter to the signal
  
```

```
11 normValuePixel=filter(b,a,newValuePixel);
```

Les coefficients de ce filtre sont les suivants :

1	a :	1.0000	-1.4001	1.2722	-0.6584	0.2722
2	b :	0.1311	0	-0.2622	0	0.1311

La figure 5.2 présente le diagramme de Bode du filtre.

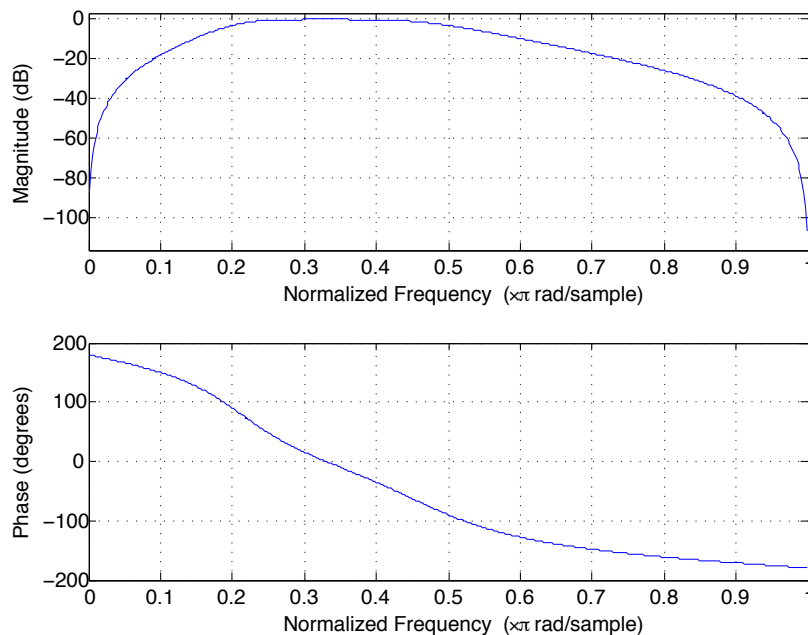


FIGURE 5.2 – Diagramme de Bode du filtre

### 5.1.2 Modification de la FFT

L'implémentation de la FFT a été modifiée pour obtenir un résultat avec une plus grande résolution. Pour cela, il faut rajouter un chiffre en deuxième paramètre de la fonction FFT de Matlab, comme le montre le scripte suivant (Listing 5.2). La valeur "10000" permet d'obtenir une résolution satisfaisante. Pour obtenir plus d'informations, il est possible de consulter l'aide Matlab (help fft).

Listing 5.2 – Modification de la FFT

```
1 %compute the fft
2 Y = fft(u,10000);
```

## 5.2 RÉSULTATS

Seul les spectres de fréquence sont commentés ci-dessous. La totalité des tests se trouvent en annexe B.

### 5.2.1 Vidéo : B2\_M2(Ephippia)

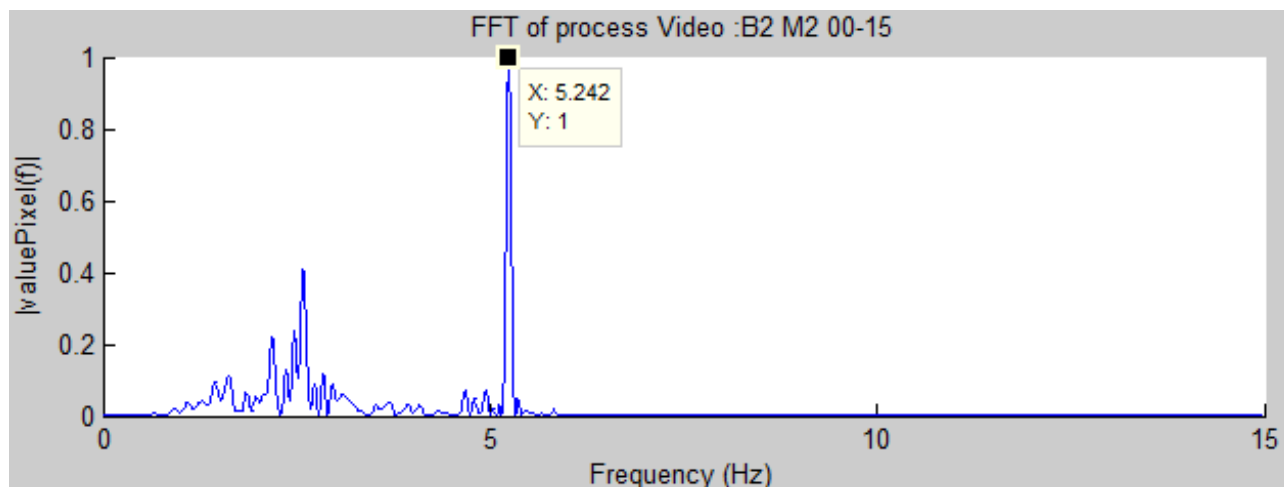


FIGURE 5.3 – FFT du signal des pixels sommés et filtrés, Vidéo : B2\_M2(Ephippia)

La figure 5.3 présente la FFT du signal (somme des pixels de la vidéo "B2\_M2(Ephippia)) filtré. La valeur de la FFT a été normalisée pour une meilleure visibilité. Le pic principal de cette FFT se situe à une fréquence de 5.24 Hz. L'erreur relative par rapport à la fréquence de référence (4.86 Hz) est de 7.8 %.

Le plus grand pic de parasites se situe proche de 2.5Hz, il représente environ 40% du vrai signal. Dans le test précédent, les parasites correspondaient à trois fois le vrai signal. La FFT rend donc un bien meilleur signal avec le filtre.

### 5.2.2 Vidéo : Loading\_t0

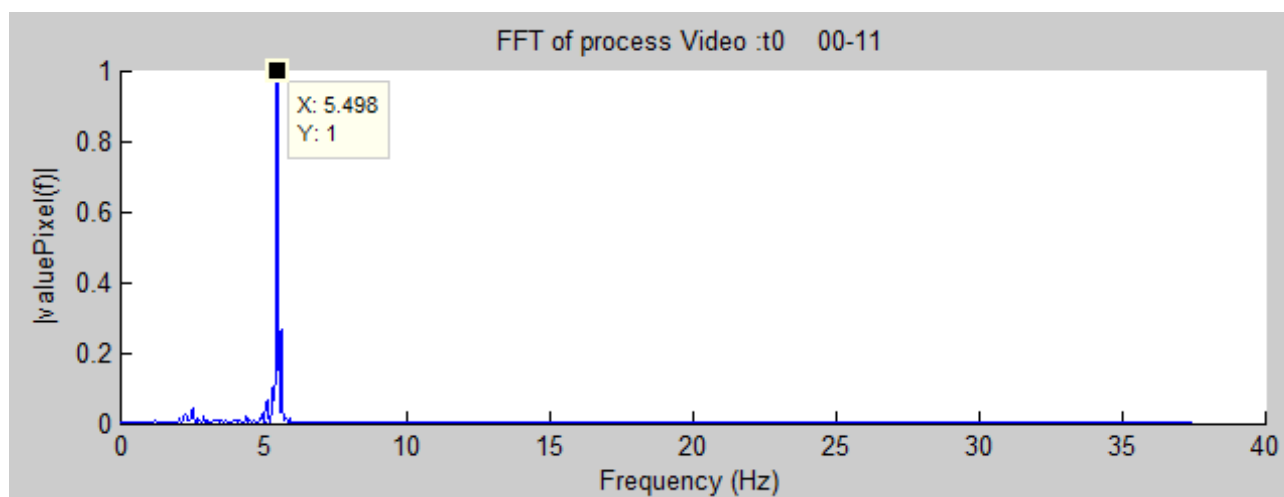


FIGURE 5.4 – FFT du signal des pixels sommés et filtrés, Vidéo : Loading\_t0

La figure 5.4 présente la FFT du signal de la somme des pixels pour la vidéo "Loading\_t0". Le pic correspondant à la fréquence des battements du cœur se situe à 5.5 Hz. L'erreur relative par rapport à la fréquence de référence (5.65 Hz) est de -2.7%.

De plus, les fréquences parasites ont disparu, elles ont donc été totalement filtrées.



### 5.2.3 Vidéo : D1\_M2

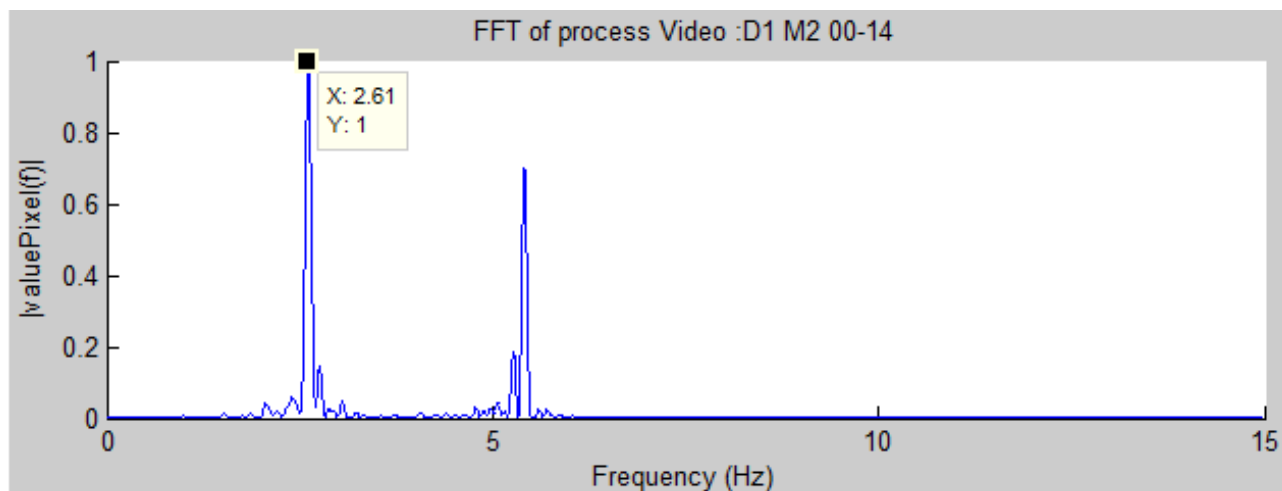


FIGURE 5.5 – FFT du signal des pixels sommés et filtrés, Vidéo : D1\_M2

La figure 5.5 présente la FFT du signal de la somme des pixels pour la vidéo "D1\_M2". Le pic correspondant à la fréquence des battements du cœur se situe à 2.61 Hz. Le pic de 2.61 Hz est plus grand que celui de 5.41 Hz, ce qui n'était pas le cas dans le test sans le filtre.

L'erreur relative par rapport à fréquence de référence de 2.59 Hz est de 0.8 %.

### 5.2.4 Vidéo : B1\_M2

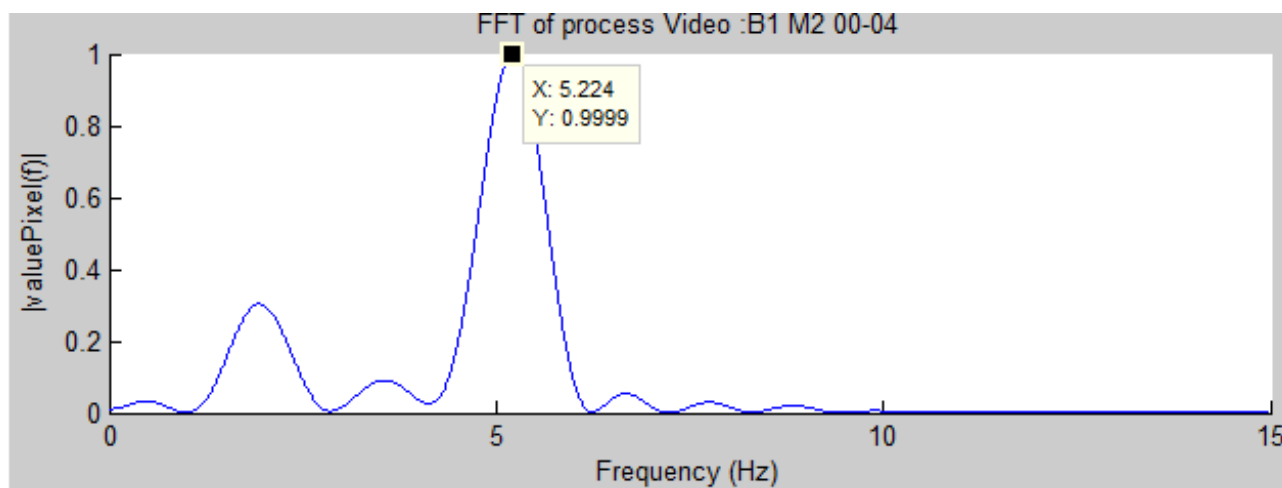


FIGURE 5.6 – FFT du signal des pixels sommés et filtrés, Vidéo : B1\_M2

La figure 5.6 présente la FFT du signal de la somme des pixels pour la vidéo "B1\_M2". Le pic maximum se situe à une fréquence de 5.22 Hz. La fréquence de référence est de 5.63 Hz, ce qui nous donne une erreur relative de -7.3 %.

Le signal de base contient 130 images. Après le filtre, les 100 premières images sont retirées pour permettre au filtre de se stabiliser. La FFT est donc réalisée sur 30 images, ce qui représente (dans cette vidéo) environ une seconde. Ce faible nombre d'images peut expliquer le manque de précision de la distribution des fréquences (pic allongé).

## 5.2.5 Vidéo : Loading\_t10

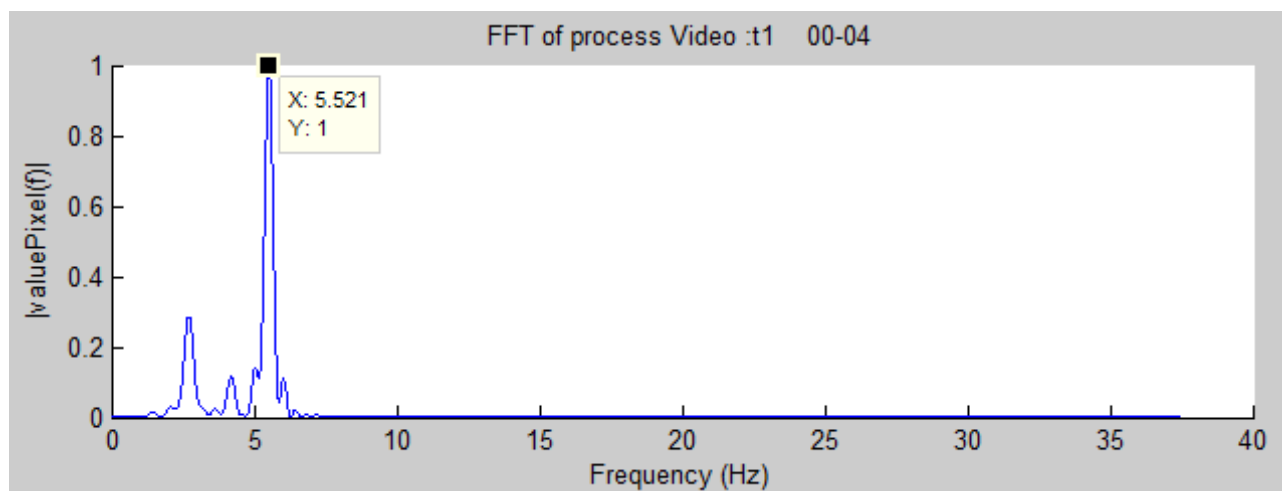


FIGURE 5.7 – FFT du signal des pixels sommés et filtrés, Vidéo : Loading\_t10

La figure 5.7 présente la FFT du signal de la somme des pixels pour la vidéo "B1\_M2". Le pic maximum se situe à une fréquence de 5.52 Hz. La fréquence de référence est de 6.75 Hz. Malgré le filtre, le résultat de cette vidéo est toujours erroné. L'erreur relative est de - 18.2 %, la même que lors du précédent test.

## 5.3 SYNTHÈSE DES RÉSULTATS ET COMMENTAIRES

Le tableau à la figure 5.8 présente les résultats des différents tests.

Vidéo	Fréquence réel [Hz]	Fréquence mesurée [Hz]	Erreur relative [%]
B2_M2(Ephippia)	4.86	5.24	7.8
Loading_t0	5.65	5.5	-2.7
D1_M2	2.59	2.61	0.8
B1_M2	5.63	5.22	-7.3
Loading_t10	6.75	5.52	-18.2

FIGURE 5.8 – Résultat avec l'algorithme avec filtre

Les erreurs obtenues pour les vidéos "B2\_M2(Ephippia)", "Loading\_t0" ont légèrement diminué par rapport au test précédent.

Celle de la vidéo "D1\_M2" a diminué de 2.7 % à 0.8 %.

L'erreur de la vidéo "B1\_M2" passe de -3 % à -7.3%. La fréquence cardiaque de référence est égale à 5.63 Hz, et le filtre a une de ses fréquences de coupure à 5 Hz. On pourrait penser que l'erreur augmente car la vraie fréquence est plus haute que la fréquence du filtre. Mais le problème ne vient pas de là. En effet, la vidéo "Loading\_t0" a également une fréquence de référence plus élevée que celle du filtre, pourtant son erreur n'augmente pas. La vidéo "B1\_M2" comporte peu d'images (environ 130), une centaine sont perdues dans le retard du filtre, la FFT est donc réalisée sur seulement 30 images. On peut tout à fait imaginer que l'augmentation de l'erreur provient de la diminution du nombre d'images.

L'erreur concernant la vidéo "Loading\_t10" reste inchangée, elle est toujours autant grande (voir chapitre 4.2.5).

L'ajout du filtre a eu un résultat positif sur le spectre et sur les erreurs de mesure. Les fréquences de coupures de 2 et de 5 Hz correspondent à un bon intervalle pour récupérer les battements relativement lents (environ 2.5 Hz) ainsi que les plus rapides (environ 5.5 Hz).

L'augmentation de l'ordre du filtre ne produit pas de meilleurs résultats, au contraire, des fréquences parasites réapparaissent. Donc le filtre d'ordre deux est un choix judicieux.

## Chapitre 6

# Implémentation de l'algorithme avec une sliding window

### 6.1 DESCRIPTION

Le principe d'une sliding window (fenêtre glissante) est de récupérer un nombre d'échantillons de "x" à "x + n" (n = taille de la fenêtre) puis de les traiter. Ensuite, la même opération est réalisée avec les échantillons "x+1" à "x + n + 1".

Le schéma 6.1 présente le principe de l'algorithme de la sliding window avec le filtre.

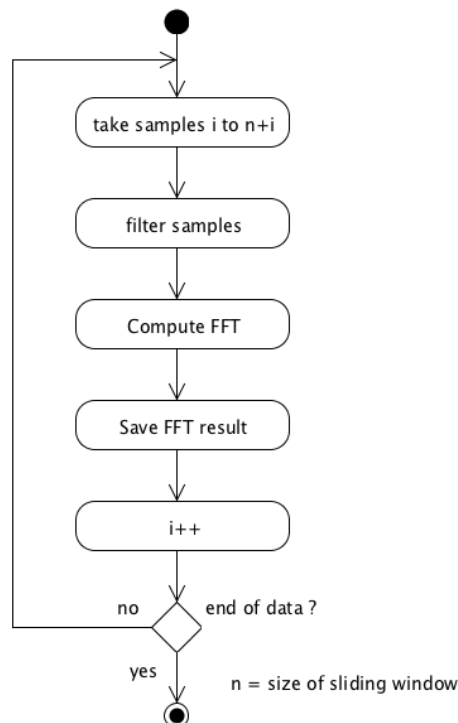


FIGURE 6.1 – Principe de la fenêtre glissante

Le but de cet algorithme est de pouvoir analyser la fréquence cardiaque de la daphnie en fonction du temps. Afin de réaliser cela, le bloc "Save FFT result" sauvegarde chaque résultat de FFT dans un vecteur.

Le scripte implémentant la sliding window est disponible en annexe sur le CD-ROM sous le nom de "sliding-WindowV1.m".

### 6.1.1 Taille de la fenêtre

Comme le filtre choisit introduit un retard d'environ 100 images (voir chapitre 5.1.1), il faut que la taille de la fenêtre soit plus grande que 100. En considérant une fenêtre de 200 images, les 100 premières serviront à "charger" le filtre et ne pourront donc pas être utilisées. Seul les 100 dernières images seront employées pour la FFT. La perte des 100 premières images apparaît car un nouveau filtre est appliqué à chaque nouvelle fenêtre, le filtre Matlab utilisé ne fonctionnant pas en continu.

Une taille de fenêtre de 170 images est utilisée. Elle permet d'obtenir 70 images pour faire la FFT. Comme le montre le tableau à la figure 6.2, à une fréquence d'échantillonnage de 30 img/s, on mesure 6 à 13 battements en 70 images. Avec une fréquence de 75 img/s on en mesure entre 2 et 5.

Fsampling [img / s]	size window [img]	HeartBeat [Hz]	nb of heartbeat
30	70	2.5	5.8
30	70	5.5	12.8
75	70	2.5	2.3
75	70	5.5	5.1

FIGURE 6.2 – Taille de la fenêtre

La fenêtre contient donc entre 2 et 13 battements, ce qui est suffisant pour déterminer leur fréquence.

## 6.2 RÉSULTATS

Seul les vidéos "B2\_M2(Ephippia)", "Loading\_t0" et "D1\_M2" ont été utilisées dans les tests de la sliding window. En effet, les autres vidéos ne contiennent pas suffisamment d'images pour être testées.

### 6.2.1 Vidéo : B2\_M2(Ephippia)

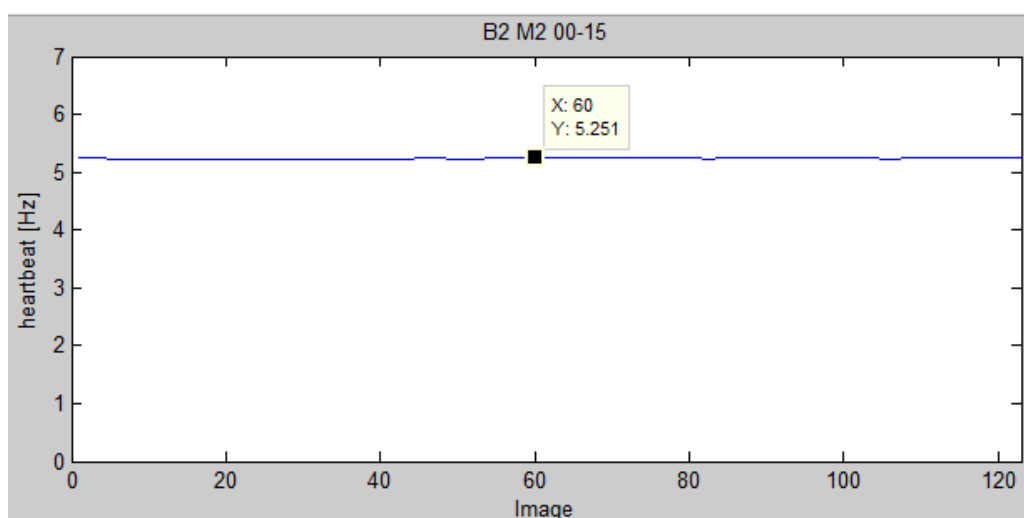


FIGURE 6.3 – Résultat de la sliding window de la vidéo :B2\_M2(Ephippia) avec taille de la fenêtre = 170 img

La figure 6.3 présente le résultat de l'algorithme de la sliding window sur la vidéo B2\_M2(Ephippia). La courbe reste relativement stable et sa moyenne est de 5.24 Hz.

## 6.2.2 Vidéo : Loading\_t0

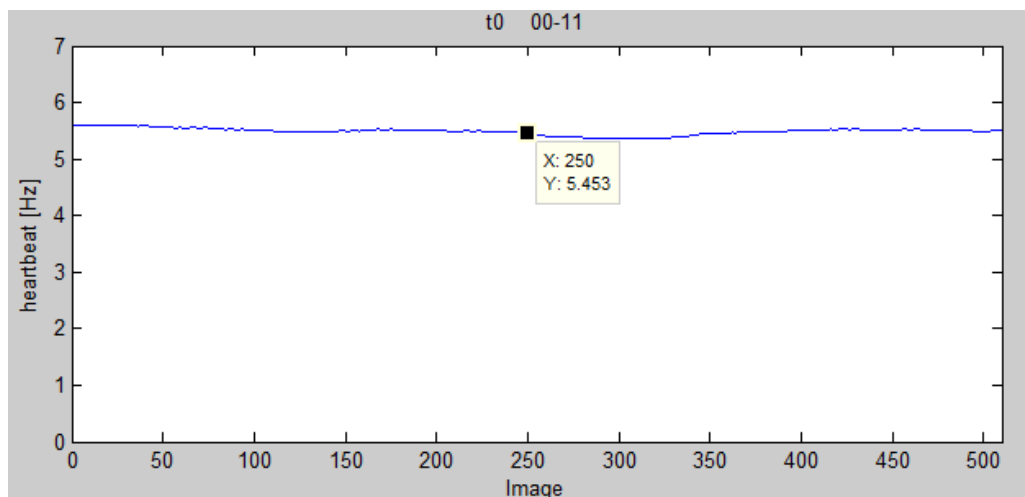


FIGURE 6.4 – Résultat de la sliding window de la vidéo :Loading\_t02 avec taille de la fenêtre = 170 img

La figure 6.4 présente le résultat de l'algorithme de la sliding window sur la vidéo Loading\_t0. La moyenne de la courbe est de 5.49.

## 6.2.3 Vidéo : D1\_M2

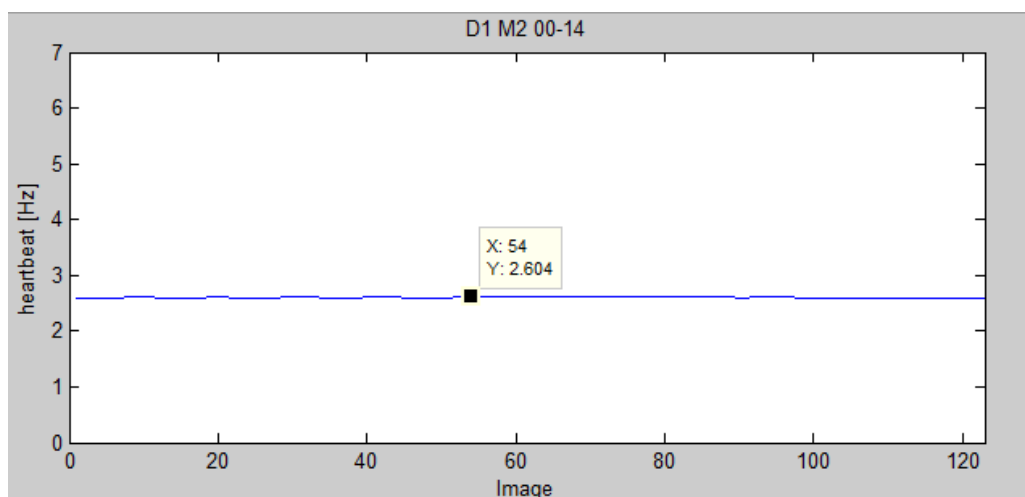


FIGURE 6.5 – Résultat de la sliding window de la vidéo : D1\_M2 avec taille de la fenêtre = 170 img

La figure 6.5 présente le résultat de l'algorithme de la sliding window sur la vidéo D1\_M2. La courbe est elle aussi relativement plate et sa moyenne est de 2.60 Hz.

## 6.2.4 Vidéo : Loading\_t0 version longue

Dans la vidéo "Loading\_t0" version longue, les pattes de la daphnie bougent et passent devant le cœur. Pour cette raison, seul les 11 premières secondes de la vidéo ont été utilisées dans les tests précédents. La totalité de la vidéo a tout de même été testée afin d'avoir un aperçu de l'impact de ce type de parasites sur la vidéo.

La figure 6.6 présente le résultat de l'algorithme de la sliding windows pour cette vidéo.

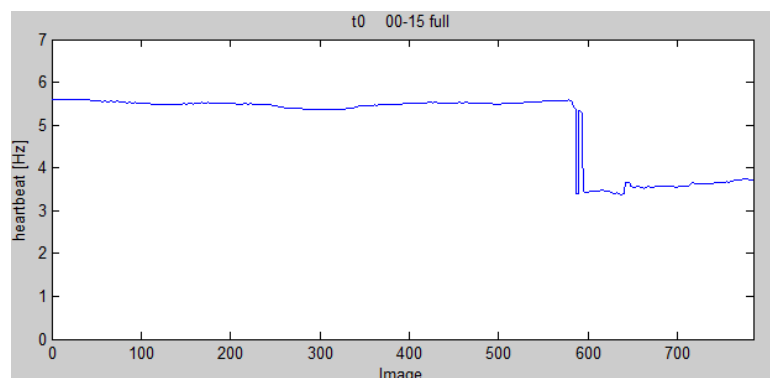


FIGURE 6.6 – Sliding windows de la vidéo : Loading\_t0 version longue avec taille de la fenêtre = 170 img

On constate que la courbe est faussée lorsque les pattes passent devant le cœur de la daphnie (images 580 à 620). Une fois les perturbations passées, la courbe ne se corrige pas. Ce phénomène est peut-être dû à la taille de la fenêtre.

Du moment que des valeurs parasites apparaissent dans la sliding window, elles continueront à influencer le signal tant qu'elles ne seront pas totalement sorties. Les antennes de la daphnie bougent proche des images 580-620. Avec une sliding window de 170 images, la courbe devrait redevenir correcte proche des images 750 - 790. Or le signal s'arrête à 785 images. Il est donc impossible de déterminer si la courbe va reprendre une valeur correcte ou non. Il serait intéressant de pouvoir analyser des antennes qui bougent sur une vidéo plus longue. Malheureusement aucune vidéo de ce type n'est disponible.

## 6.3 SYNTHÈSE DES RÉSULTATS ET COMMENTAIRES

Le tableau à la figure 6.7 présente les résultats des tests.

Vidéo	Fréquence réel [Hz]	Fréquence mesurée [Hz]	Erreur relative [%]
B2_M2(Ephippia)	4.86	5.24	7.8
Loading_t0	5.65	5.49	-2.8
D1_M2	2.59	2.6	0.4

FIGURE 6.7 – Résultat des tests avec fenêtre glissante

L'erreur relative reste assez faible et stable par rapport aux tests avec l'algorithme implémentant uniquement le filtre (chapitre 5.3).

Le calcul de la fréquence réelle ne tient pas compte des erreurs de comptage (voir annexe A.2). Si l'on estime cette erreur à plus ou moins un battement, l'erreur relative maximum est proche de 6.5 %, comme le montre le tableau à la figure 6.8.

Vidéo	Fr. réel min [Hz]	Fr. réel [Hz]	Fr. réel max [Hz]	Fr. mesurée [Hz]	Er. Rel. min[%]	Er. Rel. [%]	Er. Rel. max[%]
B2_M2	4.79	4.86	4.9	5.24	9.4	7.8	6.5
Loading_t0	5.56	5.65	5.7	5.49	-1.3	-2.8	-4.4
D1_M2	2.53	2.59	2.7	2.6	2.8	0.4	-2.3

FIGURE 6.8 – Tableau des résultats avec erreur

### Modification du filtre et de la sliding window

L'implémentation de la sliding window effectuée n'est pas tout à fait correcte. Comme expliqué au chapitre 6.1.1, le filtre Matlab utilisé ne fonctionne pas en continu. Il faudrait trouver un moyen pour modifier le script afin de filtrer en continu les données et ainsi ne plus avoir besoin d'ignorer les x premières valeurs de sortie du filtre pour chaque fenêtre.

Cette modification pourrait aider à comprendre pourquoi dans la vidéo "Loading\_t0" version longue, la courbe ne se rétablit pas après le passage des mouvements parasites (chapitre 6.2.4). Malheureusement, par manque de temps, cette modification n'a pas pu être réalisée.



## Chapitre 7

# Synthèse des résultats des tests sur vidéos

Le tableau à la figure 7.1 présente la synthèse des résultats des tests effectués avec les différents algorithmes.

Vidéo	Algo 1			Algo 2		Algo 3	
	Fr. réel [Hz]	Fr. mesurée [Hz]	Er. Rel. [%]	Fr. mesurée [Hz]	Er. Rel. [%]	Fr. mesurée [Hz]	Er. Rel. [%]
B2_M2(Ephippia)	4.86	5.26	8.2	5.24	7.8	5.24	7.8
Loading_t0	5.65	5.48	-3.0	5.5	-2.7	5.49	-2.8
D1_M2	2.59	2.66	2.7	2.61	0.8	2.6	0.4
B1_M2	5.63	5.46	-3.0	5.22	-7.3	---	---
Loading_t10	6.75	5.52	-18.2	5.52	-18.2	---	---

Algo 1 : Algorithme sans filtre

Algo 2 : Algorithme avec filtre butterworth ordre 2 ( $fc1 = 2$ ,  $fc2 = 5$ )

Algo 3 : Algorithme avec filtre et fenêtre glissante

FIGURE 7.1 – Tableau de synthèse des résultats

Les erreurs relatives restent sous la barre des 8%, sauf pour le cas particulier de la vidéo "Loading\_t10". Le calcul de la fréquence réelle ne tient pas compte des erreurs de comptage. Si l'on estime cette erreur à plus ou moins un battement, l'erreur relative maximum est proche de 6.5 %.

Selon les différents tests, il est possible d'extraire la fréquence des battements de cœur d'une daphnie à partir d'un traitement d'images. Néanmoins, les algorithmes sont peu efficaces en cas de mouvements de la daphnie. Les résultats obtenus dépendront donc grandement de l'efficacité du piège qui immobilisera le sujet.

La prochaine étape consistera à prendre des images de la daphnie dans le piège afin de déterminer si elle est suffisamment stable et tranquille pour le bon fonctionnement de l'algorithme. Si ce n'est pas le cas, il faudra essayer de modifier l'algorithme afin d'y inclure des traitements d'images complémentaires.

## Chapitre 8

# Approche du système embarqué

### 8.1 DESCRIPTION

Ce chapitre a pour but de décrire brièvement le fonctionnement du système embarqué ainsi que du programme permettant le transfert d'images par USB tel qu'ils ont été fournis au début du projet.

La figure 8.1 présente le schéma bloc du système embarqué ainsi que ses liens avec le PC. On constate que deux connections USB sont utilisées, l'une d'elles sert à la transmission des données (USB - data transfert) et l'autre permet la programmation et le débogage du micro-contrôleur Cortex M4 (USB - programming interface).

Le "FTDI" est un composant permettant de faire le pont entre une interface USB et d'autres interfaces. Ici, il permet de faire de l'"UART" pour faire des "printf" et en "JTAG" pour la programmation.

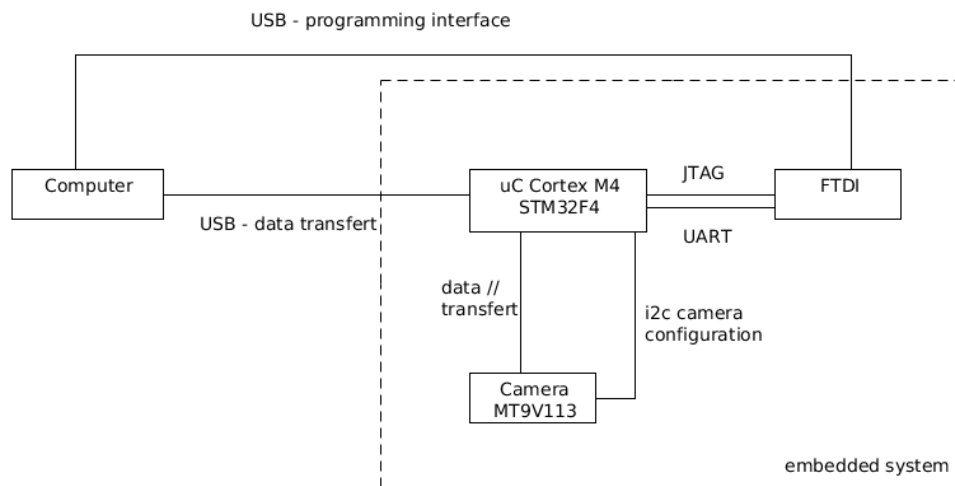


FIGURE 8.1 – Schéma bloc du système embarqué

La figure 8.2 est une photo du système embarqué. On y distingue les principaux éléments FTDI, STM32F4, la caméra et les deux connecteurs USB.

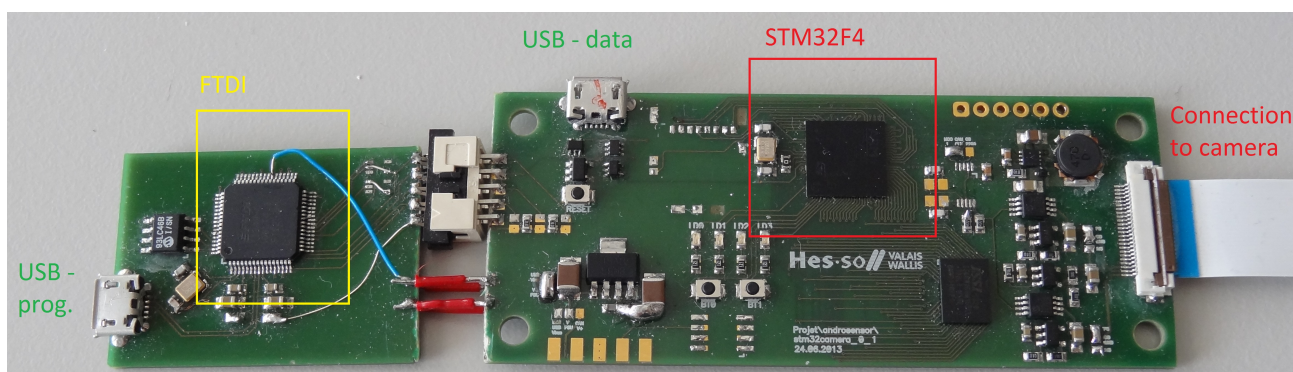


FIGURE 8.2 – Photo du système embarqué

La figure 8.3 est une photo de la caméra. Une lentille est fixée provisoirement sur la caméra (rond en verre). Les deux rectangles jaunes sont les LEDs d'éclairage.

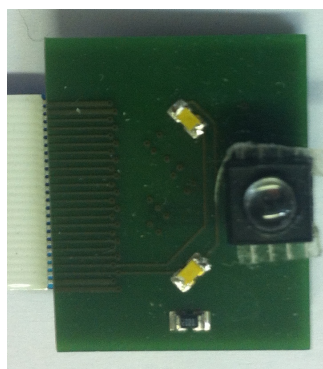


FIGURE 8.3 – Photo de la caméra

Dans la suite de la description, le terme micro-contrôleur sera utilisé pour appeler le Cortex M4 STM32F4 afin d'alléger le texte.

Les sections suivantes présentent l'environnement de développement ainsi que le fonctionnement des programmes tournant sur le Cortex M4 et sur le PC.

## 8.2 ENVIRONNEMENT DE DÉVELOPPEMENT

Le système d'exploitation de l'environnement de développement est Ubuntu version 13.10.

### 8.2.1 Micro-contrôleur

L'IDE (environnement de développement) utilisé pour la programmation du micro-contrôleur est Eclipse 4.3 (Kepler). Un tutoriel détaillant sa configuration est disponible sur le wiki de l'école à l'adresse suivante : "<http://wiki.hevs.ch/uit/index.php5/Tools/Eclipse4STM32>". La programmation du micro-contrôleur se fait en C.

Les logiciels nécessaires sont les suivants :

- ◆ Eclipse (avec le plugging C/C++ Development Toolkit)
- ◆ Make
- ◆ gcc, ld, as

- ◆ gdb
- ◆ ipenocd
- ◆ gtkterm

## 8.2.2 Programme PC

La programmation du programme tournant sur le PC se fait en C++ grâce à l'api Qt. L'IDE utilisé est QtCreator.

## 8.3 MICRO-CONTRÔLEUR

### 8.3.1 Librairie

Les principales bibliothèques utilisées sont "jpeg" et "freertos". La première permet de convertir les images au format "jpeg" avant de les envoyer. "freertos" quant à lui met à disposition un système d'exploitation (voir chapitre suivant). Un certain nombre d'autres bibliothèques sont utilisées pour le fonctionnement général du système embarqué (configuration UART, DMA, ...), elles ne seront que peu (voir pas du tout) étudiées dans la suite du projet.

### 8.3.2 Système d'exploitation

Le programme est basé sur "FreeRTOS". C'est un système d'exploitation en temps réel gratuit et Open source. Il est très utilisé et apprécié par les développeurs de système embarqué. Il permet notamment de travailler avec des tâches à priorité, des sémaphores, des événements, des queues et avec la majorité des autres fonctionnalités disponibles dans les système d'exploitation RTOS. Il est disponible pour 34 architectures différentes. Une plus ample description est disponible sur le site officiel : [http ://www.freertos.org/](http://www.freertos.org/).

### 8.3.3 Description des tâches

Le programme fourni est séparé en différentes tâches. Les paragraphes suivants les présentent brièvement.

#### initTask

Cette tâche a pour rôle d'initialiser le micro-contrôleur. Elle initialise le buffer contenant la dernière image prise et configure également la caméra. Puis elle crée les différentes tâches et les démarre.

#### usbTask

Cette tâche est très importante, c'est elle qui récupère les messages reçus par l'USB. Puis, en fonction des messages, elle exécute les commandes correspondantes. Ces différentes commandes sont détaillées dans le paragraphe 8.3.5.

#### cameraTask

Cette tâche s'occupe d'afficher en continu les images prises par la caméra sur l'écran du système embarqué. Le système embarqué à disposition ne dispose pas d'écran, cette tâche n'est donc pas utilisée.

## hzTask

Cette tâche a pour rôle d'émettre une impulsion sur une LED lorsqu'une image a été capturée.

### 8.3.4 Communication entre les tâches

Ce paragraphe décrit brièvement la communication entre les différentes tâches et couches du micro-contrôleur.

#### Réception des données USB

Les données USB sont réceptionnées par la fonction "cdc\_datarx". Cette fonction récupère les données et les transmet à la tâche "usbTASK" au travers de la queue de message "usbRxQueue". Le message envoyé est un struct de type "usb\_rx\_msg\_t". Ce struct est composé de deux uint8\_t, un stockant la commande "cmd" et l'autre la valeur "val".

#### Envoi des données USB

Les images à envoyer par USB sont stockées dans une structure. Le listing 8.1 présente cette structure.

Listing 8.1 – Structure des messages envoyés par le micro-contrôleur

```
1 struct qSimpleCam_msg_t {  
2     uint32_t magic;  
3     uint32_t status;  
4     uint32_t resolution;  
5     uint32_t auto_exposure;  
6     uint32_t jpeg_size;  
7     uint8_t data[640 * 480 * 2];  
8 };
```

L'adresse de cette structure est ensuite transmise à la fonction "usbd\_cdc\_tx" qui s'occupe d'envoyer les données sur le port USB.

#### Image capturée

Un événement est envoyé lorsqu'une image est capturée. Une explication plus détaillée est donnée au chapitre 8.3.6.

### 8.3.5 Description des commandes possibles

Cette section liste les différentes commandes pouvant être envoyées depuis le port USB sur le micro-contrôleur. Les différentes commandes sont identifiées grâce à un caractère alphabétique envoyé dans le message ('a','c','d','r','e').

#### Prise d'une image : 'a'

Cette commande correspond au caractère 'a'. Lorsqu'un message avec la commande 'a' est reçu, le micro-contrôleur attend que l'événement indiquant que l'image est prête arrive. Puis il utilise le sémaphore et récupère l'image. Il transmet ensuite un pointeur sur l'image à la fonction "send\_picture()" qui réalise la compression "jpeg" et envoie l'image compressée sur le port USB. Enfin, le sémaphore est libéré et la tâche attend le prochain message.

#### **option : 'c'**

Cette commande permet de faire passer la caméra en mode :

- ◆ 0 : normal
- ◆ 1 : mono
- ◆ 2 : sepia
- ◆ 3 : negative
- ◆ 4 : solar
- ◆ 5 : solaruv

Cette commande n'est pas disponible depuis l'interface graphique.

#### **Activation / désactivation de l'auto-exposition : 'd'**

Ce commande permet d'activer ou de désactiver l'auto-exposition de la caméra. Pour cela , la fonction "camera\_setup()" est appelée, avec comme paramètres les nouvelles options de la caméra.

#### **Changement de résolution : 'r'**

Ce commande permet de changer la résolution de la caméra. Trois résolutions différentes sont disponibles.

- ◆ 640 x 480
- ◆ 320 x 240
- ◆ 160 x 120

La commande "camera\_setup()" configure les nouveaux paramètres de la caméra.

#### **Luminosité éclairage : 'e'**

Cette commande permet de changer l'intensité des LEDs qui se trouvent à coté de la caméra. Ces LEDs fournissent un éclairage de l'objet pris en photo par la caméra. L'intensité peut varier de 0 à 100 %. Ce réglage s'effectue grâce à la fonction "bsp\_light\_set()".

### **8.3.6 Description du fonctionnement de la caméra**

La caméra est de type "MT9V113" de "Aptina". Elle possède une résolution maximum de 640 x 480 pixels (format VGA). Son prix est relativement faible, environ 5.2 chf par pièce pour 50 pièces commandées. Elle est configurée par un bus i2c. Les images sont transférées directement en mémoire RAM grâce au contrôleur DMA.

Les librairies "stm32\_camera" et "heivs\_stm32\_camera\_mt9v114" permettent de contrôler la caméra (pour la première) et de la configurer (pour la deuxième).

La librairie "stm32\_camera" permet de lancer la caméra soit en mode "one\_shot" (une seule image capturée), soit en mode "continuous" (images prisent en continu). Les principales fonctions commandant la caméra sont :

- ◆ camera\_continuous\_start() : démarre le mode de prise d'images en continu.
- ◆ camera\_continuous\_stop() : arrête le mode de prise d'images en continu.
- ◆ camera\_one\_shot\_start() : démarre la prise d'une image.

- ◆ camera\_one\_shot\_wait() : attend que l'image soit correctement prise.
- ◆ camera\_setup() : configure les paramètres d'image de la caméra (résolution, exposition, mode, ...).

La caméra est principalement utilisée en mode continu. En mode continu, un événement est généré par le système d'exploitation lorsqu'une nouvelle image est disponible en mémoire. Cela a pour effet de libérer toutes les tâches qui attendaient sur celui-ci.

La génération d'un événement se fait grâce à la primitive du système d'exploitation "xEventGroupSetBitsFromISR" et l'attente grâce à "xEventGroupWaitBits".

### 8.3.7 Format de l'image capturée

L'image envoyée par la caméra est au format RGB565. Ce format est composé de 16 bits par pixel, 5 pour le rouge, 6 pour le vert et 5 pour le bleu.

Un pointeur permet d'accéder à l'image, les données des pixels sont stockées à la suite dans la mémoire, comme le montre la figure 8.4.

	8 bits	8 bits	8 bits	8 bits
pointeur =>	pixel 1		pixel 2	
	pixel 3		pixel 4	
	pixel 5		pixel 6	
	pixel 7		pixel 8	
	...		...	

FIGURE 8.4 – Disposition de l'image dans la mémoire

## 8.4 PROGRAMME D'ACQUISITION D'IMAGES SUR PC

### 8.4.1 Description de l'interface

La figure 8.5 présente l'interface du programme QSimpleCam. Ce programme permet de capturer une image avec le micro-contrôleur, de l'afficher et si besoin de la sauvegarder. On peut également activer / désactiver l'exposition automatique et changer la résolution de 640X480 à 320X240. Le curseur "light :intensity" permet de changer la luminosité des LEDs d'éclairage.

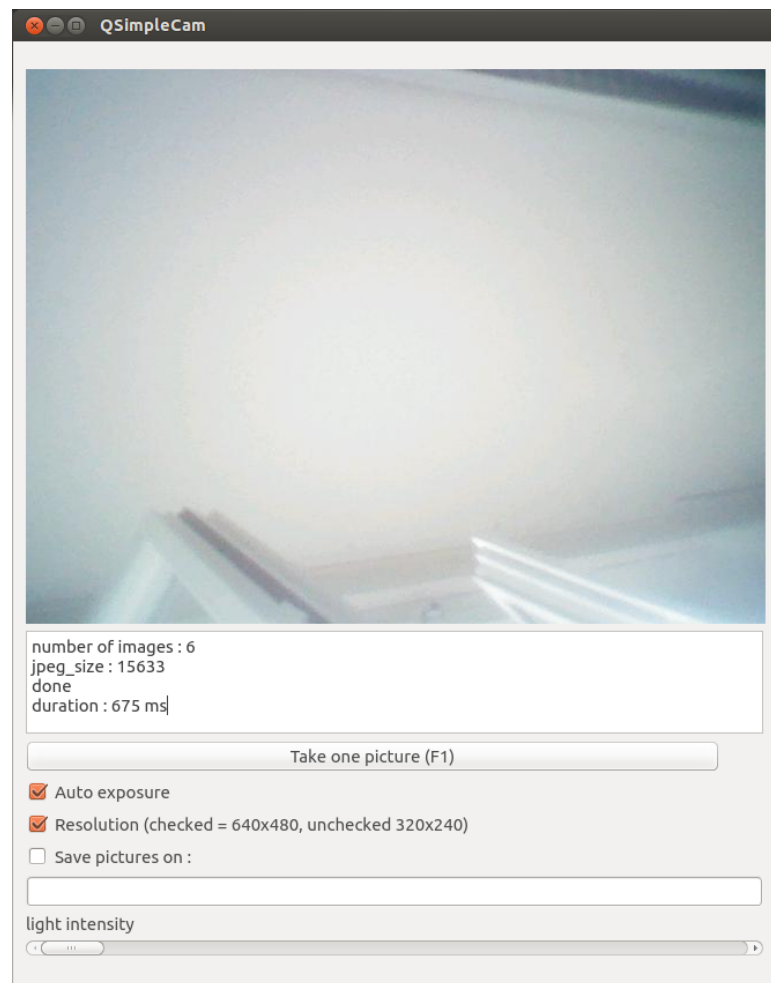


FIGURE 8.5 – Interface du programme PC basique

### 8.4.2 Environnement

Le programme est développé en C++ grâce à l'API QT. Cette API orienté objet permet de créer facilement des interfaces graphiques.

La librairie "qextserialport" a été utilisée pour communiquer à travers le port USB. Elle permet de créer un port série à partir du port USB.

### 8.4.3 Diagramme de classes

La figure 8.6 présente le diagramme de classes du programme d'acquisition d'images.



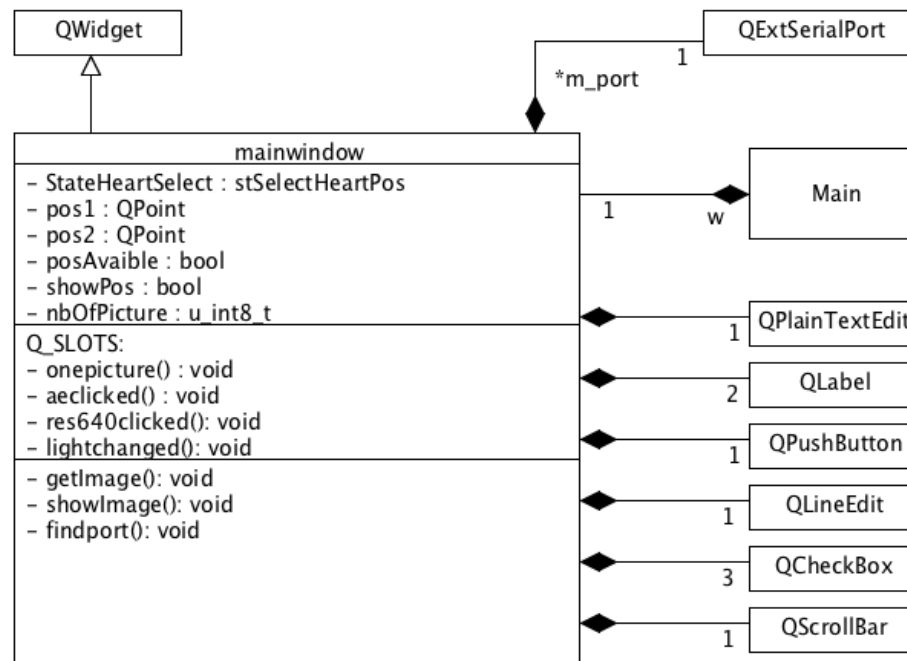


FIGURE 8.6 – Diagramme de classes

#### 8.4.4 Fonctionnement

Le programme dispose d'un certain nombre de fonctions, voici la description des principales :

##### **findport()**

Cette fonction permet de configurer et de créer le port série à travers le port USB correct.

##### **onepicture()**

Cette fonction envoie par USB l'ordre de prendre une photo (voir chapitre : 8.3.5). Puis elle appelle la fonction `getImage()` et `showImage()` (voir description ci-dessous).

##### **getImage()**

Cette fonction lit les données en provenance du port USB (dans ce cas les images) et les placent dans un buffer. La figure 8.7 présente le format dans lequel l'image est reçue.

magic
status
resolution
auto_exposure
jpeg_size
data

FIGURE 8.7 – Image reçue par USB

Cette fonction contrôle également que l'image a été entièrement reçue. Elle compare la taille des données "data" avec celle écrite dans le champ "jpeg\_size". Si l'image n'est pas reçue totalement ou qu'un autre problème est détecté (valeur de "magic" ou de "status" incorrecte) un message est affiché à l'utilisateur.

### **showImage()**

Cette fonction récupère les images depuis le buffer et elle crée l'image à afficher grâce à la méthode "QImage : :fromData(...)". Puis elle affiche l'image à l'écran et l'enregistre si la case "Save pictures on :" (figure : 8.5) est cochée.

### **aeClicked(), res640Clicked(), lightChanged()**

Ces trois fonctions sont appelées lorsque l'utilisateur interagit avec l'interface graphique. Elles correspondent aux cases "Auto exposure" ou "Résolution" et à la barre de changement de luminosité "light.intensity". Dès qu'une des ces fonctions est appelée, elle envoie sa commande sur le port USB (voir chapitre 8.3.5).

## Chapitre 9

# Modification : prise d'images en continu

Le programme du système embarqué doit être modifié pour prendre en continu les images et les envoyer sur l'ordinateur. Ces images permettront de tester le bon fonctionnement des algorithmes développés au préalable.

La totalité du code source des différents programmes est disponible en annexe sur le CD-ROM.

### 9.1 LIMITATION ACTUELLE DÛ AU TAUX DE TRANSFERT USB

Le cœur de la daphnie peut atteindre une fréquence de battement de 6hz. Il faut donc avoir théoriquement au minimum 12 images/secondes. Une fréquence entre 15 et 20 img./sec. serait souhaitable pour avoir une marge. A titre indicatif, les vidéos utilisées pour les tests précédents avaient 30 ou 75 img./sec.

La configuration actuelle ne permet pas d'atteindre une prise d'images autant élevée. Les images sont compressées en "jpeg" puis envoyées, leur taille une fois compressées atteint environ 16 Kbyte. La compression "jpeg" demande beaucoup de ressources et de temps au micro-contrôleur. Il faut en moyenne compter 500ms par image. La vitesse atteinte serait donc de 2 img/sec, ce qui est insuffisant.

L'envoi des images entières sans compression n'est également pas envisageable. Une image de 480\*640 pixels correspond à 614 Kbyte. Avec un débit du port USB d'environ 1 Mbyte/s (USB1.0) on ne pourrait même pas en envoyer 2 par seconde.

La solution pour résoudre ce problème consiste à découper l'image et à envoyer uniquement la zone découpée non compressée. Cette zone contiendra seulement le cœur de la daphnie et devra être sélectionnée manuellement par un opérateur.

Le tableau à la figure 9.1 présente les fréquences théoriques qui pourraient être atteintes en découpant les images.

USB vitesse (bytes/sec)      1000000

pixel x	100	125	150	160	200
pixel y	100	125	150	160	200
taille img (byte)	20000	31250	45000	51200	80000
transfert max (image/sec)	50.0	32.0	22.2	19.5	12.5

FIGURE 9.1 – Taux de transfert maximum théorique

On constate que transférer 20 images par secondes est possible avec une zone d'environ 150 par 150 pixels.

## 9.2 MODIFICATION DES PROGRAMMES

### 9.2.1 Principe de fonctionnement

Le diagramme à la figure 9.2 présente le principe utilisé pour récupérer les images. Le diagramme n'est pas complet, le nom des méthodes ainsi que des instances ne correspondent pas.

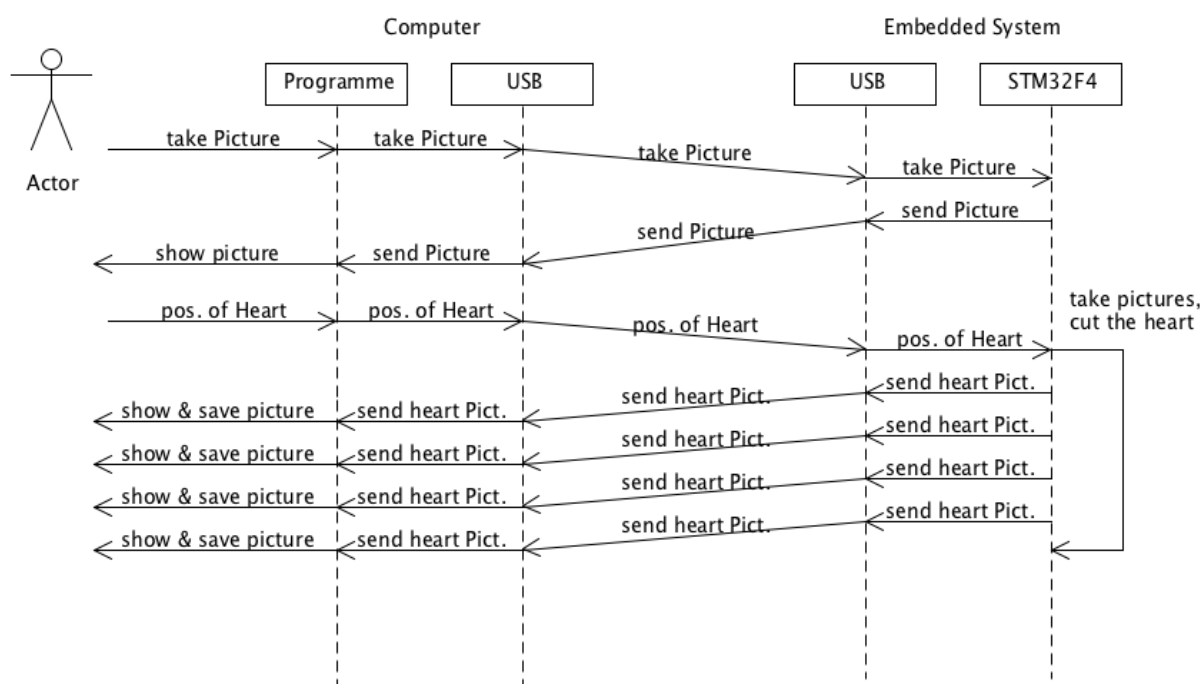


FIGURE 9.2 – Diagramme de séquence du principe d'extraction d'images

L'utilisateur (représenté par le symbole "Actor") emploie le programme sur le PC pour ordonner au micro-contrôleur de capturer et d'envoyer une image (take Picture). L'image est ensuite affichée sur le programme PC (show picture). L'utilisateur peut choisir de reprendre une image ou de sélectionner l'emplacement du cœur sur l'image (pos. of Heart). Une fois que les coordonnées sont déterminées, elles sont envoyées au micro-contrôleur. Les images sont alors prises en continu, découpées puis envoyées par USB sur l'ordinateur. Ensuite, elles sont affichées et sauvegardées.

#### Digramme de flux

La figure 9.3 représente le digramme de flux qui sera implémenté. Sur ce digramme, l'état "start" correspond à la réception du message indiquant la position du cœur.

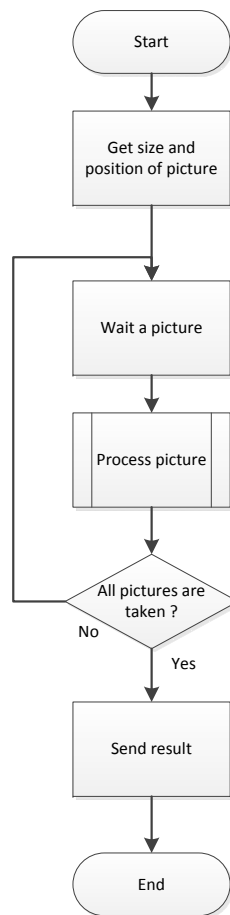


FIGURE 9.3 – Diagramme de flux

### Diagramme de classes

La figure 9.4 présente le nouveau diagramme de classes du programme d'acquisition d'images. On peut y voir plusieurs nouvelles fonctions, notamment "getPartImage()" "showPartImage()" et le "Q\_SLOT" "mouseReleaseEvent(...)".

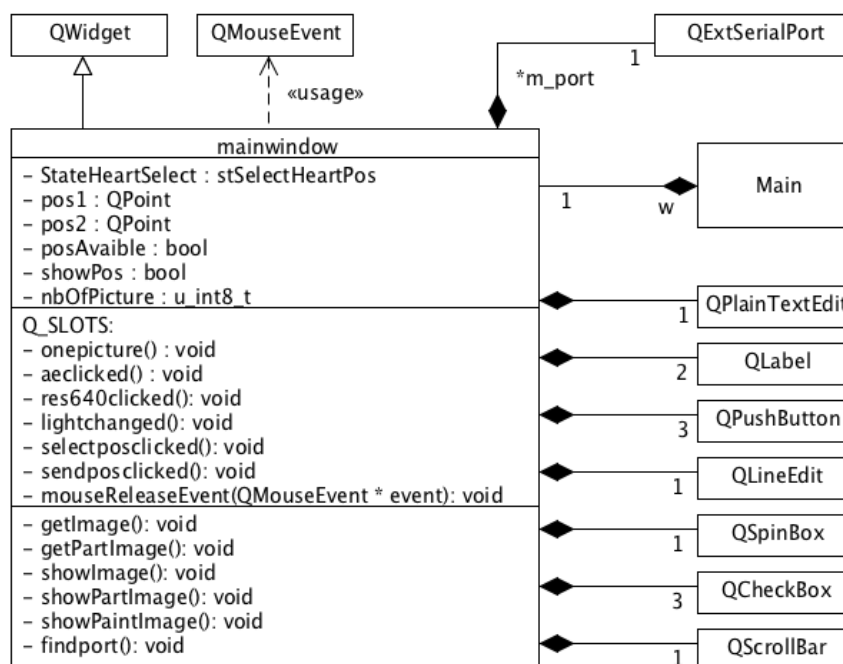


FIGURE 9.4 – Diagramme de classes

## 9.2.2 Sélection de la zone (Qt)

Pour la sélection de la zone, deux boutons ont été rajoutés à l'interface de base, "Select heart position (F2)" et "Send position and start process (F3)".

La sélection de la zone est très simple. Il suffit de cliquer sur le bouton "Select heart position (F2)", puis avec le clique droit de la souris, indiquer directement sur l'image la position de deux coins du rectangle allant être découpé. Le rectangle s'affiche alors en rouge sur l'image, voir figure 9.5.

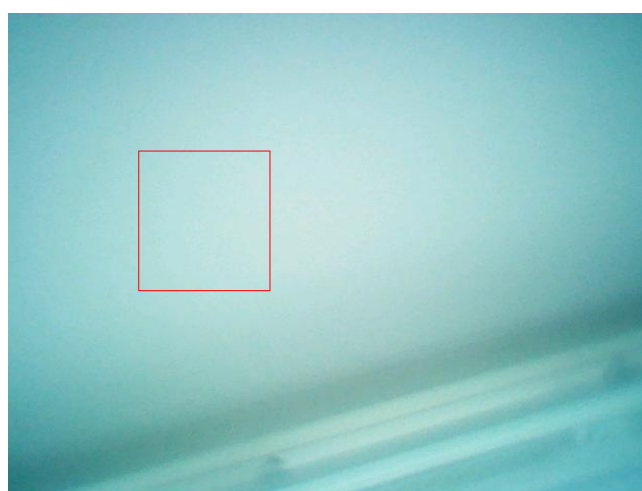


FIGURE 9.5 – Zone sélectionnée en rouge

Une zone de texte permet d'indiquer le nombre d'images (entre 0 et 255) allant être capturées. Une valeur de 0 correspond à 1000 images. Ensuite, une pression du bouton "Send position and start process (F3)" permet de démarrer le processus d'acquisition continu.

## Récupération des points de l'image (Qt)

Ce paragraphe décrit brièvement la méthode utilisée dans Qt pour récupérer les positions des cliques de souris.

La méthode `mouseReleaseEvent()` de la classe `mainWindow` est appelée à chaque fois qu'un clique est "relâché" dans la fenêtre (gérer par l'API Qt). Lorsque cette méthode est appelée et que l'on souhaite récupérer la position de la souris, la fonction `QCursor : :pos()` est alors appelée. Cette dernière renvoie la position absolue du curseur, c'est à dire par rapport au coin supérieur gauche de l'écran. Il faut donc soustraire la position de l'image (`m_ui->label->x()` ou `y()`) à cette valeur. Il faut également soustraire la taille de la bordure de la fenêtre (en x 2 et en y 30). Le listing 9.1 affiche le code complet correspondant.

Listing 9.1 – Récupération de la position du curseur

```
1 pos1.setX(QCursor::pos().x()-m_ui->label->x()-x()-2);  
2 pos1.setY(QCursor::pos().y()-m_ui->label->y()-y()-30);
```

### 9.2.3 Transfert des points

La commande qui indique au micro-contrôleur qu'il peut démarrer la prise d'images en continu est représenté par le caractère 'p'. Elle est reçue et traitée par la tâche "usbTask" au même titre que les commandes expliquées au chapitre 8.3.5. La commande contient comme valeurs les quatre points correspondant à la découpe et également le nombre d'images allant être capturées.

Les données sont envoyées sous forme d'un tableau de `uint8_t`. Les valeurs de position varient entre 0 et 639 pour l'axe des X et entre 0 et 479 pour l'axe des Y. Les 8 bits d'un `uint8_t` (0 à 255) ne suffisent donc pas à stocker une valeur de position.

Une particularité existe dans la librairie USB du micro-contrôleur. Lorsqu'une donnée contenant uniquement des 0 (0x00) est reçue, le système embarqué voit une séparation de paquet et ignore cette donnée. Par exemple, lors de l'envoi d'un message contenant 0xffaa00bb, le micro-contrôleur reçoit un message contenant 0xffaa et un autre 0xbb. Ce problème est gênant. En effet il est tout à fait possible qu'une des positions soit égale à zéro, ou alors que lors de la concaténation des positions une valeur 0x00 apparaisse.

Pour résoudre ce problème rapidement et simplement, la structure suivante (voir figure 9.6) de messages a été adoptée. Elle assure qu'aucun 0x00 ne soit transmis. Sur la partie gauche de l'image, on peut voir les positions avant concaténation, les valeurs sont stockées dans des `uint16_t` et seul 14 bits sont utilisés. Sur la partie de droite de l'image, on peut voir chaque octet du tableau qui sera envoyé. La position une correspond à la commande ('p'), les positions 2 à 9 aux valeurs des positions et la position 10 au nombre d'images devant être prisent. Les bits de poids fort sont mis à '1'. De cette façon, on s'assure qu'aucun paquet n'est égal à zéro.

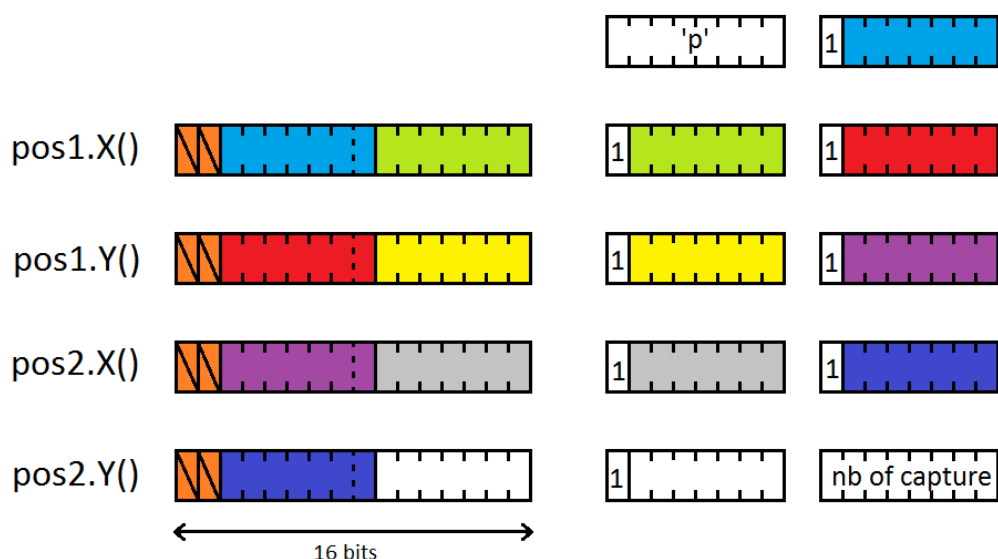


FIGURE 9.6 – Concaténation des positions

Le listing 9.2 présente le code réalisant la concaténation des positions.

Listing 9.2 – Concaténation des messages

```

1 char command[10] = {'p',
2   (char) ((pos1.x() & 0x3F80) >> 7) | 0x80,
3   (char) ((pos1.x() & 0x007F) >> 0) | 0x80,
4   (char) ((pos1.y() & 0x3F80) >> 7) | 0x80,
5   (char) ((pos1.y() & 0x007F) >> 0) | 0x80,
6   (char) ((pos2.x() & 0x3F80) >> 7) | 0x80,
7   (char) ((pos2.x() & 0x007F) >> 0) | 0x80,
8   (char) ((pos2.y() & 0x3F80) >> 7) | 0x80,
9   (char) ((pos2.y() & 0x007F) >> 0) | 0x80,
10  (char) nb_of_capture};
  
```

## 9.2.4 Réception de la position des points (micro-contrôleur)

Un légère modification de la fonction de réception (cdc\_datarx) des messages USB du micro-contrôleur a dû être faite.

Avant les modifications, les messages reçus par le micro-contrôleur avaient tous une taille de 2 bytes. Le première byte contenait la commande (voir chapitre 8.3.5) et le deuxième byte contenait le paramètre. Si le paramètre n'existe pas, le deuxième byte contient la même valeur que le premier.

Depuis la modification faite au chapitre 9.2.3, les messages reçus peuvent avoir une taille supérieur à deux. Le message contenant les positions des points comporte 10 bytes. Probablement pour des raisons de "padding" le micro-contrôleur en reçoit 14.

La fonction réceptionne les données et les stockes dans un message qui sera par la suite mis dans une queue pour être traité par la tâche "usbTask" (voir chapitre 8.3.3).

La nouvelle structure des messages est disponible au listing 9.3.

Listing 9.3 – Structure des message

```

1 struct usb_rx_msg_t {
2   uint8_t cmd;
3   uint8_t val;
4   uint8_t pos1x1;
5   uint8_t pos1x2;
6   uint8_t pos1y1;
7   uint8_t pos1y2;
8   uint8_t pos2x1;
  
```



```

9   uint8_t pos2x2;
10  uint8_t pos2y1;
11  uint8_t pos2y2;
12  uint8_t nbCapture;
13 };

```

## Récupération des points concaténés

La dé-concaténation est faite pas la tâche "usbTask" à la réception de la commande 'p'. Les valeurs sont sauvegardées dans des "uint32\_t". Le code correspondant est disponible au listing 9.4.

Listing 9.4 – Récupération des points

```

1 position1X = (msg.pos1x1 & 0x7F) << 7) | (msg.pos1x2 & 0x7F);
2 position1Y = (msg.pos1y1 & 0x7F) << 7) | (msg.pos1y2 & 0x7F);
3 position2X = (msg.pos2x1 & 0x7F) << 7) | (msg.pos2x2 & 0x7F);
4 position2Y = (msg.pos2y1 & 0x7F) << 7) | (msg.pos2y2 & 0x7F);

```

## 9.2.5 Découpe de l'image (micro-contrôleur)

Lorsque le message indiquant la position est reçu, le micro-contrôleur entre dans le cycle d'envoi en continu d'images. Il attend qu'une image soit prise par la caméra, puis il la découpe et l'envoie par USB. Il recommence ensuite le cycle en attendant la prochaine image.

La découpe d'image se fait dans la fonction "send\_part\_uncompress\_picture". Tout d'abord, les coordonnées des points sont comparées pour déterminer le point de départ en x (startX) et en y (startY). La hauteur et la largeur de l'image sont également récupérées à ce moment (height et width). Sur la figure 9.7 (page suivante) on peut voir en noir les différents éléments récupérer.

Deux boucles imbriquées permettent de découper l'image (voir figure 8.4 disposition de l'image en mémoire). Le listing 9.5 présente le code copiant uniquement la partie intéressante de l'image.

Listing 9.5 – Code découpant la partie intéressante de l'image

```

1
2   uint8_t *base = (uint8_t *)src;
3   uint8_t *dest = (uint8_t *)msg.data;
4
5
6   // skips the "startY" first lines of the picture (A)
7   base += startY*640*2;
8
9   for(countY=0; countY < height; countY++)
10  {
11      // skips the "startX" first column of the picture (B)
12      base += startX*2;
13
14      for(countX=0; countX<width*2 ; countX++)
15      {
16          *dest = *base;
17          dest ++;
18          base ++;
19      }
20
21      // skips the last pixels of the picture (C)
22      base += (640*2 - width*2-startX*2);
23
24  }

```

Le fonctionnement de ce code est relativement simple. En premier lieu, les pointeurs "base" et "dest" sont initialisés, ils pointent respectivement sur l'image entière et sur la zone dans laquelle l'image sera sauvegardée.

La ligne 7 décale le pointeur "base" à la ligne à laquelle l'image commence (A sur la figure 9.7). La ligne 12 quant à elle décale le pointeur à la colonne correcte (B sur la figure 9.7) et la ligne 22 décale le pointeur à la dernière colonne (C sur la figure 9.7).

De cette manière la ligne 16 copie uniquement la zone d'intérêt de l'image.

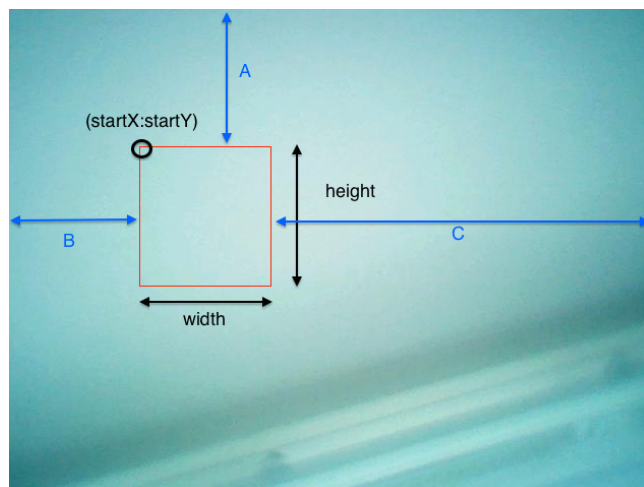


FIGURE 9.7 – En rouge : zone sélectionnée, en noir : éléments récupérés, en bleu : repère pour l'explication de la découpe d'image

Finalement, l'envoi de la partie de l'image est très simple. Il suffit d'indiquer la taille de l'image dans "msg.jpeg\_size" puis d'appeler la fonction "usbd\_cdc\_tx" en lui donnant un pointeur sur les données et la taille de celles-ci.

### 9.2.6 Réception et affichage de l'image découpée (Qt)

La réception d'une image découpée ne diffère pas de celle d'une image entière (voir chapitre 8.4.4 Fonctionnement, getImage()). L'affichage de l'image quant à lui diffère légèrement, les données reçues ne sont plus au format "jpeg" mais au format "RGB565". La méthode pour lire une telle image depuis Qt est la suivante :

```
1 QImage image(buffer, sizeX, sizeY, QImage::Format_RGB16);
```

Avant la lecture, les données sont contenues dans le buffer, sizeX et sizeY correspondent à la hauteur et à la longueur de l'image. Format\_RGB16 correspond dans Qt au format RGB565.

## 9.3 PREMIER TEST DE CAPTURE EN CONTINU

Les premiers tests d'envoi s'avèrent peu concluants. La vitesse moyenne de prise d'images est de 10.25 images/secondes. La figure 9.8 présente un extrait des mesures de vitesse réalisées.

Taille (byte)	Nb d'img	Temps (ms)	FPS (img/s)
3936	10	945	10.58
	100	9553	10.47
	250	23987	10.42
	1000	95887	10.43
64064	10	981	10.19
	10	1038	9.63
	10	1015	9.85
	1000	95907	10.43

FPS moyen : 10.25

FIGURE 9.8 – Mesure de la vitesse d'envoi des images (FPS)

On constate que la taille de l'image n'a pas une grande influence sur le taux de transfert des images. Dans le premier cas, une image de petite taille est envoyée et dans le second, c'est une image bien plus grande (environ 16 fois) qui est envoyée.

## 9.4 AUGMENTATION DU NOMBRE D'IMAGES PAR SECONDE

Selon les estimations faites au chapitre 9.1, le port USB devrait être capable de transférer environ 20 image/secondes. Le résultat de 10 est bien inférieur. Cette limitation ne provient sans doute pas de l'usb mais d'un autre élément.

### 9.4.1 Test de la fréquence de prise d'images par la caméra

Après la consultation d'un collaborateur, il s'avère qu'il est possible de configurer (à l'initialisation) la fréquence de capture de la caméra. Le paramètre à changer se trouve dans le fichier "heivs\_stm32\_camera\_mt9v114.c" à la ligne 76. Le paramètre 0x0010 doit changer sa valeur de 0x21d à 0x0125, ce qui correspond à une fréquence de capture de 20hz.

Cette modification n'atteint pas les résultats escomptés, la fréquence de prise d'images descend à 7 FPS.

Plusieurs tests ont été effectués avec différents paramètres dans l'espoir de trouver un point optimum satisfaisant. Malheureusement la fréquence maximum atteinte est d'environ 11 images par seconde, ce qui reste insuffisant pour l'application.

### 9.4.2 Changement des paramètres de DMA

Après avoir consulté un collaborateur, il semblerait que le problème provienne de la copie des images en mémoire ram. La copie de l'image depuis la caméra jusque dans la ram est gérée par le contrôleur DMA. Or, avec la configuration actuelle, le contrôleur DMA ne parvient pas à copier l'image en RAM et à la recopier en partie (uniquement la zone à découper) dans le processeur assez rapidement.

Une modification a alors été faite dans la configuration du contrôleur DMA pour qu'il copie des "morceaux" plus grands de données. En envoyant des parties plus grandes, moins de paquets sont transférés, ce qui a pour effet d'augmenter la bande-passante.

La "FIFO" a également été activée. Lorsque la caméra veut envoyer une image et que la RAM est occupée, l'opération est mise en attente dans la "FIFO" et n'est donc pas interrompue.

## 9.5 DEUXIÈME TEST DE CAPTURE EN CONTINU

Les seconds tests s'avèrent bien plus concluants. La vitesse moyenne de prise d'images est proche de 20 images/secondes.

La figure 9.9 présente les mesures de vitesse réalisées.

Taille (byte)	Nb. d'img	Temps (ms)	FPS (img/s)
21384	1000	50176	19.93
	100	5048	19.81
7040	100	5008	19.97
	1000	50125	19.95
	1000	50150	19.94
31828	100	5015	19.94
	100	5033	19.87
	1000	50176	19.93
	1000	50201	19.92
45188	100	5068	19.73
	100	5076	19.7
	1000	50176	19.93
	1000	50176	19.93
50560	100	5056	19.78

FPS moyen : 19.88

FIGURE 9.9 – Mesure 2 de la vitesse d'envoi des images (FPS)

A nouveau, la taille de l'image n'a pas une grande influence sur le nombre d'images par seconde. La moyenne est de 19.88 images/secondes, ce résultat devrait suffire pour l'algorithme déterminant la fréquence cardiaque des daphnies.

### 9.5.1 Limitation de la taille des images

Avec un taux de transfert de 19 img./sec. un problème apparaît lorsqu'une certaine taille d'image est atteinte. Lorsque la taille de l'image dépasse 45 bytes, une erreur peut survenir lors du transfert. Cette erreur survient systématiquement lorsqu'on dépasse les 51 bytes et se produit sporadiquement entre 45 et 51 bytes. Ce problème est dû à la limite de transfert du port USB. Sur le tableau de la figure 9.1 du chapitre 9.1, on constate effectivement que pour un taux de transfert d'environ 19-20 img /sec. la taille maximum d'une image est de 45 Kbytes.

Pour éviter ce problème, une modification a été effectuée dans le programme du PC. Lorsque l'utilisateur sélectionne une zone dont la taille est supérieur à 45 bytes, un message s'affiche pour lui indiquer qu'il doit sélectionner une nouvelle zone plus petite. Une taille de 45 bytes correspond à un carré de 150x150 pixels.

Il est trop tôt pour dire si cette taille est suffisante pour contenir entièrement le cœur de la daphnie. Dans les tests réalisés sur les vidéo de daphnies, la taille du cœur correspondait environ à 160x160 pixels. La taille de l'image reçue dépendra notamment du dispositif optique utilisé (lentille). Pour le moment, aucun test avec le dispositif complet n'a pu être réalisé.

### 9.5.2 Vérification de la fréquence de prise d'images

Les mesures d'images par seconde effectuées précédemment ont été calculées approximativement depuis le logiciel tournant sur le PC. Le temps a été calculé depuis l'envoi de la demande de capture en continu jusqu'à la réception de la dernière image. Avec ce temps et le nombre d'images, la fréquence de prise d'images par seconde a pu être déterminée.

Le problème de cette méthode est qu'il n'y a pas de certitude que toutes les images ont été prises à intervalle régulier. Cet intervalle représente la fréquence d'échantillonnage, il est donc important qu'il soit constant pour le traitement avec l'algorithme.

Une nouvelle modification du programme du micro-contrôleur a été réalisée. Une mesure de temps est faite à la capture de chaque image avec la fonction "timeget()". Une soustraction est effectuée entre le temps mesuré et l'ancienne mesure de temps. On obtient donc le temps entre la prise de deux images. Ce temps est sauvegardé dans un tableau. Le listing 9.6 présente le code correspondant.

Listing 9.6 – Mesure du temps entre les images

```

1  // wait a new picture
2  xEventGroupWaitBits(imgEventGroup, NEW_IMAGE_FLAG, 1, 1, portMAX_DELAY);
3
4  // save the old time
5  oldTimeout = currentTimeout;
6
7  // take the new time
8  currentTimeout = time_get();
9
10 // compute and save the diff. between the two time
11 timeCapture[nb_total_of_capture - nb_of_remaining_capture] = time_diff_ms(currentTimeout,oldTimeout);

```

Les mesures montrent que ce temps reste très stable (entre 50ms et 51ms) entre les images. La première valeur du tableau "timeValue" ne doit pas être prise en compte. En effet au premier passage, la variable "oldTimeout" ne peut pas contenir l'ancienne valeur de temps vu qu'il s'agit de la première mesure effectuée.

Une simple boucle permet alors d'afficher via le "printf" un message d'erreur lorsque la fréquence d'échantillonnage n'est plus correcte. Le listing 9.7 présente cette boucle.

Listing 9.7 – Boucle de contrôle de la fréquence d'échantillonnage

```

1  for (i = 0; i < nb_total_of_capture; i++) {
2      if ((timeCapture[i] != 50) & (timeCapture[i] != 51) & (i != 0))
3          printf("img :%d,timeCapture : %d\n", (int)i, (int)timeCapture[i]);
4  }

```

### 9.5.3 Remarque

Malgré les modifications apportées au chapitre 9.5.1, il arrive parfois qu'un problème survienne lors du transfert des images. Les images ne sont plus reçues suffisamment rapidement et un décalage apparaît. Lorsque ce problème apparaît, un message indiquant que le champ "image magic" est incorrecte.

L'origine du problème se situe peut-être dans la technique utilisée pour réceptionner les données (coté Qt.).

Ce problème se produit rarement et pour des raisons de manque de temps, il n'a pas pu être corrigé.

## Chapitre 10

# Mise en place du dispositif de capture d'images

Ce chapitre explique les différentes opérations ainsi que les tests qui ont été effectués afin de mettre en place le dispositif de capture d'images. Cela comprend, le piège, les éléments optiques et l'éclairage.

### 10.1 PIÈGE A DAPHNIE

La figure 10.1 présente le dispositif mis en place pour tenter d'immobiliser la daphnie. Une petite pompe fait circuler un faible flux d'eau depuis le réservoir jusqu' dans le piège.

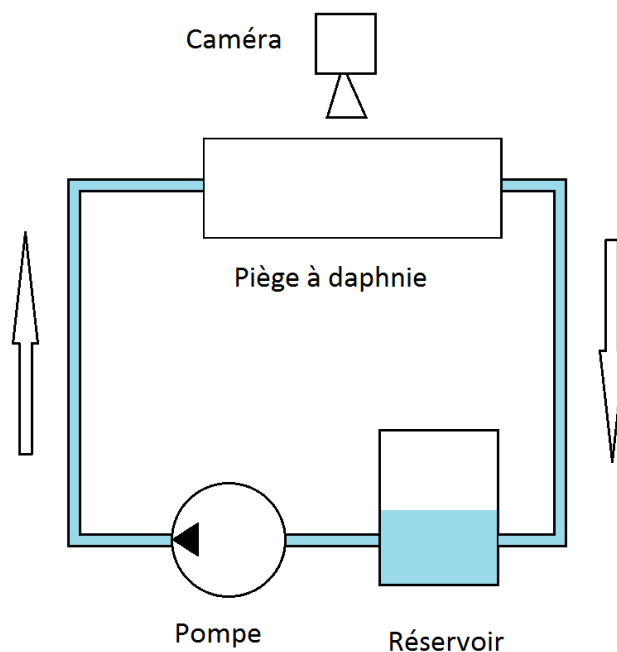


FIGURE 10.1 – Dispositif de mesure

#### 10.1.1 Pompe

La pompe est une micro-pompe piézoélectrique du type "mp6" de "microComponents". Elle est contrôlée par un circuit réalisé à la HES-SO de Sion. Cette pompe est commandée à l'aide d'impulsions, un petit afficheur

LCD ainsi que des boutons permettant de régler différents paramètres : la tension (de 220V à 60V), la fréquence des impulsions (2hz, 4hz, 6hz, 8hz, 10hz, 20hz, 30hz, ... 120hz), le nombre de pulsations et le mode de fonctionnement (continu, séquence ou impulsion). La figure 10.2 représente une photo de la pompe ainsi que de son dispositif de contrôle.

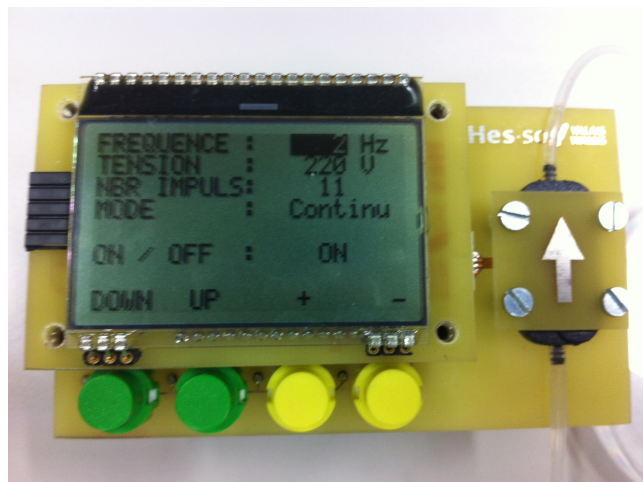


FIGURE 10.2 – Pompe

### 10.1.2 Piège

Le piège est composé de quatre parties : un support en plastique, une partie en silicone (empêchant la daphnie de bouger), une vitre en verre et enfin une tôle en aluminium appliquant une pression pour rendre le tout étanche. La figure 10.3 est une photo de coté du piège.

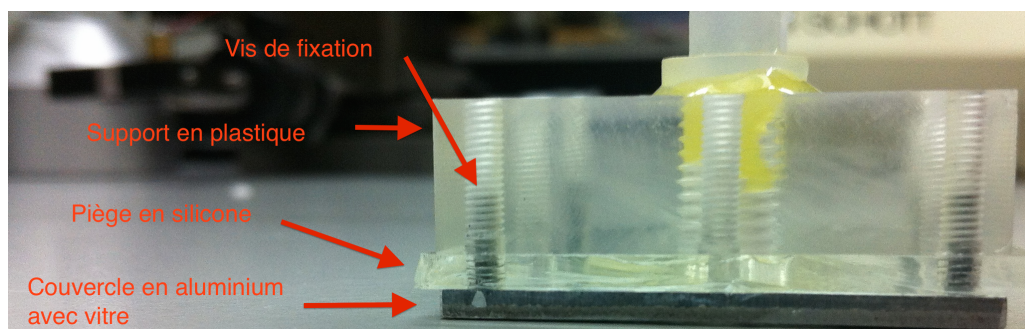


FIGURE 10.3 – Piège vu de coté



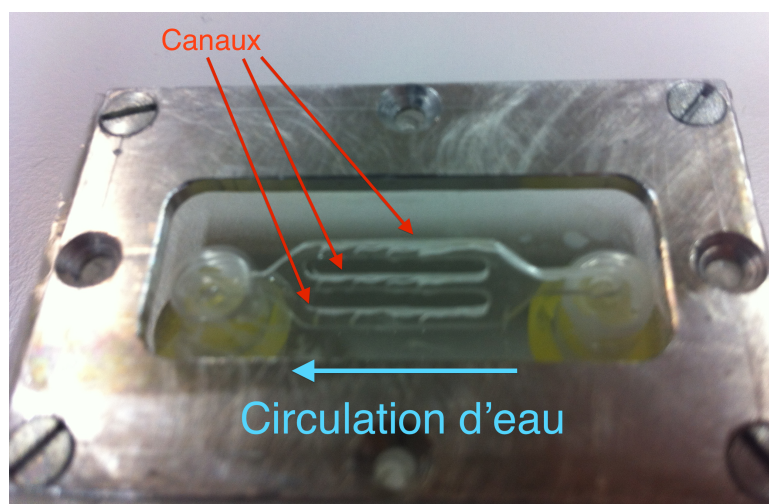


FIGURE 10.4 – Piège vu de dessous

La figure 10.4 est une photo du piège vu depuis dessous. Sur l'image, on distingue les trois canaux servant de piège. Ils ont la même largeur, mais leur profondeur diminue de droite à gauche.

La caméra filme la daphnie à travers la vitre en verre (visible à la figure 10.4). Elle est très fine ( $150\mu m$ ) et fragile à la manipulation. Ce type de vitre est très utilisée pour l'observation d'échantillons liquides au microscope.

## 10.2 CHOIX DE LA LENTILLE

Différentes lentilles ont été utilisées pour réaliser les prise de vues. Elles ont été posées au-dessus de la lentille de base de la caméra.

### Lentille 1

La première lentille testée possède une distance focale de 24mm. La figure 10.5 montre l'image obtenue avec cette lentille.



FIGURE 10.5 – Image prise avec la lentille n°1



La daphnie est beaucoup trop petite, elle mesure environ 75\*80 pixels. Il est inimaginable de pouvoir extraire le cœur d'une image si petite. Sur le film réalisé, seul les mouvements des pattes et des antennes peuvent être distingués.

## Lentille 2

La deuxième lentille testée a une distance focale de 1.80mm. La figure 10.6 montre l'image obtenue avec cette lentille.



FIGURE 10.6 – Image prise avec lentilles n°2

La daphnie est bien plus grande qu'avec la lentille précédente. Elle mesure 180\*340 pixels et la taille de son cœur est estimée à 50\*50 pixels. Le manque de contraste de l'image ne permet par contre pas de distinguer les organes de la daphnie.

Cette lentille présente les meilleurs résultats, c'est donc celle-ci qui est choisie pour la suite des mesures. Sa spécification détaillée est disponible en annexe D.1.

La prochaine étape consiste à faire différents tests d'éclairage afin d'améliorer la qualité et le contraste de l'image.

## 10.3 ÉCLAIRAGE

### 10.3.1 Positionnement de l'éclairage

L'éclairage est un élément central lorsqu'on veut prendre des images d'éléments très petits. Pour ce type de prise d'images, la meilleure façon d'éclairer est depuis l'arrière. Un éclairage de face produit beaucoup trop de reflets et empêche de distinguer quoi que ce soit.

La figure 10.7 présente le positionnement de l'éclairage par rapport au piège. La source d'éclairage utilisée est une lumière dirigée de couleur blanche standard (même couleur que les ampoules d'éclairage communes). Elle est alimentée directement depuis le secteur (230V, 50Hz). Ce 50Hz pourrait éventuellement provoquer certains problèmes lors de l'analyse avec la FFT (une fréquence de repli (aliasing) pourrait apparaître). Il faudra peut-être envisager de remplacer cet éclairage par un éclairage continu.

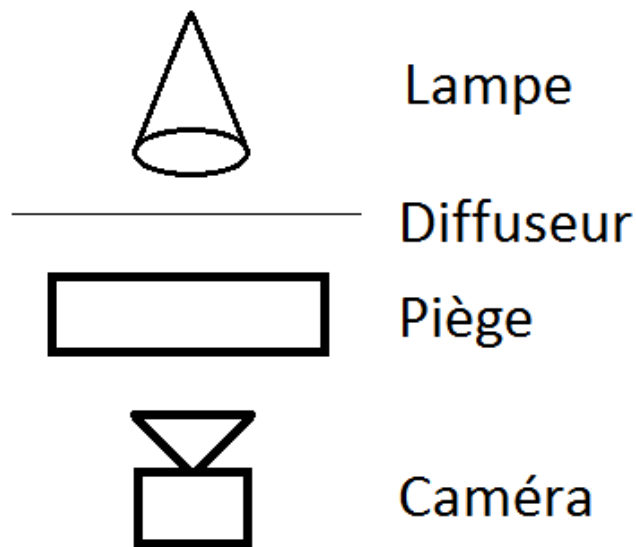


FIGURE 10.7 – Positionnement de l'éclairage

### 10.3.2 Diffuseur de lumière

Les diffuseurs de lumière s'intercalent entre la source de lumière et le sujet à éclairer (voir figure 10.7). Ils permettent, comme leur nom l'indique, de diffuser la lumière et d'éviter qu'une lumière directe n'éclaire la caméra.

Plusieurs tests ont été effectués et il s'avère que le diffuseur avec la plus grande efficacité est le .003/75 $\mu$ m.

La figure 10.8 montre une image capturée avec ce diffuseur. La qualité de l'image a augmenté, par contre il est toujours difficile de distinguer la position du cœur sur une seule image.

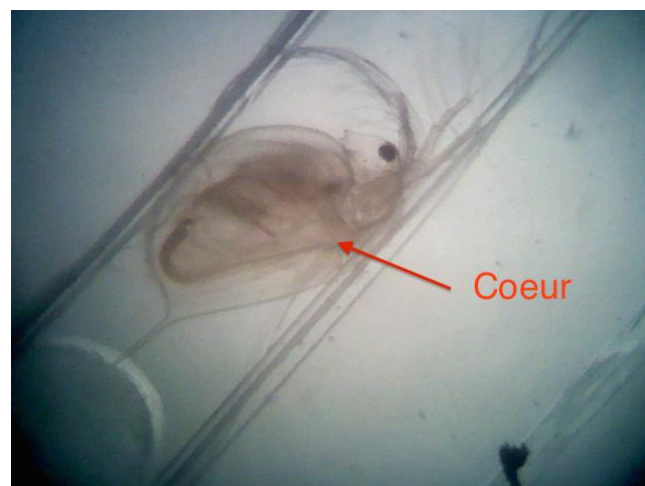


FIGURE 10.8 – Image prise avec un diffuseur

### 10.3.3 Filtre

Pour tenter d'améliorer le contraste du cœur, des filtres de couleurs ont été testés. Les filtres se placent entre la source de lumière et le diffuseur. Ils modifient la longueur d'onde de la lumière émise.

Certains éléments sont plus "réceptifs" à certaines longueurs d'ondes. Le but de ces tests est de trouver la longueur d'onde faisant au mieux ressortir le sang passant dans le cœur de la daphnie.

Malheureusement, aucun des filtres testés n'a montré l'effet attendu. La figure 10.9 montre un exemple des images obtenues avec quelques filtres.

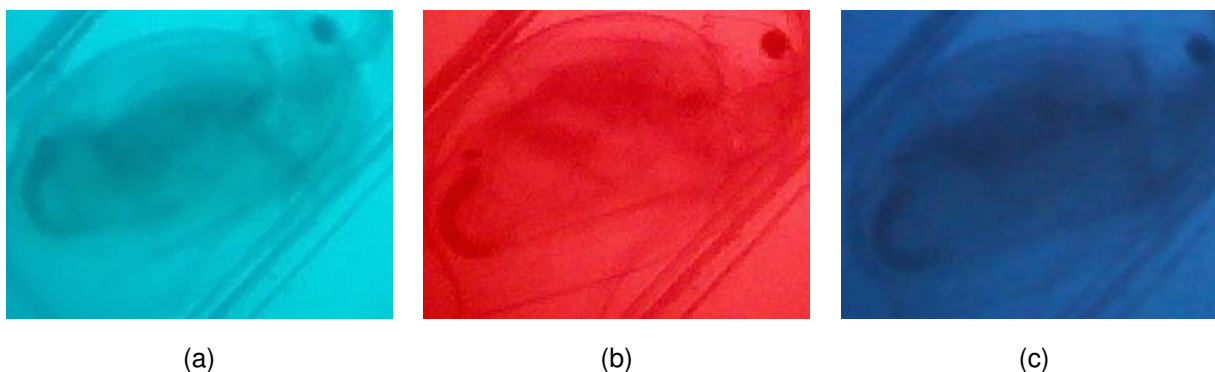


FIGURE 10.9 – Différents filtres testés

### Problème avec la caméra

Le test des ces filtres a toutefois permis de détecter un bug de la caméra. Systématiquement, lorsqu'un filtre bleu à été testé, l'image reçue était incorrecte (voir figure 10.10). L'origine de ce problème n'a pas pu être déterminée mais elle provient probablement de la caméra en elle-même.

Les images sont traitées par le micro-contrôleur comme des données brutes. Il est donc improbable que les valeurs de pixels correspondant au bleu puissent avoir une quelconque influence sur leur traitement. De plus, un "reset" de la caméra (par le biais d'une reconfiguration à l'aide du bouton "auto exposition" par exemple) permet de rétablir les images, ce qui confirme que la caméra est à l'origine du problème.



FIGURE 10.10 – Image prise incorrectement par la caméra

## Chapitre 11

# Test de l'algorithme sur les images de daphnies capturées

Ce chapitre a pour but de tester si la qualité des images prises avec la caméra est suffisante pour le bon fonctionnement des algorithmes développés précédemment.

### 11.1 PREMIÈRE SÉRIE D'IMAGE

Une première série d'images a été faite avec différentes daphnies. L'éclairage est le même que celui qui a été expliqué au chapitre 10.3, c'est à dire avec diffuseur et éclairage dirigeable.

Ces images ont été prises à une fréquence de 20 hz. Les différentes zones sélectionnées contiennent uniquement le cœur.

Le support du rapport ne se prête pas très bien à la visualisation d'images de ce type. En effet, on distingue mieux le cœur en faisant défiler les images les-unes après les autres.

La figure 11.1 tente malgré tout de montrer une image du cœur de la daphnie, le cœur étant contenu dans la zone entourée en rouge.



FIGURE 11.1 – Cœur de la daphnie

### 11.2 TEST DE L'ALGORITHME DE DÉTECTION SUR LA PREMIÈRE SÉRIE D'IMAGES

#### 11.2.1 Échantillon 1 : "Test8"

Cet échantillon comporte 356 images d'une taille de 112\*90 pixels. La figure 11.2 présente la FFT résultante de l'algorithme avec filtre (voir chapitre 5).

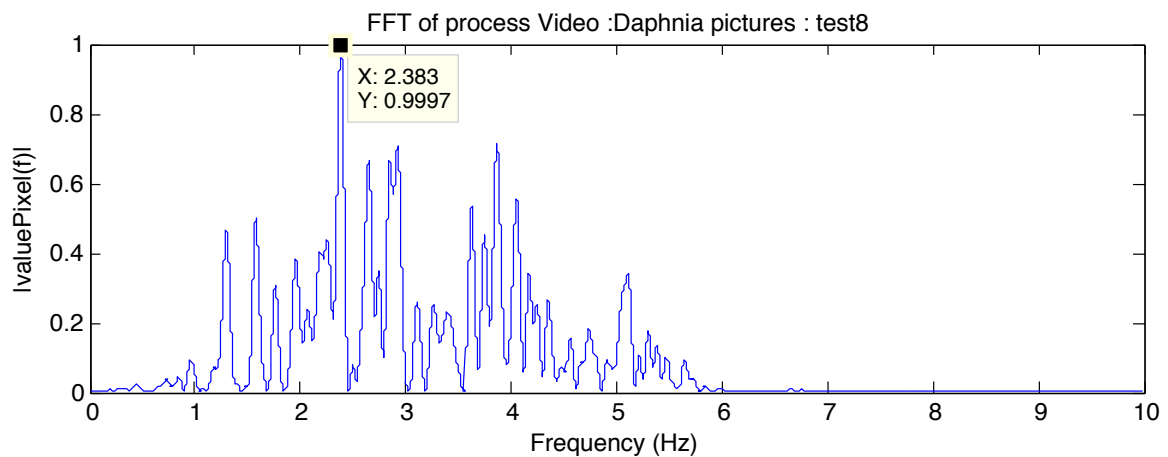


FIGURE 11.2 – FFT des échantillons "Test8" après algorithme avec filtre

Le résultat n'est pas très concluant. On obtient énormément de pics et la fondamentale (2,5hz) n'est pas la fréquence correcte du cœur (estimée à 5hz par comptage).

Pour obtenir un meilleur résultat, le contraste des images a été ajusté grâce à la fonction "imadjust" de Matlab. Le listing 11.1 montre les paramètres utilisés avec cette fonction.

Listing 11.1 – Modification du contraste grâce à Matlab

```
1 imgCut = imadjust(imgCut, [0.5 0.9], []);
```

La figure 11.3 montre une image avant la modification de contraste et la même image après.



FIGURE 11.3 – (a) sans ajustement du contraste (b) avec ajustement du contraste

L'algorithme a été modifié pour y inclure la modification du contraste et le résultat de la FFT est présenté à la figure 11.4. Ce résultat est proche du précédent, un changement de contraste sur les images n'améliore donc pas les résultats.

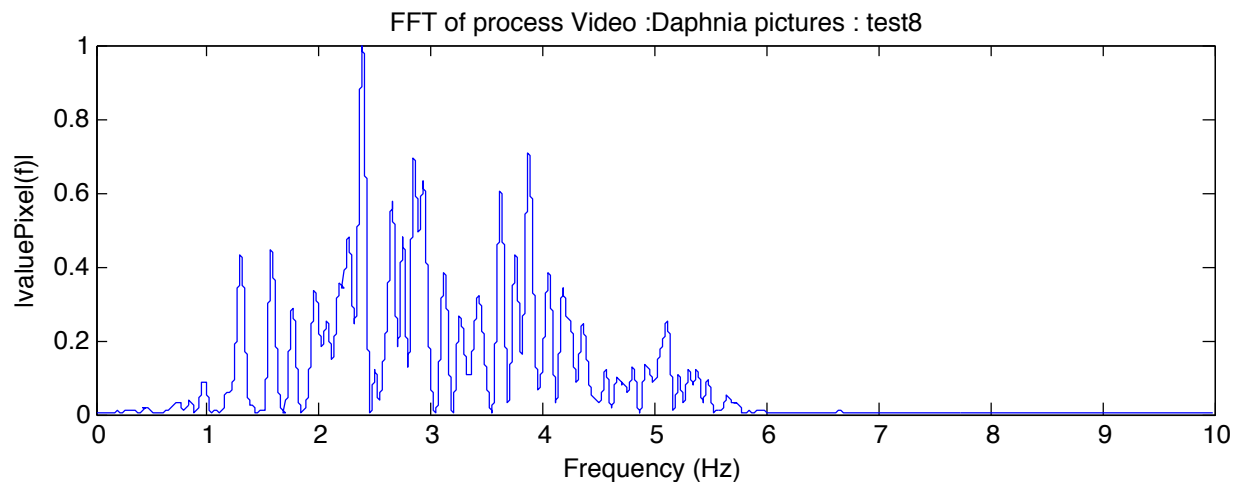


FIGURE 11.4 – FFT des échantillons "Test8" après algorithme avec filtre

### 11.2.2 Échantillon 2 : "TestCoeur"

Cet échantillon comporte 450 images également d'une taille de 112\*90 pixels. La figure 11.5 présente la FFT résultante de l'algorithme avec filtre (voir chapitre 5).

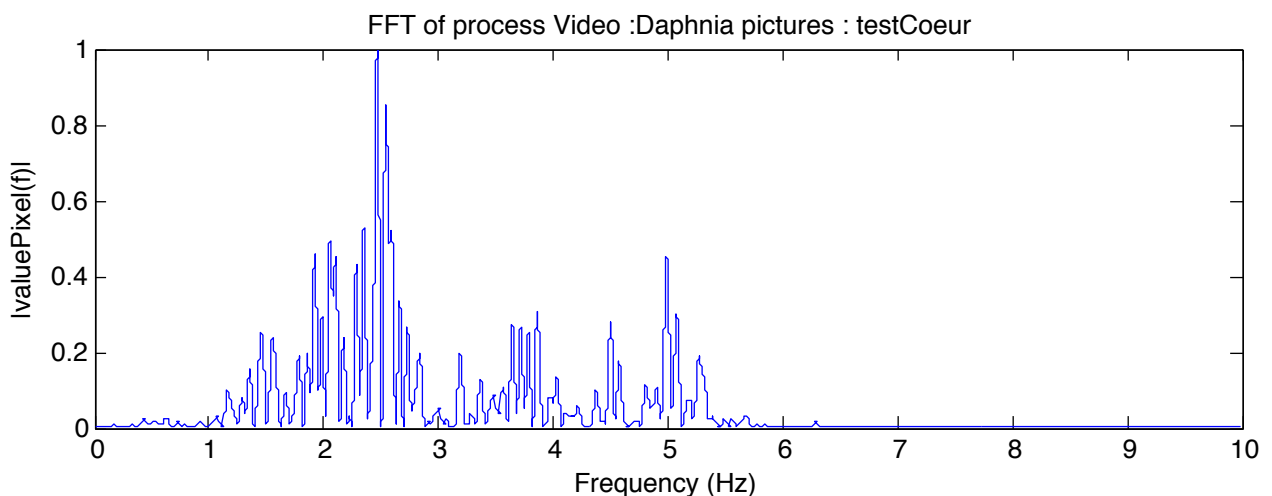


FIGURE 11.5 – FFT des échantillons "TestCoeur" après algorithme avec filtre

La fondamentale proche de 2.5hz est plus marquée que dans les échantillons précédents, mais elle ne correspond toujours pas à la fréquence des battements du cœur (environ 5hz). Par contre, on distingue une harmonique à 5Hz qui pourrait bien correspondre à la fréquence cardiaque de la daphnie.

### 11.2.3 Limitation

Un test de la fréquence de capture des images (fréquence d'échantillonnage) a été réalisé après l'analyse des échantillons 1 et 2. Il s'avère que cette fréquence n'est pas stable, elle varie entre 16hz et 20hz. Ce problème altère la validité des échantillons. Les résultats obtenus ne sont donc pas fiables.

Un autre problème existe, les images ont été enregistrées par l'application Qt. au format "jpeg". Ce format a passablement réduit la qualité des images.

De plus, en faisant défiler les images une à une, on constate que la luminosité de l'image entière varie. Ces variations sont plus prononcées sur les bords de l'image et s'atténuent au centre. Normalement on devrait observer le phénomène inverse. Le cœur est au centre de l'image, cette zone devrait avoir de plus grands changements de luminosité que les bords. Ce phénomène est peut-être dû à un mauvais éclairage.

Dans les prochaines prises d'images, il faudra tenter de changer la méthode d'éclairage afin d'obtenir un contraste meilleur sur la zone contenant le cœur.

## 11.3 MODIFICATIONS DES PROGRAMMES

Cette section traite des modifications des programmes effectuées en vu d'une deuxième prise d'images.

Tout d'abord, le programme du micro-contrôleur a été modifié pour qu'il prenne les images à une fréquence de 20hz. Le problème était dû à un "printf", utilisé pour le débogage. Il se trouvait dans la boucle de capture d'images. Un message était envoyé à travers le port USB à chaque prise d'image, ce qui créait un temps de capture inconstant. Une suppression de ce printf a résolu le problème.

De plus, pour les prochains tests, la fréquence sera surveillée à travers une console grâce à la boucle de contrôle développée au chapitre 9.5.2. Cette vérification nous permettra de s'assurer d'avoir une fréquence d'échantillonnage constante à 20hz.

La dernière modification faite concerne le programme de réception d'images Qt. Le format de sauvegarde d'images a été modifié de "jpeg" à "png". Le code concerné se trouve dans la fonction "showPartImage()". Le code modifié est le suivant :

Listing 11.2 – Sauvegarde d'une image sous Qt

```
1 image.save(m_ui->fichier->text() + QString().sprintf("/image%d.png", numeroFpics), "PNG");
```

## 11.4 DEUXIÈME SÉRIE D'IMAGES

Une deuxième série d'images a été faite avec une daphnie. Pour tenter d'augmenter le contraste, un éclairage différent a été mis en place. Le diffuseur de lumière n'a pas toujours été utilisé. Une feuille noire avec un trou au milieu a permis de diriger la lumière, cela dans le but d'éviter un éclairage direct.

La figure 11.6 montre un exemple de la qualité d'image obtenue.



FIGURE 11.6 – Daphnie de petite taille, observée sans diffuseur avec feuille noire trouée



On distingue beaucoup plus facilement le cœur. La figure 11.7 présente l'image découpée. On y discerne le cœur, la zone sombre au milieu de l'image.

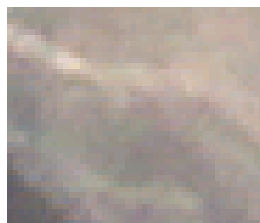


FIGURE 11.7 – Image découpée contenant le cœur de la daphnie

La daphnie utilisée était de petite taille, elle s'est bloquée vers la fin du piège et est restée particulièrement immobile durant toute les prises d'images.

En analysant attentivement les images capturées, on remarque que la daphnie bouge légèrement lorsque la pompe est activée. Au contraire, elle reste parfaitement immobile lorsque la pompe était éteinte. Plusieurs séries d'images ont été prises avec la pompe éteinte et allumée (2hz et 4hz).

## 11.5 TEST DE L'ALGORITHME DE DÉTECTION SUR LA DEUXIÈME SÉRIE

L'algorithme utilisé est celui qui implémente le filtre Butterworth. Le détail de cet algorithme est disponible au chapitre 5.

### 11.5.1 Échantillon 1 : "test1-0hz220V"

Cet échantillon comporte 1000 images d'une taille de 54\*42 pixels. Pour ce test, la pompe est éteinte. La figure 11.8 présente la somme des pixels de chaque image (voir chapitre 3.1.2). Ce signal n'est pas filtré. On y distingue deux sinus qui se superposent au signal de base, un avec une petite fréquence et l'autre avec une fréquence plus grande.

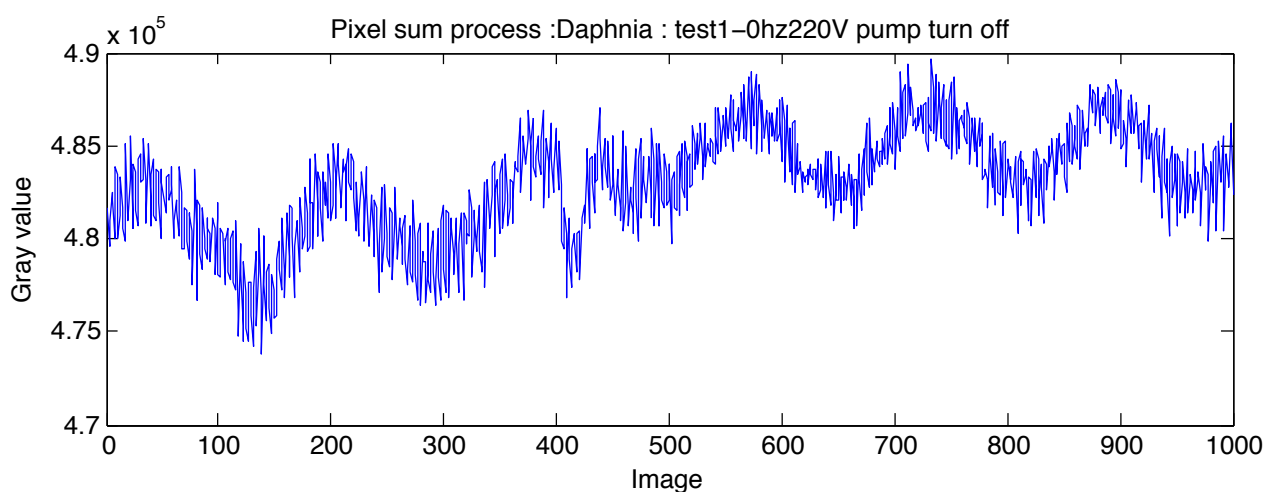


FIGURE 11.8 – Somme des pixels des échantillons "test1-ohz220v"

La figure 11.9 présente la FFT du signal des pixels sommés. Sur la FFT on distingue une composante continue (à 0hz), une harmonique proche de 0.1 hz, qui correspond au sinus de petite fréquence. On remarque



également la présence d'un petit pic proche de 4.8 hz, qui correspond probablement à la fréquence cardiaque de la daphnie.

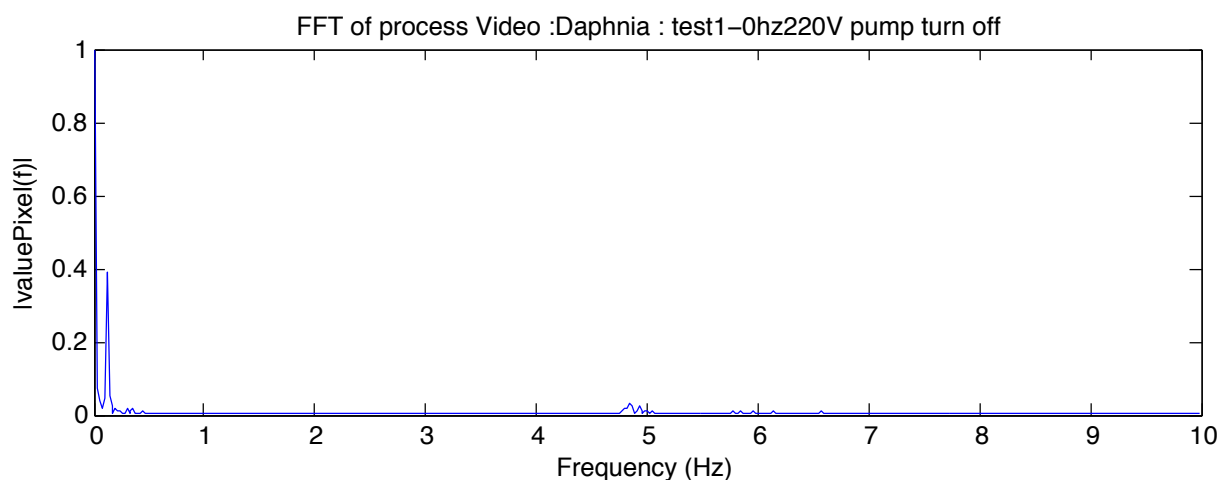


FIGURE 11.9 – FFT de la somme des pixels des échantillons "test1-0hz220v"

Le filtre développé au chapitre 5 a été appliqué sur le signal des pixels sommés. La figure 11.10 présente le résultat de la FFT du signal filtré. On constate très clairement un pic à la fréquence de 4.8hz. La fréquence cardiaque de la daphnie étant estimée à 5hz, le résultat est correct.

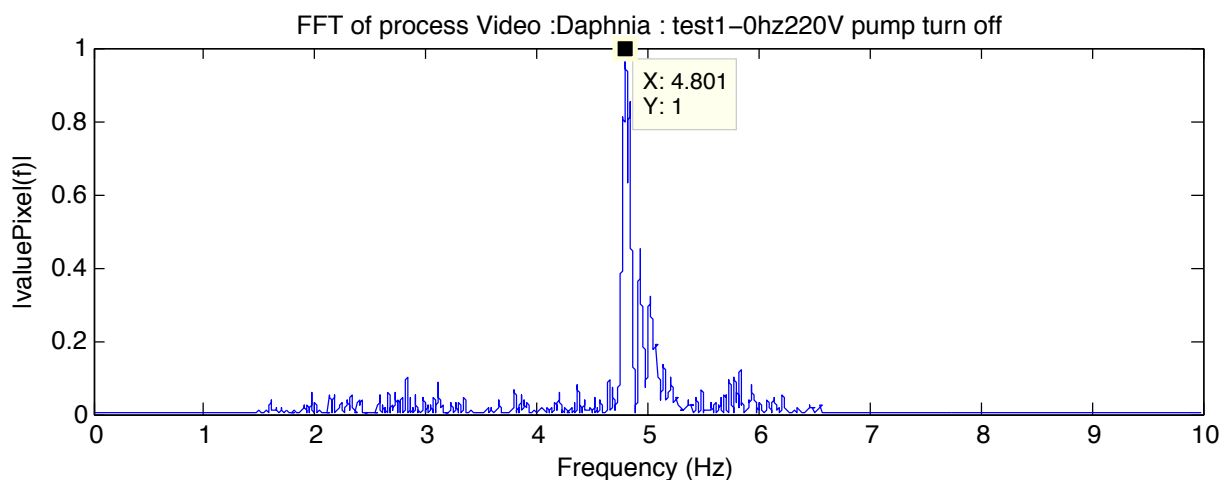


FIGURE 11.10 – FFT de la somme des pixels filtrés des échantillons "test1-0hz220v"

### 11.5.2 Échantillon 2 : "test2-0hz220vdiphnoir"

Cet échantillon comporte 1000 images d'une taille de 46\*39 pixels. Pour ce test, la pompe est éteinte. Un diffuseur de lumière ainsi qu'une feuille noire opaque trouée ont été utilisés pour éclairer la daphnie.

La figure 11.11 présente la FFT résultante de l'algorithme avec filtre (voir chapitre 5).

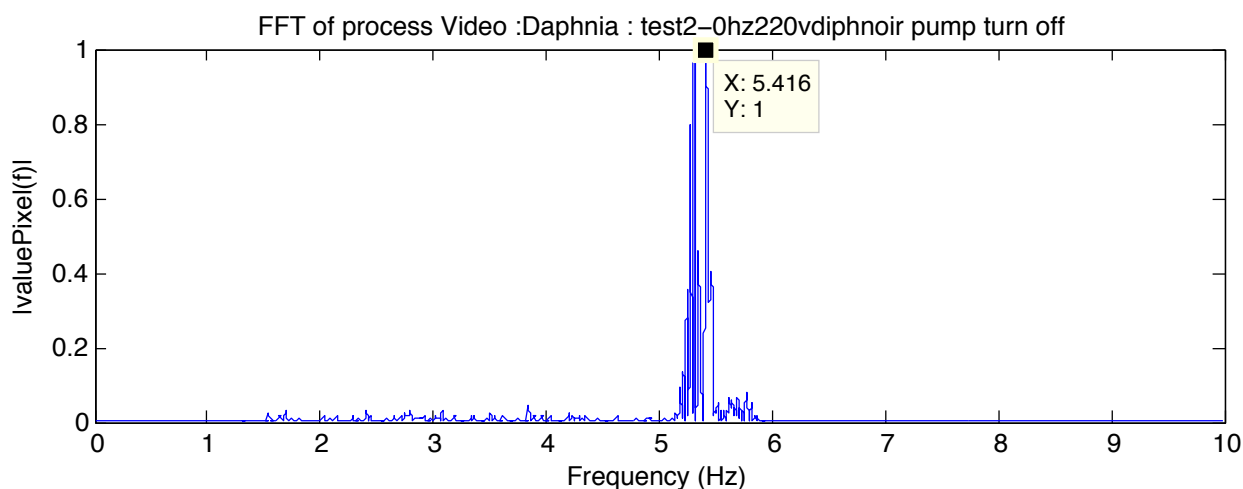


FIGURE 11.11 – FFT de la somme des pixels filtrés des échantillons "test2-0hz220vdiphnoir"

On distingue la présence de plusieurs pics proche des fréquences de 5.2 - 5.4 hz. Ceux-ci correspondent à la fréquence cardiaque de la daphnie. La durée relativement longue de la vidéo (50 secondes) peut expliquer ces nombreux pics. Durant ce laps de temps, le rythme cardiaque de la daphnie a très bien pu se modifier et passer de 5.2 à 5.4 hz ou inversement.

### 11.5.3 Échantillon 3 : "test1-2hz220v"

Cet échantillon comporte 1000 images d'une taille de 54\*42 pixels. Pour ce test, la pompe est allumée et est réglée à une fréquence de 2hz. Aucun diffuseur de lumière n'a été utilisé.

La figure 11.12 présente la FFT résultante de l'algorithme avec filtre (voir chapitre 5).

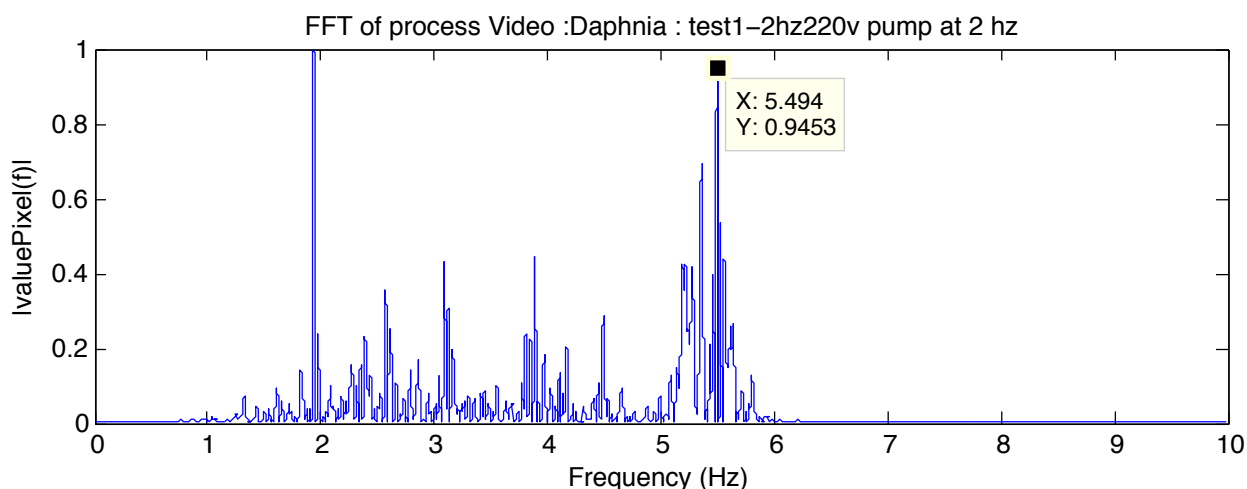


FIGURE 11.12 – FFT de la somme des pixels filtrés des échantillons "test1-2hz220v"

Sur ce graphe, on constate que la fondamentale se situe à 2hz et l'harmonique la plus importante à environ 5.5hz.

Le pic à 2 hz correspond à la fréquence de la pompe. En regardant les images manuellement, on constate que la daphnie bouge de façon régulière. Ce mouvement est dû à la pompe.

Le pic de la fréquence cardiaque de la daphnie est tout de même visible à 5.5hz.

### 11.5.4 Échantillon 4 : "test2-4hz220v"

Cet échantillon comporte 1000 images d'une taille de 72\*43 pixels. Pour ce test, la pompe est allumée et est réglée à une fréquence de 4hz. Aucun diffuseur de lumière n'a été utilisé.

La figure 11.13 présente la FFT résultante de l'algorithme avec filtre (voir chapitre 5).

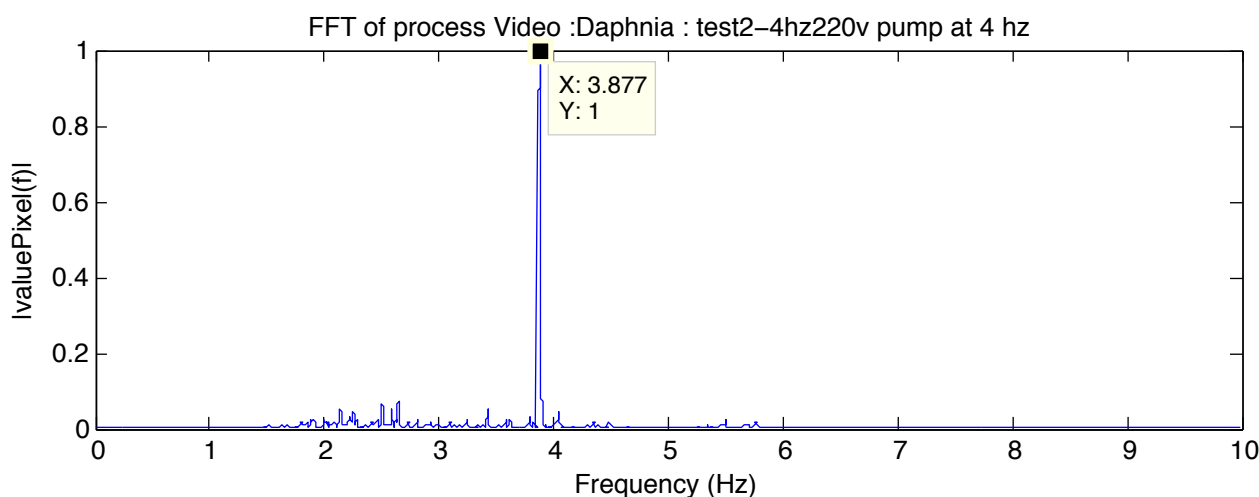


FIGURE 11.13 – FFT de la somme des pixels filtrés des échantillons "test2-4hz220v"

Sur ce graphe, on constate que la fondamentale se situe à 3.88hz. Cette fréquence n'est pas celle du cœur de la daphnie mais celle de la pompe. Le cœur de la daphnie peut battre entre 2.5 et presque 6hz. Avec la pompe enclenchée à 4hz il est impossible de déterminer si le pic est dû à la pompe ou à la daphnie.

### 11.5.5 Synthèse et vérification

Un comptage manuel des battements de cœur a été fait, le résultat est disponible en annexe C.1. Les échantillons sont relativement longs, c'est pourquoi le comptage c'est fait par tranches d'environ 200 images.

Les fréquences obtenues lors du comptage sont visible à la figure 11.14. La dernière colonne représente la fréquence moyenne obtenue sur les 1000 images.

On constate que ces fréquences sont très proches et correspondent à celles vues dans la FFT aux figures précédentes.

**Echantillon 1 : "test1-0hz220V"**

Nb image :	371	246	198	185	1000
Fréquence :	4.85	5.20	6.06	5.41	5.28

**Echantillon 2 : "test2-0hz220vdiphnoir"**

Nb image :	240	258	298	204	1000
Fréquence :	5.00	5.43	5.37	5.49	5.32

**Echantillon 3 : "test1-2hz220v"**

Nb image :	263	230	406	101	1000
Fréquence :	5.32	5.39	5.12	5.15	5.24

FIGURE 11.14 – Fréquence obtenue par comptage

**Remarque**

On a pu constater qu'en adaptant l'éclairage pour obtenir un contraste plus élevé, il est possible d'extraire la fréquence cardiaque de la daphnie avec l'algorithme développé précédemment.

On remarque également que lorsque la pompe est enclenchée, la daphnie a tendance à bouger beaucoup plus. La fréquence de la pompe apparaît également sur la FFT.

Une fréquence de la pompe de 2hz peut-être neutralisée grâce à un filtre. Par contre, une fréquence de 4hz est quasiment impossible à éliminer. De plus, l'utilisation d'une fréquence supérieure à la fréquence d'échantillonnage / 2 (10 hz) pourrait faire apparaître une fréquence de repliement (aliasing) sur la FFT.

Il est donc souhaitable que la pompe soit éteinte (ou réglée sur 2hz) lors des mesures. Malheureusement, cela risque de ne pas être toujours possible. La daphnie observée était plus ou moins immobile même lorsque la pompe était éteinte, mais cela ne sera pas toujours le cas. Parfois la daphnie tente de remonter le piège à l'envers lorsque le flux d'eau est éteint.

## Chapitre 12

# Améliorations de l'algorithme Matlab

Ce chapitre traite des différentes modifications et améliorations réalisées sur les algorithmes Matlab permettant de détecter la fréquence cardiaque des daphnies. Ces modifications sont réalisées en vue de l'implémentation de l'algorithme sur le micro-contrôleur.

### 12.1 SUPPRESSION DU FILTRE

L'algorithme développé au chapitre 5 dispose d'un filtre. Mais, en réalité, ce dernier n'est pas nécessaire. Il est appliqué après l'échantillonnage (transformation du signal analogique (lumière) en signal numérique par la caméra) ce qui veut dire qu'il ne permet pas de supprimer les éventuelles fréquences de repliement (aliasing).

Si certaines fréquences doivent être supprimées, il suffit de les ignorer dans le spectre des fréquences. En effet, le spectre des fréquences de la FFT affiche toutes les fréquences contenues dans le signal. Étant donné que l'on connaît la gamme de fréquence des battements du cœur de la daphnie (entre 2hz et 6hz), il suffit simplement d'ignorer celles qui ne sont pas contenues dans cet intervalle.

### 12.2 AMÉLIORATION DE LA FENÊTRE GLISSANTE

L'implémentation de la fenêtre glissante réalisée au chapitre 6 est relativement complexe, notamment à cause du retard généré par le filtre (voir chapitre 6.1.1).

Dans cette nouvelle implémentation, le filtre n'existe plus. Le code Matlab qui déplace la fenêtre est donc grandement simplifié, il est visible au listing 12.1.

Listing 12.1 – Fenêtre glissante simplifiée

```
1 index1 = 1;
2
3 % create the window filled with zeros
4 window = zeros(1,sizeSWimg);
5
6 while(index1 < length(newValuePixel)+1)
7
8     % move window
9     window = [window,newValuePixel(index1)];
10    window(1) = [];
11    index1 = index1 + 1;
12
13    % processing on window
14    %...
15
16 end;
```

Sur le listing 12.1, à la ligne 6, on crée un nouveau tableau avec, en dernière position, la valeur entrante dans la fenêtre. A la ligne 7, on supprime la première valeur du tableau, on fait donc sortir de la fenêtre la valeur la plus ancienne. La fenêtre se déplace.

## 12.3 FFT

### 12.3.1 Résolution et fenêtre de "Hann"

La résolution de la FFT a posé un certain nombre de problèmes. En effet, elle dépend de la taille de l'entrée fournie. Dans le cas de la fenêtre glissante, une FFT est réalisée sur chaque fenêtre, et donc la taille de l'entrée est égale à celle de la fenêtre.

Une fenêtre de 100 images correspond à 5 secondes (20 img/sec), ce qui fait environ 25 battements. Avec une entrée de 100 images, la résolution de la FFT est insuffisante (environ 0.2hz).

Pour augmenter la résolution il existe une solution déjà utilisée au chapitre 5.1.2. La fonction FFT de Matlab peut accepter, en deuxième paramètre, la taille du signal d'entrée. Si la taille indiquée est supérieure à la taille réelle du signal, une série de '0' est rajoutée au signal de base pour atteindre la taille souhaitée ("padding").

Le signal de base possède une composante verticale importante (offset), rajouter une série de zéro après ce signal va créer un échelon important. Le signal va passer brutalement à zéro. Pour éviter ce phénomène, une fenêtre "Hann" est appliquée au signal d'entrée. Elle modifie le signal pour qu'il commence et se termine proche de zéro. Le choix de cette fenêtre ainsi que le détail de son fonctionnement sont disponibles au chapitre 13.1.3.

Le listing 12.2 présente le script Matlab créant la fenêtre de Hann, le listing 12.3 présente quant à lui le scripte réalisant la FFT.

Listing 12.2 – Création de la fenêtre de Hann

```

1 windowHanning = zeros(1,sizeSWimg);
2
3 for x = 1 : sizeSWimg
4     windowHanning(x) = 0.5*(1-cos((2*pi*x/(sizeSWimg-1))));
5 end;
```

Listing 12.3 – FFT de la fenêtre glissante

```

1 % apply Hann window
2 for x = 1: sizeSWimg
3     windowToFFT(x) = window(x) * windowHanning(x);
4 end;
5
6 % FFT
7 Y = 0;
8 power = 0;
9 freq = 0;
10 Y = fft(windowToFFT,sizeSWimg*100);
11 Y(1) = [];
12
13 n=length(Y);
14
15 % take the half abs value of the fft complex
16 % and make the square of this value
17 power = abs(Y(1:floor(n/2))).^2;
18
19 % build the vector for the x axis
20 nyquist = 1/2;
21 freq = (1:n/2) / (n/2)*nyquist;
22 freq = freq*Fs;
```

## Remarque

L'utilisation d'une fenêtre n'était pas nécessaire lors de la première implémentation de la fenêtre glissante au chapitre 6. Le signal était filtré, le filtre avait pour effet de supprimer la composante continue. Le signal était donc centré par rapport à 0. Il n'y avait pas de problème d'échelons lors que les zéros étaient ajoutés.

### 12.3.2 Suppression des fréquences sans intérêt

Comme expliqué au chapitre 12.1, l'utilisation d'un filtre n'est plus nécessaire. Il suffit de supprimer les fréquences n'ayant pas d'intérêt. Cette opération est relativement simple, il faut mettre à zéro les résultats de la FFT correspondant aux fréquences plus basses que 2,5 Hz et plus hautes que 8 Hz.

Le listing 12.4 présente le scripte réalisant cette opération.

Listing 12.4 – Supression des fréquences sans intérêt

```
1 fLowCut = 2.5;  
2 fHighCut = 8;  
3  
4 %FLowCut  
5 power(1:floor(fLowCut*(length(power)/10))) = 0;  
6  
7 %FHighCut  
8 power(floor(fHighCut*(length(power)/10):length(power))) = 0;
```

## 12.4 ALGORITHME FENÊTRE GLISSANTE

Le reste de l'algorithme n'a que peu été modifié par rapport à celui développé au chapitre 6. Les maximums sont toujours récupérés et stockés de la même manière. Le scripte complet de l'algorithme de la fenêtre glissante est disponible sur le CD-ROM sous le nom de "slidingWindowV2.m".

## 12.5 TEST DU NOUVEL ALGORITHME

Cette section présente le résultat des tests de l'algorithme de la fenêtre glissante améliorée. Ces tests ont été effectués sur les images du chapitre 11.4.

Les images sont prises sur une durée de 50 secondes et la fréquence de prise d'images est de 20 hz.

### Échantillon 1 : "test1-0hz220V"

La figure 12.1 présente le résultat du nouvel algorithme de la fenêtre glissante. On y voit la fréquence(axe y) qui évolue au cours du temps (axe x). Le temps est exprimé en images.

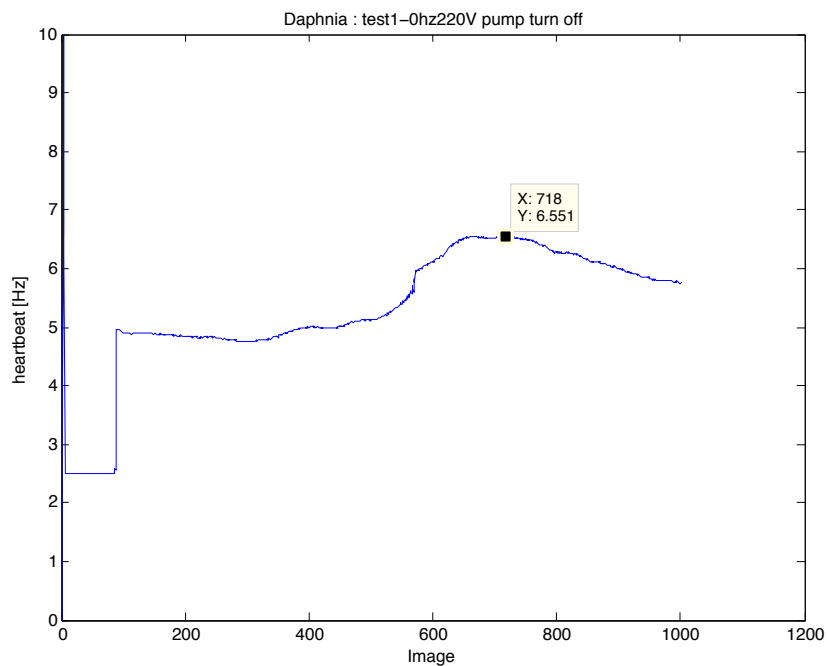


FIGURE 12.1 – Fréquence en fonction du temps

Sur le graphique, on constate que le résultat met 100 images (taille de la fenêtre) avant d'être correct. Si on analyse ce résultat avec la FFT faite au chapitre 11.5 on constate que les valeurs obtenues sont correctes.

### Échantillon 2 : "test2-0hz220vdiphnoir

La figure 12.2 présente le résultat de l'algorithme de la fenêtre glissante.

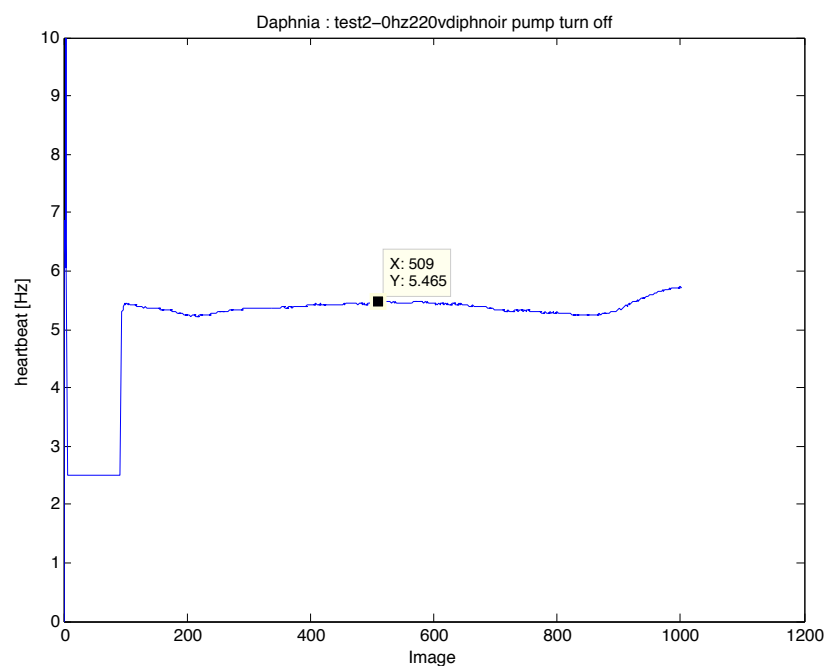


FIGURE 12.2 – Fréquence en fonction du temps

Si on analyse ce résultat avec la FFT faite au chapitre 11.5 on constate que les valeurs obtenues sont correctes.



### Échantillon 3 : "test1-2hz220v"

La figure 12.3 présente le résultat de l'algorithme de la fenêtre glissante.

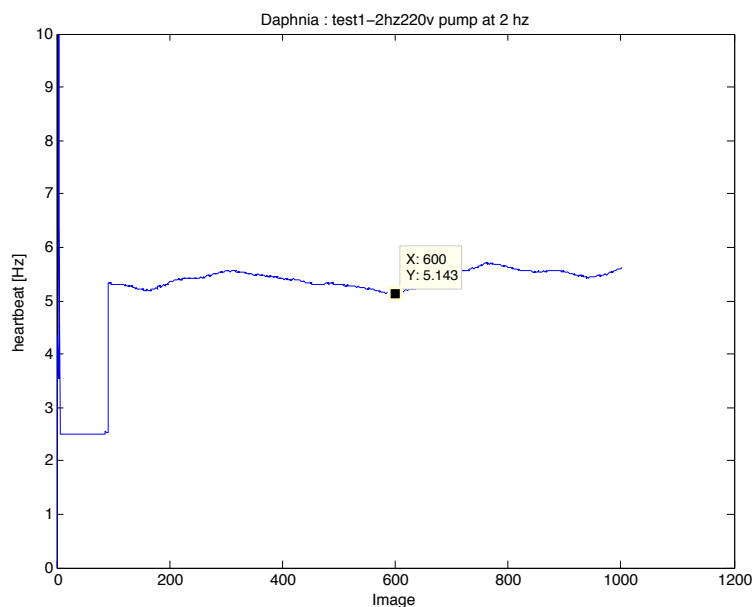


FIGURE 12.3 – Fréquence en fonction du temps

Si on analyse ce résultat avec la FFT faite au chapitre 11.5 on constate que les valeurs obtenues sont également correctes.

### Échantillon 4 : "test2-4hz220v"

La figure 12.4 présente le résultat de l'algorithme de la fenêtre glissante. Dans cette vidéo, la pompe est enclenchée à 4hz. On constate à nouveau que le résultat obtenu est la fréquence de la pompe.

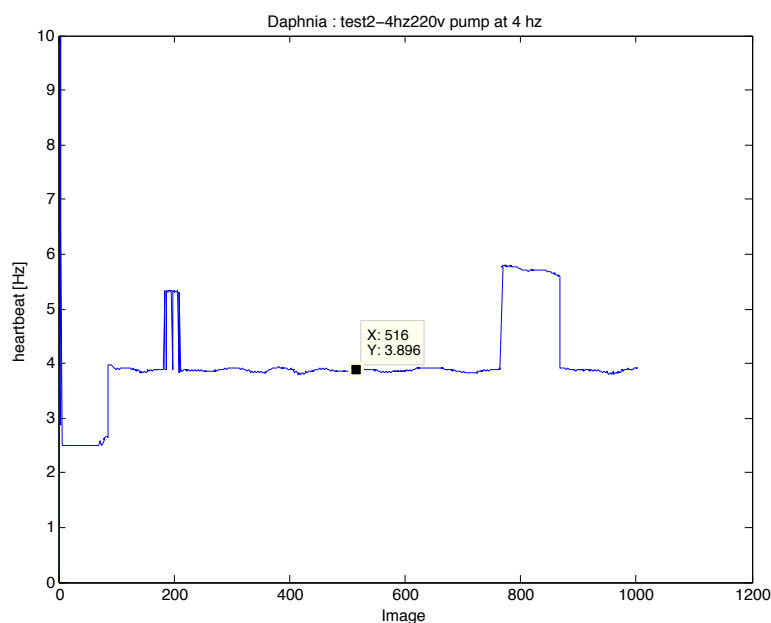


FIGURE 12.4 – Fréquence en fonction du temps

## Chapitre 13

# Implémentation des fonctions basiques de l'algorithme sur le micro-contrôleur

Ce chapitre a pour but de détailler l'implémentation des fonctions basiques de l'algorithme sur le micro-contrôleur. Il abordera également le test de ces différentes fonctions.

Les fonction suivantes seront abordées :

- ♦ FFT sur le micro-contrôleur
- ♦ Conversion des images en noir et blanc et somme des pixels

### 13.1 FFT SUR LE MICRO-CONTRÔLEUR

#### 13.1.1 Librairie utilisée

La librairie "TransformFunctions" du paquet "CMSIS-DSP" est utilisée pour réaliser les FFT. Les librairies "CMSIS-DSP" sont des librairies standardisées et optimisées pour fonctionner sur les Cortex-M0, Cortex-M3 et Cortex-M4. Elles mettent à disposition plus de 60 fonctions permettant de réaliser des opérations sur les nombres à virgule flottante (float, q7, q15, q31).<sup>1</sup>

La version utilisée est celle optimisée pour le Cortex-M4.

#### 13.1.2 Fonctions et paramètres utilisés

Dans cette application, les données transmises à la FFT seront des réelles de type float32\_t.

Tout d'abord, il faut déclarer deux structures nécessaires à la réalisation de la FFT :

```
1 am_rfft_instance_f32 fft_instance;  
2 am_cfft_radix4_instance_f32 fft_instance_radix4;
```

Puis il faut initialiser la FFT avec la fonction suivante :

```
1 arm_status arm_rfft_init_f32(  
2     arm_rfft_instance_f32 * S,  
3     arm_cfft_radix4_instance_f32 * S_CFFT,  
4     uint32_t fftLenReal,  
5     uint32_t ifftFlagR,  
6     uint32_t bitReverseFlag)
```

1. Source : <http://www.arm.com/products/processors/cortex-m/cortex-microcontroller-software-interface-standard.php> consulté le 30.06.14

Les deux premiers paramètres correspondent aux structures créées précédemment. Le paramètre "fftLenReal" correspond à la longueur de la FFT, il peut prendre comme valeurs 128, 512 ou 2048. Pour les tests, la valeur de 2048 a été choisie. Le paramètre "ifftFlagR" indique si l'on souhaite faire une FFT ou l'inverse d'une FFT (iFFT). Dans notre cas, nous devons faire une FFT, donc la valeur '0' est utilisée. Le dernier paramètre "bitReversFlag" concerne l'inversion des valeurs de sortie, pour obtenir des valeurs non-inversées, il faut mettre cette valeur à '1'.

La fonction réalisant la FFT peut ensuite être appelée :

```

1 void arm_rfft_f32(
2   const arm_rfft_instance_f32 * S,
3   float32_t * pSrc,
4   float32_t * pDst)

```

Le premier paramètre est la structure "arm\_rfft\_instance\_f32" déclarée précédemment. Le paramètre "\*pSrc" est un pointeur sur le tableau de float32\_t contenant les valeurs du signal d'entrée. La taille de ce tableau doit être la même que le paramètre "fftLenReal" vu plus haut. Le paramètre "\*pDst" est également un pointeur sur un tableau de float32\_t. Ce tableau contiendra le résultat complexe de la FFT, il doit avoir une taille de "fftLenReal" \* 2. La sortie sera sous la forme suivante : real(0), imag(0), real(1), imag(1), ....

Un extrait de la documentation Doxygene est disponible en annexe D.2.

### 13.1.3 Fenêtre du signal

Les séries de Fourier s'appliquent sur un signal théorique infini, ce type de signal est inexistant dans la pratique. Les transformées de Fourier s'appliquent quant à elles sur un signal périodique, mais de nombreux signaux ne le sont pas. C'est à ce moment qu'apparaissent les fenêtres. Elles se multiplient au signal afin de le rendre périodique. Elles diminuent le signal symétriquement au début et à la fin, selon une fonction prédéterminée.

Une fenêtre de "Hann" a été utilisée. Elle a été choisie car elle est couramment utilisée, simple d'implémentation et qu'elle démarre à zéro. De plus, sa réponse fréquentielle semble adaptée à l'application. La figure 13.1 représente la fonction.

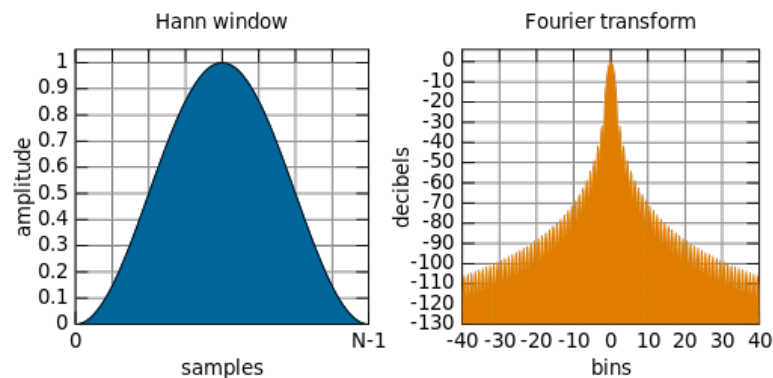


FIGURE 13.1 – Hann fonction et sa réponse fréquentielle<sup>2</sup>

La fenêtre est donnée par le calcul suivant :

$$\omega(n) = 0.5 * (1 - \cos(\frac{2 * \pi * n}{N - 1}))$$

2. Source image : [http://en.wikipedia.org/wiki/Hann\\_function](http://en.wikipedia.org/wiki/Hann_function), consulté le 30.06.14

### 13.1.4 Environnement de test

Un certain nombre de tests ont été effectués pour vérifier le bon fonctionnement et la bonne configuration de la FFT. Afin de pouvoir la contrôler, il a fallu trouver un moyen pour récupérer le tableau contenant les résultats.

La méthode la plus rapide à mettre en place consiste à afficher chaque valeur de sortie à l'aide d'un "printf" à travers le port série USB. La méthode "printf" permet l'affichage d'un nombre à virgule flottante à l'aide du symbole "%f".

Malheureusement, lors de l'utilisation de cette méthode, le programme du micro-contrôleur plante. Il s'avère que faire un "printf" d'un nombre à virgule flottante nécessite une grande quantité de "stack". Une tentative a été faite pour augmenter la mémoire "stack" de la tâche mais elle était toujours insuffisante.

L'alternative trouvée pour résoudre ce problème consiste à faire un changement de type (cast) en "uint32\_t" avant l'affichage par le "printf". Le code correspondant est disponible au listing 13.1.

Listing 13.1 – affichage d'un float32\_t grâce à un printf

```

1  for (j = 0; j < LENGTH_FFT * 2; j++)
2  {
3      printf("%d\n", (uint32_t) outputFFT_f32[j]);
4  }

```

Les données récupérées sont alors écrites dans un fichier, puis traitées avec un script Matlab.

Ce script est relativement simple. Il récupère les valeurs, calcule les modules des nombres complexes ( $\sqrt{Re^2 + Im^2}$ ), et supprime la composante DC du signal (première et deuxième valeurs du tableau). Ensuite il crée la base de temps correcte en fonction de la fréquence d'échantillonnage et de la longueur de la FFT. Finalement, il affiche le résultat dans une figure.

Ce script est disponible sur le CD-ROM en annexe sous le nom de "readDataShowFFT.m".

### 13.1.5 Test

#### Simple sinus

Tout d'abord, un simple sinus d'une fréquence de 30hz a été testé. Le code correspondant est le suivant :

Listing 13.2 – Sinus 30hz avec fenêtre de Hann

```

1  #define LENGTH_FFT 2048
2
3  float32_t inputTest_f32[LENGTH_FFT];
4  float32_t f1 = 30;
5
6  uint32_t j;
7
8  for (j = 0; j < LENGTH_FFT; j++)
9  {
10
11      inputTestFFT_f32[j] = (0.5*(1-cos((2 * 3.1415 * j)/(LENGTH_FFT-1)))) *
12                          (100.0 + 100.0* sin((2 * 3.1415 * j * f1)/ LENGTH_FFT));
13  }

```

La figure 13.2 représente la forme du signal à l'entrée et le résultat de la FFT. La fondamentale se situe à la fréquence de 30 hz.

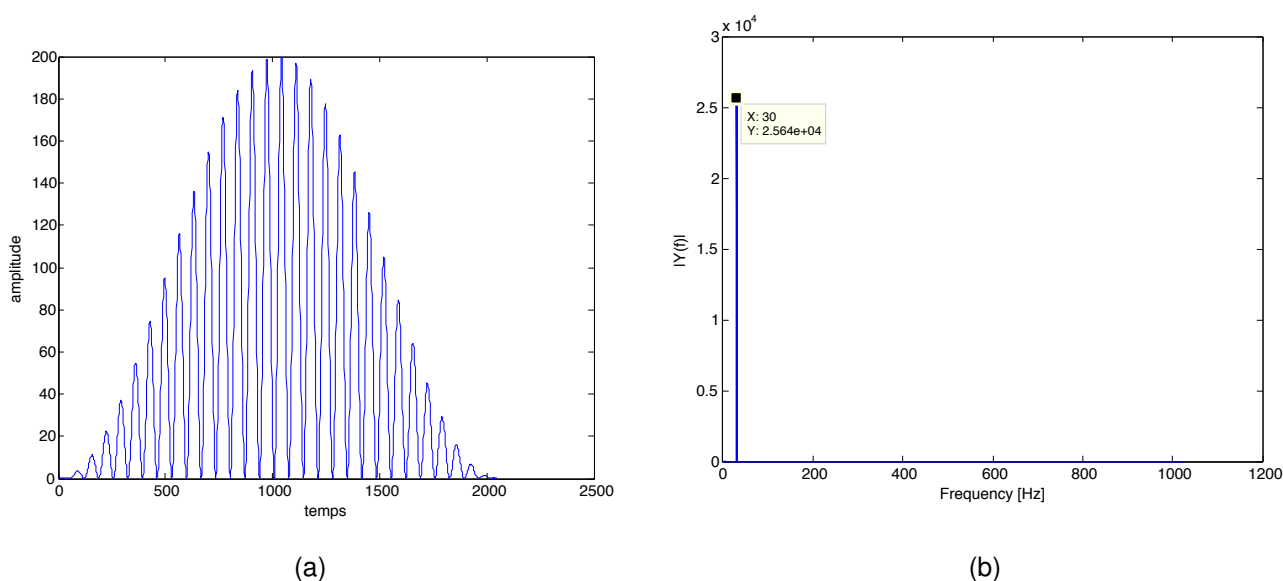


FIGURE 13.2 – (a) sinus à 30hz avec fenêtre de Hann (b) résultat de la FFT

### Triple sinus

Pour le second test, trois sinus de fréquence (360hz, 60hz, et 120hz) et d'amplitude (20,50,100) différentes ont été utilisées. Le code correspondant à la fonction est le suivant :

Listing 13.3 – Sinus 60hz 120hz 360hz avec fenêtre de Hann

```
1 inputTestFFT_f32[j] = (0.5*(1-cos((2*3.1415*j)/(LENGTH_FFT-1))))*  
2 (300.0  
3 + 20.0* sin((2 * 3.1415 * j * f1) / LENGTH_FFT)  
4 + 50.0 * sin((2 * 3.1415 * j * f2) / LENGTH_FFT)  
5 + 100.0 * 3sin((2 * 3.1415 * j * f3) / LENGTH_FFT));
```

La figure 13.3 représente la forme du signal à l'entrée et le résultat de la FFT.

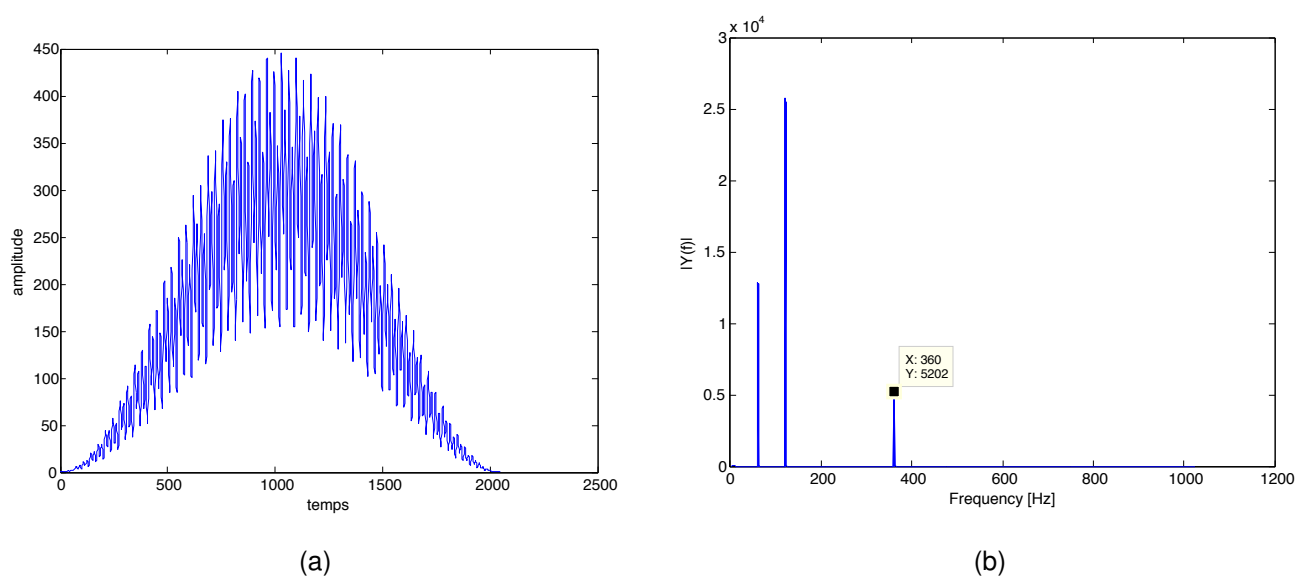


FIGURE 13.3 – (a) 3 sinus (60,120 et 360hz) superposés avec fenêtre de Hann (b) résultat de la FFT

Le résultat de la FFT est détaillé dans le tableau 13.1. On constate que les valeurs d'amplitude lues sur le graphique de la FFT sont proportionnelles aux amplitudes des sinus.

TABLE 13.1 – Résultat

	Fréquence	Amplitude	Valeur FFT
Sinus 1	120	100	25748
Sinus 2	60	50	12840
Sinus 3	360	20	5202

### Sinus et bruit aléatoire

Pour ce test, un bruit est superposé au signal de l'entrée. Ce bruit est créé grâce à la fonction "rand() %80". Cette fonction renvoie de façon "pseudo" aléatoire un nombre en 0 et 80, qui est additionné au signal. Le code correspondant à la fonction est le suivant :

Listing 13.4 – Sinus 360hz bruit aléatoire et fenêtre de Hann

```

1 inputTestFFT_f32 = (0.5*(1-cos((2*3.1415*j)/(LENGTH_FFT-1))))*
2 (200.0
3   + 100.0* sin((2 * 3.1415 * j * f1)/ LENGTH_FFT)
4   + rand()%80);

```

La figure 13.4 représente la forme du signal à l'entrée et le résultat de la FFT.

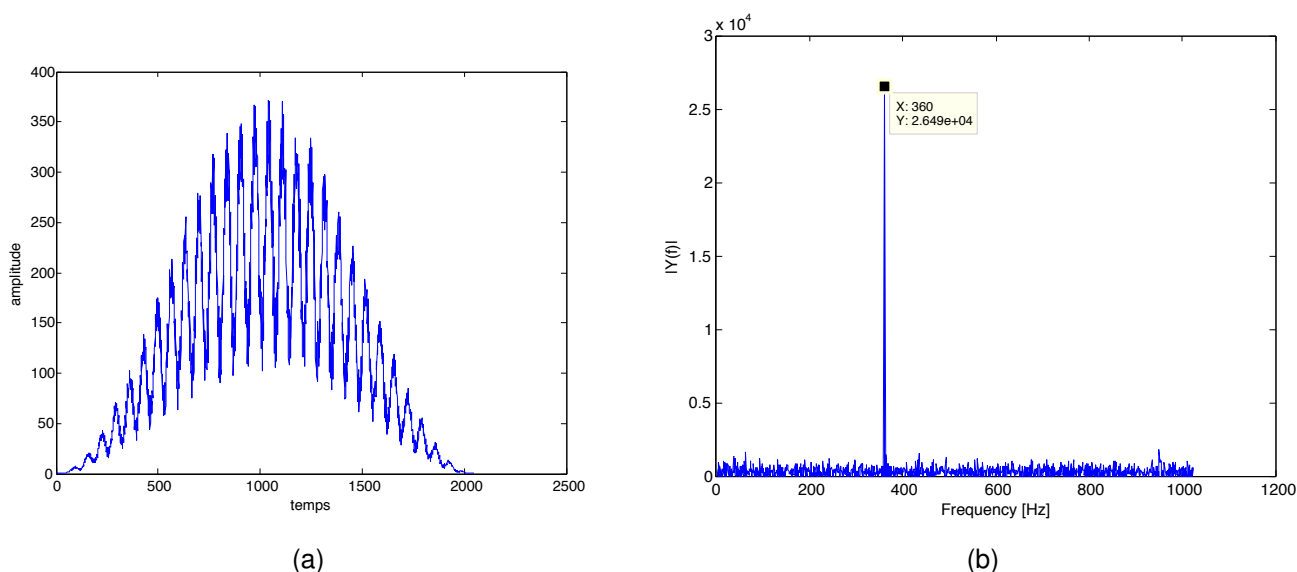


FIGURE 13.4 – (a) sinus à 360hz avec bruit superposé et fenêtre de Hann (b) résultat de la FFT

On constate que malgré le bruit, la FFT parvient toujours à récupérer le sinus. Ce bruit n'est cependant pas vraiment représentatif des bruits pouvant être rencontrés en pratique. L'idéal serait de pouvoir tester avec un bruit blanc ou même un bruit rose. Le bruit blanc a pour particularité de contenir "toutes" les fréquences du spectre.

## Sinus et bruit avancé

Dans ce test, une fonction a été utilisée pour générer le bruit. Cette fonction a été trouvée sur un site<sup>3</sup> mettant à disposition différentes sortes de bruit cohérent. Le code correspondant à la fonction est le suivant :

Listing 13.5 – Fonction IntegerNoise (générateur de bruit)

```

1 double IntegerNoise (int n)
2 {
3     n = (n >> 13) ^ n;
4     int nn = (n * (n * n * 60493 + 19990303) + 1376312589) & 0x7fffffff;
5     return 1.0 - ((double)nn / 1073741824.0);
6 }

```

L'amplitude du bruit et du sinus sont réglées à 100, la fréquence du sinus est de 360hz. Le code correspondant à la fonction est le suivant :

Listing 13.6 – Sinus 360hz bruit généré par la fonction IntegerNoise et fenêtre de Hann

```

1 inputTestFFT_f32[j] = (0.5*(1-cos((2*3.1415*j)/(LENGTH_FFT-1)))) *
2     (200.0
3     + 100.0* sin((2 * 3.1415 * j * f1) / LENGTH_FFT)
4     + ((float32_t)IntegerNoise(j)+1)*50.0);

```

La figure 13.5 représente la forme du signal à l'entrée et le résultat de la FFT.

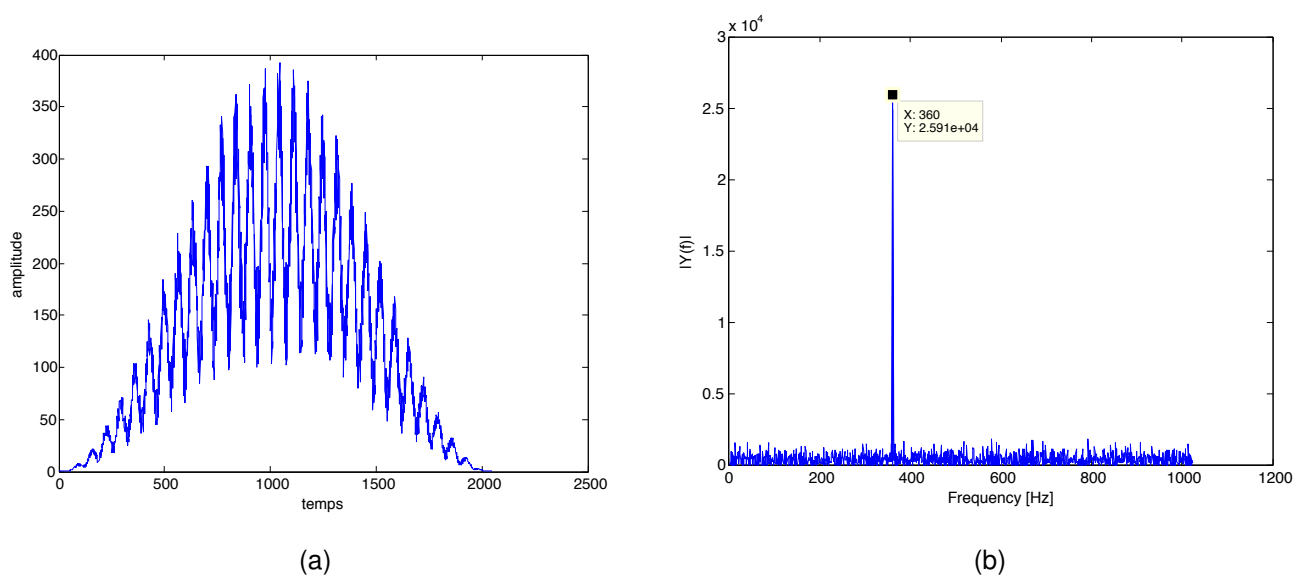


FIGURE 13.5 – (a) sinus à 360hz avec bruit superposé et fenêtre Hann (b) résultat de la FFT

Malgré le bruit, la FFT récupère facilement le sinus composant le signal. La fonction FFT du micro-contrôleur fonctionne donc correctement.

## 13.2 CONVERSION DES IMAGES EN NOIR ET BLANC ET SOMME DES PIXELS

### 13.2.1 Conversion RGB565 to grayscale

Les images reçues par la caméra sont au format RGB565, un pixel est codé sur 16 bits, 5 pour le rouge 6 pour le vert et 5 pour le bleu. Le code implémentant la conversion et la somme des pixels est disponible au listing 13.7.

3. <http://libnoise.sourceforge.net/noisegen/>, consulté le 30.06.14

Listing 13.7 – Conversion des pixel (en gris) et somme

```

1  uint16_t *img = (uint16_t *) msg.data;
2
3  for(j=0 ; j<msg.jpeg_size/2; j++)
4  {
5      valueOfPixel = (((*img & 0xf800) >> 11) << 3) +
6                      (((*img & 0x07e0) >> 5) << 2) +
7                      ((*img & 0x001f) << 3);
8
9      img++;
10     sumPixel += valueOfPixel / 3;
11 }

```

Dans cet extrait, "*\*img*" est un pointeur sur l'image. La boucle parcourt la totalité de l'image et récupère la valeur de rouge de vert et de bleu de chaque pixel. Puis elle somme ces valeurs et les divise par 3 pour obtenir la valeur de gris correspondante. Voir le chapitre 3.1.2 pour plus de détails concernant la somme des pixels de l'image.

### 13.2.2 Test

Une série de 200 images a été prise, convertie une fois par la fonction "*rgb2gray*" et une fois par le micro-contrôleur. La figure 13.6 présente les deux courbes obtenues.

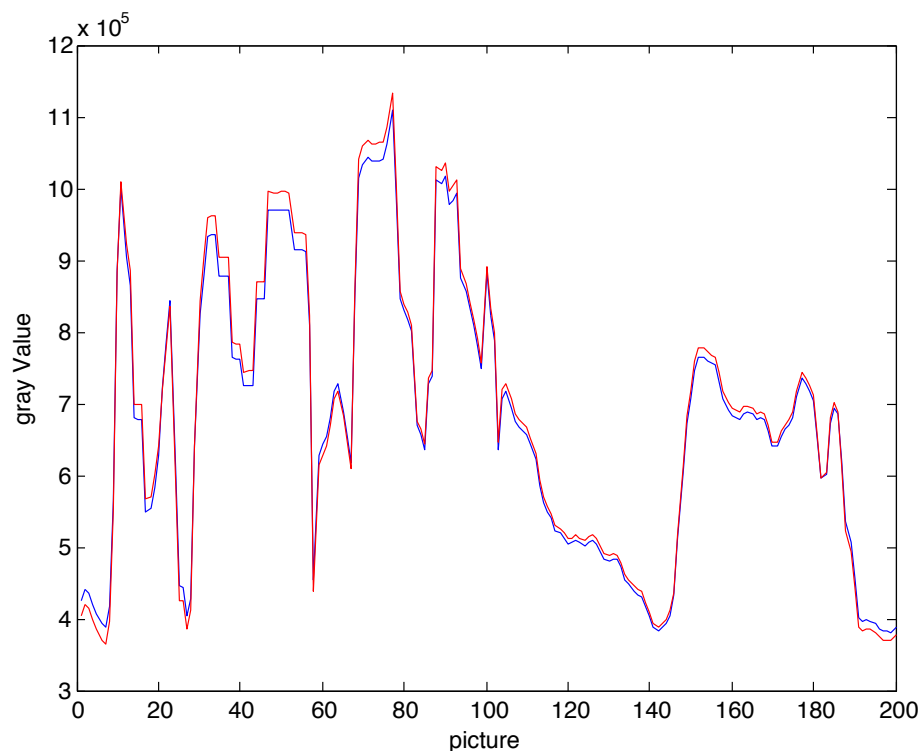


FIGURE 13.6 – en rouge la fonction "*rgb2gray*" de Matlab et en bleu la transformation grâce au micro-contrôleur

On constate une légère différence entre les deux types de conversions, mais dans l'ensemble la conversion implémentée dans le micro-contrôleur correspond assez bien à celle faite par la fonction "*rgb2gray*".



## Chapitre 14

# Implémentation de l'algorithme avec fenêtre glissante

### 14.1 DIAGRAMME DE FLUX

La figure 14.1 présente le diagramme de flux du traitements d'images effectué sur le micro-contrôleur dans son ensemble. Il complète le diagramme de la figure 9.3 du chapitre 9.2.1.

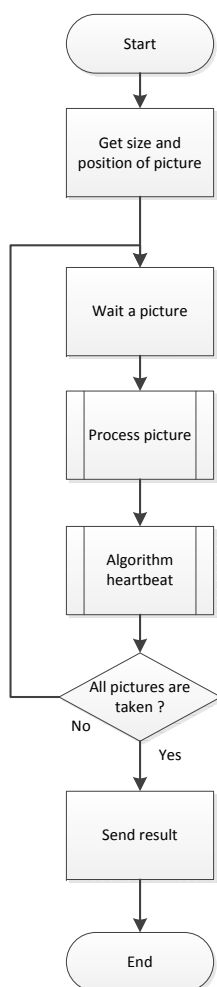


FIGURE 14.1 – Diagramme de flux

Sur ce diagramme, l'état "start" correspond à la réception d'un message USB indiquant la position du cœur ('p')(voir chapitre 9.2.3). Les fonctions "Process picture" et "Algorithm heartbeat" sont détaillées dans les paragraphes ci-dessous.

### 14.1.1 Process picture

La figure 14.2 présente le diagramme de flux de la fonction "Process picture".

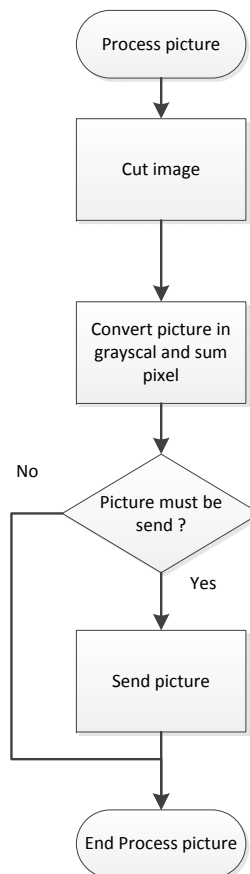


FIGURE 14.2 – Diagramme de flux "Process picture"

L'implémentation des parties "Cut image" et "Send picture" sont détaillées au chapitre 9 (Modification : prise d'images en continu). La partie "Convert picture in grayscale and sum pixel" est quant à elle expliquée au chapitre 13.2.

### 14.1.2 Algorithm heartbeat

La figure 14.3 présente le diagramme de flux de la fonction "Algorithm heartbeat".

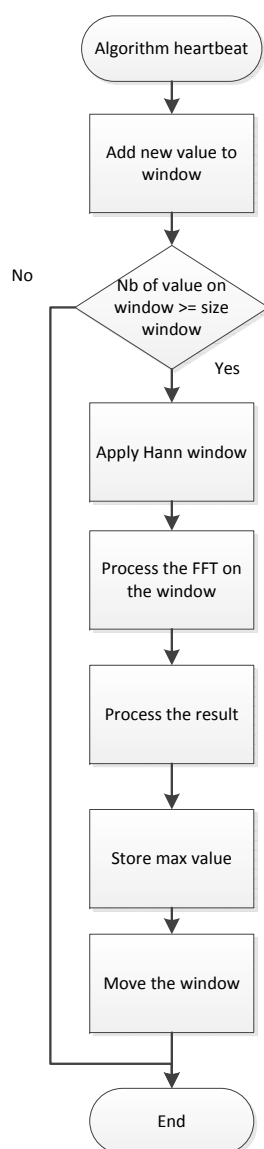


FIGURE 14.3 – Diagramme de flux "Algorithm heartbeat"

Ce dernier diagramme de flux présente l'algorithme qui récupérera la fréquence des battements de cœur. L'implémentation de la FFT ainsi que de la fenêtre de "Hann" sont détaillées au chapitre 13.1.

L'algorithme comporte également une fenêtre glissante. Son fonctionnement ainsi que son implémentation seront détaillés dans les paragraphes suivants.

## 14.2 FENÊTRE GLISSANTE

### 14.2.1 Fonctionnement

La fenêtre glissante est réalisée à l'aide d'un tableau et d'un pointeur. Le tableau se remplit au fur et à mesure que les valeurs mesurées arrivent. La figure 14.4 présente le tableau des valeurs ainsi que la fenêtre.

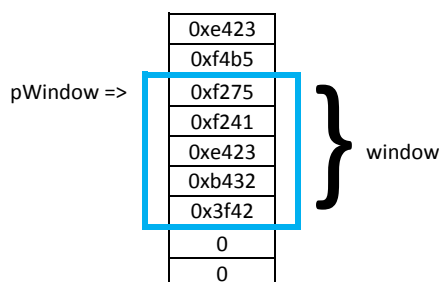


FIGURE 14.4 – Fenêtre glissante

Le pointeur "pWindow" pointe toujours sur la fenêtre, cette dernière est représentée en bleu sur la figure. Dans ce cas, la taille de la fenêtre est de 6 et le nombre maximum de mesures est de 10.

Tant que la fenêtre n'est pas totalement chargée, le pointeur reste sur la première position du tableau.

Cette méthode a l'avantage d'être rapide à l'exécution. En effet, lorsqu'une nouvelle valeur apparaît, elle est stockée dans le tableau et le pointeur est incrémenté. Ces deux opérations sont effectuées rapidement et ne demandent que peu de ressources au processeur.

L'inconvénient est que la totalité des mesures sont stockées en mémoire. Le nombre maximum de mesures est fixé à 1000 images par cycle. Un tableau d'une taille de 1000 "float32\_t" est donc nécessaire.

### Alternative

Pour économiser de la mémoire, on pourrait imaginer stocker uniquement les valeurs de la fenêtre. A chaque nouvelle valeur, tout le tableau serait décalé pour laisser une place à la nouvelle valeur. Le grand inconvénient de cette méthode est qu'à chaque nouvelle valeur, la totalité du tableau est réécrit. Le nombre d'opérations effectuées dépendrait donc de la taille de la fenêtre.

Une autre solution serait de recopier le tableau décalé depuis l'ancien grâce à un "memcpy".

Ces deux alternatives sont envisageables mais elles nécessitent une implémentation plus complexe et pas forcément plus performante. Pour ces raisons, la solution avec le tableau et le pointeur décrite au début du chapitre est adoptée.

## 14.2.2 Implémentation

L'implémentation de la fenêtre glissante est détaillée ci-dessous. Elle concerne les bloc "Add new value to window" et "Move the window".

Le tableau ("sumValuePixel") contenant les valeurs des mesures est de type "float32\_t". Sa taille est définie par le "#define" "MAX\_SIZE\_MEASUREMENT".

Le déplacement de la fenêtre "Move the window" se fait simplement en incrémentant le pointeur "pWindow".

## 14.3 FENÊTRE DE "HANN"

La description de la fenêtre de "Hann" ainsi que le but de son utilisation sont décrits au chapitre 13.1.3.

Les coefficients de la fenêtre de "Hann" sont pré-calculés et stockés dans un tableau à l'initialisation. Le listing 14.1 présente le code de la fonction réalisant ces options.

Listing 14.1 – Calcule des coefficients de "Hann"

```

1 static void compute_hann_window()
2 {
3     uint32_t j = 0;
4     for (j=0 ; j<SIZE_SLIDING_WINDOW; j++)
    
```

```

5   {
6       hann_window[j] = 0.5*(1-cos((2*3.1415*j)/(SIZE_SLIDING_WINDOW-1)));
7   }
8 }

```

## Application

La fenêtre de "Hann" est appliquée à la fenêtre de mesure simplement à l'aide d'une multiplication réalisée dans une boucle. Cette opération est représentée sur le diagramme de flux de la figure 14.3 sous le nom "Apply Hann window". Le listing 14.2 présente le code réalisant cette opération.

Listing 14.2 – Fenêtre de Hann appliquée au signal

```

1  pointer = pWindow;
2
3  // apply hann window
4  for(j=0; j<SIZE_SLIDING_WINDOW ;j++)
5  {
6      window[j] = *pointer * hann_window[j];
7      pointer ++;
8  }

```

## 14.4 TRAITEMENT DU RÉSULTAT DE LA FFT

### 14.4.1 Interprétation des résultats de la FFT

Le résultat obtenu en sortie de la FFT est un tableau contenant des nombres réels et imaginaires (voir chapitre 13.1.4). Afin de sortir de la FFT la valeur intéressante, il est nécessaire de calculer le module des nombres complexes ( $\sqrt{Re^2 + Im^2}$ ). La racine n'a pas besoin d'être utilisée dans ce cas, on souhaite connaître la position de la valeur maximum et non la valeur en elle même.

Pour obtenir la fréquence correspondant à la l'index du tableau des modules, il faut multiplier ce dernier par  $\frac{Fs}{N}$  avec  $Fs$  = fréquence échantillonnage et  $N$  = longueur de la FFT. La dernière valeur du tableau correspond donc à  $Fs$ . Les valeurs entre  $\frac{Fs}{2}$  et  $Fs$  sont au dessus de la moitié de la fréquence d'échantillonnage et ne présentent donc aucun intérêt.

Le listing 14.3 présente le code calculant le module au carré des nombres complexes ( $Re^2 + Im^2$ ). Seul les coefficients au dessus de  $\frac{Fs}{2}$  (moitié) sont calculés.

Listing 14.3 – Calcul du module de la FFT

```

1  //compute module of complex number
2
3  for (j = 0; j < LENGTH_FFT_WINDOW / 4; j++)
4  {
5      result_fft[j] = outputFFT_f32[2*j] * outputFFT_f32[2*j] + outputFFT_f32[2*j+1] * outputFFT_f32[2*j+1];
6  }

```

### 14.4.2 Suppression des fréquences sans intérêt

La suite des opérations concerne la suppression des fréquences sans intérêt (voir chapitre 12.1 pour plus d'explications). Le code réalisant cette opération est présenté au listing 14.5.

Listing 14.4 – Suppression des fréquences sans intérêt

```

1  for(j=0; j<LENGTH_FFT_WINDOW/2;j++)
2  {
3      if(j < (uint32_t) (fcLow * (LENGTH_FFT_WINDOW/fs)))
4      {
5          result_fft[j]=0;

```

```

6     }
7     if(j > (uint32_t) (fcHigh*(LENGTH_FFT_WINDOW/fs)))
8     {
9         result_fft[j]=0;
10    }
11 }
  
```

fcLow correspond à la fréquence de coupure basse, fcHigh à la fréquence de coupure haute et fs à la fréquence d'échantillonnage.

## 14.5 STOCKAGE DE LA VALEUR MAXIMUM

Cette opération ("Store max value" dans le diagramme de flux de la figure 14.3) a pour objectif de trouver la fréquence correspondant à la valeur maximum de la FFT. Une fois cette fréquence trouvée, elle est stockée dans un tableau. Le listing 14.5 présente le code correspondant.

Listing 14.5 – Récupération de la fréquence dominante

```

1 float32_t maxValue = 0;
2 float32_t indexMaxValue = 0;
3
4 //find max value
5 for(j=0; j<LENGTH_FFT_WINDOW/2;j++)
6 {
7     if(result_fft[j]>maxValue)
8     {
9         maxValue = result_fft[j];
10        indexMaxValue = j;
11    }
12 }
13
14 // convert index of max value in frequency
15 indexMaxValue = indexMaxValue * (fs/LENGTH_FFT_WINDOW);
  
```

## 14.6 TEST

### 14.6.1 Test du traitement des résultats de la FFT

Différents tests ont permis de valider le bon fonctionnement du traitement des résultats de la FFT par le micro-contrôleur. La suppression des fréquences indésirables ainsi que la récupération de la valeurs maximum fonctionnent correctement.

Les signaux utilisés pour ces tests sont semblables à ceux utilisés au chapitre 13.1.5. Les fréquence obtenues à la sortie du micro-contrôleur correspondent à celles du sinus d'amplitude maximum.

### 14.6.2 Test de l'algorithme dans son ensemble

Pour des raisons de manque de temps, l'algorithme n'a pas pu être testé dans son ensemble. Le fonctionnement de la fonction "algorithm\_heartbeat" n'est pas garanti pour le moment.

Pour tester cette fonction, il va falloir créer un signal ressemblant à celui de la moyenne des pixels de chaque image. Ce signal comportera une composante DC (offset), un sinus correspondant à la fréquence cardiaque de la daphnie et quelques autres éléments parasites. Puis une analyse minutieuse des résultats permettra de valider le fonctionnement, en vu d'un test sur les daphnies.

Ensuite, le test sur les daphnies pourra être réalisé. En parallèle de l'algorithme, il faudra transmettre les images au PC. Puis exécuter l'algorithme Matlab et comparer les résultats obtenus dans les deux cas.

Il faudra probablement jouer sur la longueur de la FFT. Elle peut être de 128, 512 ou de 2048. Avec 2048, la résolution est bien meilleure (0.01Hz/pts), mais le temps de traitement est plus important. Si ce temps s'avère

plus long que celui de la période de capture d'images, il faudra probablement adopter une taille de FFT de 512 et donc obtenir une résolution de 0.04Hz/pts.

## Chapitre 15

# Déroulement du travail

### 15.1 PLANNING

Le planning établi est disponible au chapitre D.3.

Les cases vertes correspondent au travail prévu et les case avec un "x" au travail effectué. La première partie du tableau comporte les tâches prévues et la deuxième les imprévus.

#### 15.1.1 Modifications du planning et imprévus

Le planning n'a pas été suivi à la lettre, et ce pour différentes raisons.

##### Problème printf

Deux jours ont dû être consacrés à la mise en place de la fonction printf. Aucun message n'était reçu lorsque la fonction printf était appelée.

La carte a été développée pour pouvoir être cassée en deux et ainsi retirer la partie servant au debug et à la programmation. La figure 15.1 présente une partie du système embarqué, on y voit la coupure permettant de détacher la partie "debug"(gauche sur l'image).

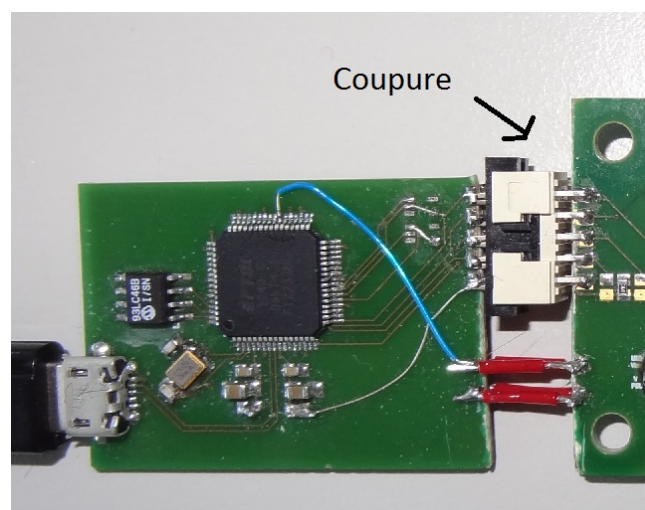


FIGURE 15.1 – Photo du système embarqué



Une fois la coupure effectuée, un connecteur (blanc et noir sur l'image) a été soudé pour pouvoir tout de même programmer le système. Le connecteur soudé ne concernait que la partie "JTAG" de la carte. Il a fallu rajouter deux ponts (fils rouge), un pour le RX et un pour TX de l'UART du printf.

Malgré cela, le printf ne fonctionnait toujours pas. Une analyse des schémas a alors permis de constater que la piste de l'UART passait dans une couche interne du circuit et que le pont rouge n'était d'aucune utilité. Il a donc fallu rajouter un pont directement depuis le "FTDI" (fil bleu sur l'image).

### Problème librairie USB

Un certain temps a également été perdu pour trouver la faute et corriger un bug de la librairie USB. Lorsqu'une donnée d'une taille supérieur à 65'536 bytes était envoyée, la donnée n'était pas reçue correctement. Le bug a été trouvé dans la librairie "stm32\_usbd\_cdc.c" aux lignes 160-161. La taille des données restantes à envoyer n'était pas correctement décrémentée. Le listing 15.1 présente le code avant modification et le listing 15.2 le code modifié.

Listing 15.1 – Code avant correction

```
1 data += max_size/4;  
2 len -= max_size;
```

Listing 15.2 – Code après correction

```
1 data += max_size;  
2 len -= max_size;
```

### Problème fréquence de prise d'images insuffisante

Les images étaient prises à environ 10 img/sec, ce qui était totalement insuffisant pour l'application. Un certain nombre de modifications ont été nécessaires pour augmenter cette fréquence à 20 img/sec. Le chapitre 9.4 détaille ces modifications.

### Suppression de l'algorithme C sur ordinateur

Pour des raisons de temps, la version C de l'algorithme devant tourner sur l'ordinateur, ainsi que les tests s'y rapportant ont été mis de côté.

### Test du fonctionnement de la librairie FFT de ARM

Le temps pour tester et configurer la librairie FFT sur le ARM n'a pas été prévu dans le planning initial. Il s'avère que ces tests ont été nécessaires et longs.

### Amélioration de l'algorithme Matlab

Le temps nécessaire à l'adaptation et à l'amélioration de l'algorithme Matlab a été oublié dans le planning initial. Ces modifications concernent principalement l'algorithme de la fenêtre glissante.

### 15.1.2 Tâches non réalisées

#### Carte SD

Une des tâches consistait à sauvegarder les images sur la carte SD.

Dans la version du système embarqué, le support de la carte SD a été routé à l'envers et n'a donc pas été soudé. Il est donc nécessaire de souder le support avec des ponts pour pouvoir inverser les contacts.

Une librairie développée par la HES-SO de Sion devrait permettre de réaliser la sauvegarde d'images relativement facilement. Cette tâche n'a pas été réalisée par manque de temps.

#### Test de l'algorithme "heartbeat"

Pour des raisons de manque de temps, le test de l'algorithme "heartbeat" n'a pas pu être terminé. Son fonctionnement n'est donc pas garanti. Plus de détails sont disponibles au chapitre 14.6.2.

## Chapitre 16

# Améliorations

### 16.1 AMÉLIORATION DE L’AFFICHAGE DES BATTEMENTS CARDIAQUES

Pour le moment, l’affichage des battements cardiaques est très sommaire. La valeur est envoyée sur le port USB de debug grâce à un printf. Il serait très intéressant de rajouter un onglet dans l’application Qt pouvant stocker et afficher ces résultats. On obtiendrait alors l’évolution de la fréquence cardiaque de la daphnie de façon simplifiée, directement lisible.

### 16.2 AMÉLIORATION DE LA SÉLECTION DE LA POSITION DU CŒUR

Actuellement, le cœur doit être sélectionné manuellement par un opérateur sur l’ordinateur. Il serait intéressant de pouvoir faire une sélection de la zone à partir d’un smartphone Android.

Pour que cette amélioration soit envisageable, il faudrait rajouter un module Bluetooth au système embarqué. Une image compressée serait alors envoyée sur le smartphone à l’utilisateur qui sélectionnerait la zone contenant le cœur simplement en touchant l’image. Les positions seraient ensuite transmises au micro-contrôleur. Les battements cardiaques pourraient être affichés sur le smartphone pratiquement en temps réel.

## Chapitre 17

# Conclusion

Le but de ce projet était de réaliser un système embarqué permettant de filmer de manière autonome une daphnie et de détecter son rythme cardiaque.

Pour des raisons de limitation de la bande passante USB, le système embarqué peut filmer uniquement une partie de la daphnie. La taille maximum est fixée à 150x150 pixels. Le programme d'acquisition d'images développé permet de sélectionner la zone contenant le cœur de la daphnie.

L'algorithme développé sur Matlab a par la suite permis de détecter la fréquence cardiaque à partir des images de la daphnie prises par le système embarqué. Malgré la mauvaise qualité de la caméra utilisée (résolution relativement faible,... ) les images obtenues ont permis de détecter la fréquence cardiaque. L'éclairage joue aussi un rôle très important lors de la prise d'images de ce type. Il est nécessaire de l'adapter afin d'obtenir un contraste permettant de voir correctement le cœur de la daphnie.

Le piège servant à l'immobilisation de la daphnie est également un élément critique. Dans certaines prises d'images, le sujet avait tendance à ne pas rester en place, rendant la détection de la fréquence cardiaque difficile, voir impossible. De plus, la daphnie ne se positionne pas toujours au même endroit dans le piège. Il est nécessaire de déplacer la caméra pour la positionner en face de la daphnie.

Le portage de l'algorithme Matlab sur le système embarqué a pu être en parti réalisé. Les tests le concernant seront réalisés durant les semaines suivant la remise de ce rapport.

Sion, le 11 juillet 2014

Jérémie MAYOR

## Chapitre 18

# Bibliographie

- Page : 3      Article Daphnie Wikipédia  
<http://fr.wikipedia.org/wiki/Daphnie>, consulté le 01.05.14
- Page : 8      Matlab FFT Help  
<http://www.mathworks.ch/ch/help/matlab/ref/fft.html>, consulté le 06.03.14
- Page : 8      FFT Tutorial  
<http://www.ele.uri.edu/hansenj/projects/ele436/fft.pdf>, consulté le 06.03.14
- Page : 66      Documentation CMSIS - ARM  
<http://www.arm.com/products/processors/cortex-m/cortex-microcontroller-software-interface-standard.php>, consulté le 30.06.14
- Page : 77      libnoise : Generating coherent noise  
<http://libnoise.sourceforge.net/noisegen/>, consulté le 30.06.14

## Annexe A

# Annexe : Vidéos

### A.1 CHOIX DES VIDÉOS POUR LES TESTS

Vidéo	durée [s]	[img/s]	utilisée	début uti.	fin uti.	commentaires
A1_M1	44	30	non	-	-	La daphnie bouge énormément ces pattes et elle tourne sur elle-même
B1_M2	44	30	oui	0 s	4.5s	Au-delà de 5 seconde la daphnie bouge également
B2_M2	45	30	oui	0 s	15 s	A 15 seconde la caméra change de plan
C1_M1	46	30	non	-	-	La daphnie bouge énormément ces pattes et elle tourne sur elle-même
D1_M2	46	30	oui	0 s	15 s	A 15 seconde la caméra change de plan
Loading_t0	15	75	oui	0 s	11 s	Mouvements de pattes de la daphnie à partir de 11 secondes
Loading_t10	15	75	oui	0 s	4 s	Mouvements de pattes de la daphnie à partir de 4 secondes

FIGURE A.1 – choix des vidéos pour les tests

## A.2 MESURE DES FRÉQUENCES CARDIAQUES PAR COMPTAGE

**Vidéo : Loading\_t0 :**

F. Sampl [img/s]	First pict.	Last pict.	heartbeat	# of pictures	length [s]	heatBeat [Hz]
75	1	850	64	850	11.33	5.65
		heartbeat - 1	63			5.56
		heartbeat + 1	65			5.74

**Vidéo : B2\_M2 (Ehippia) :**

F. Sampl [img/s]	First pict.	Last pict.	heartbeat	# of pictures	length [s]	heatBeat [Hz]
30	1	463	75	463	15.43	4.86
		heartbeat - 1	74			4.79
		heartbeat + 1	76			4.92

**Vidéo B1\_M2 :**

F. Sampl [img/s]	First pict.	Last pict.	heartbeat	# of pictures	length [s]	heatBeat [Hz]
30	1	128	24	128	4.27	5.63
		heartbeat - 1	23			5.39
		heartbeat + 1	25			5.86

**Vidéo Loading\_t1 :**

F. Sampl [img/s]	First pict.	Last pict.	heartbeat	# of pictures	length [s]	heatBeat [Hz]
75	1	300	27	300	4.00	6.75
		heartbeat - 1	26			6.50
		heartbeat + 1	28			7.00

**D1\_M2 :**

F. Sampl [img/s]	First pict.	Last pict.	heartbeat	# of pictures	length [s]	heatBeat [Hz]
30	1	463	40	463	15.43	2.59
		heartbeat - 1	39			2.53
		heartbeat + 1	41			2.66

FIGURE A.2 – choix des vidéos pour les tests

L'erreur de comptage est estimée à plus ou moins une image.

## Annexe B

# Annexe : Résultats complets des tests

### B.1 RETARD DU FILTRE

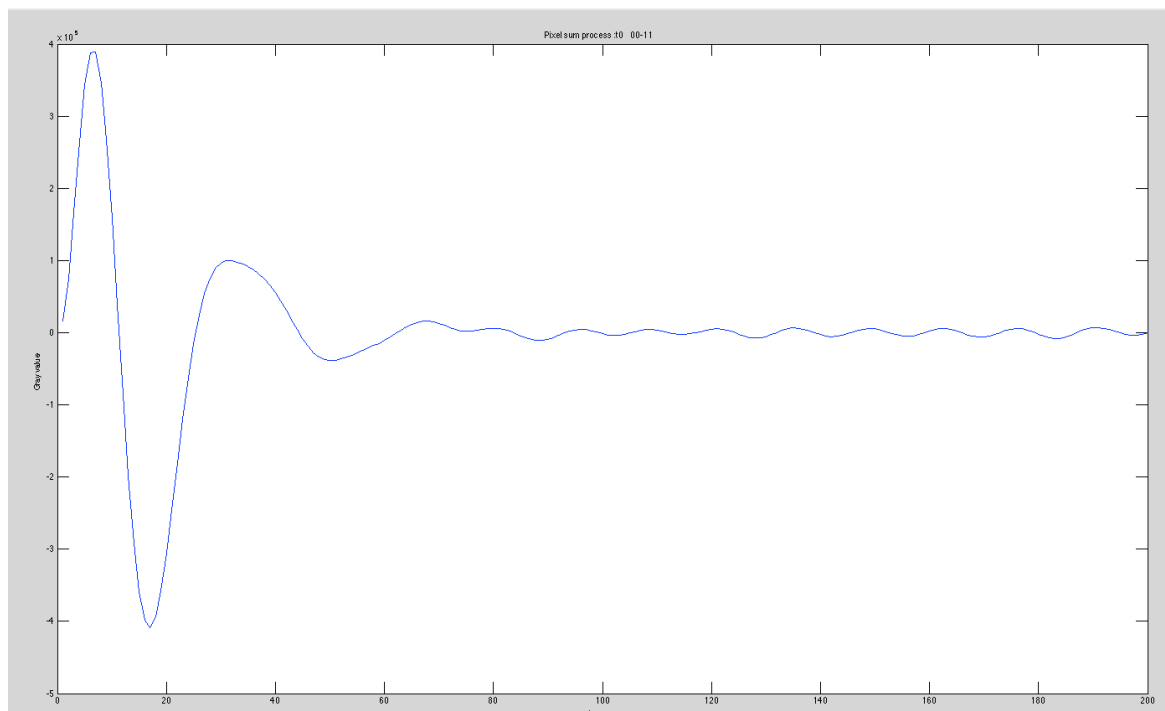


FIGURE B.1 – Retard d'environ 100 images dû au filtre Butterworth d'ordre 2



## B.2 RÉSULTATS : ALGORITHME DE BASE

Vidéo : B2\_M2(Ephippia)

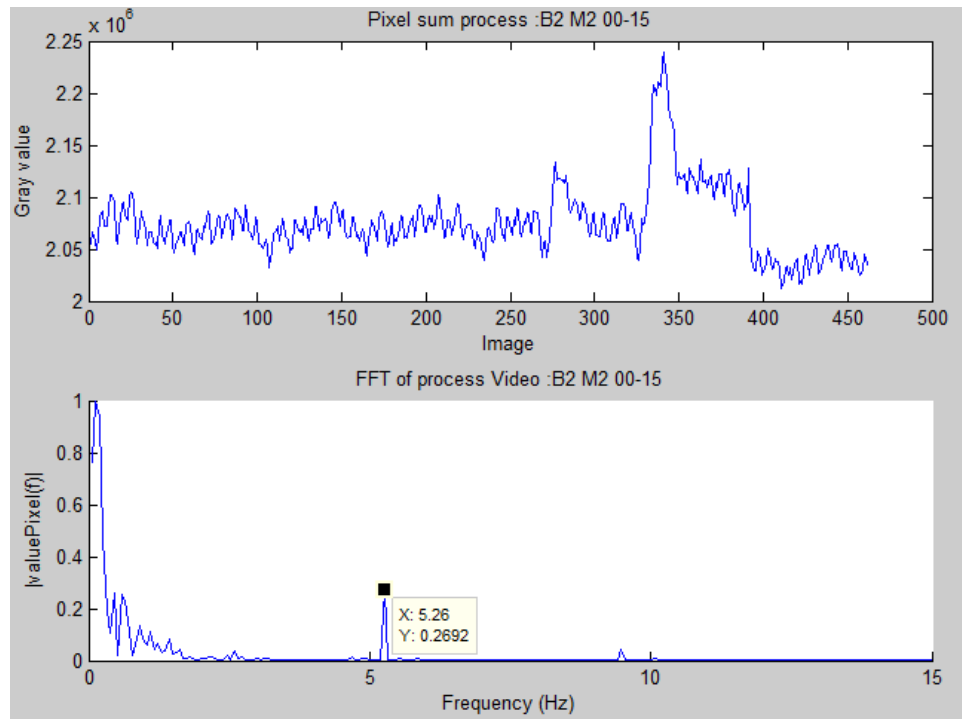


FIGURE B.2 – Résultat vidéo : B2\_M2(Ephippia)

Vidéo : Loading\_t0

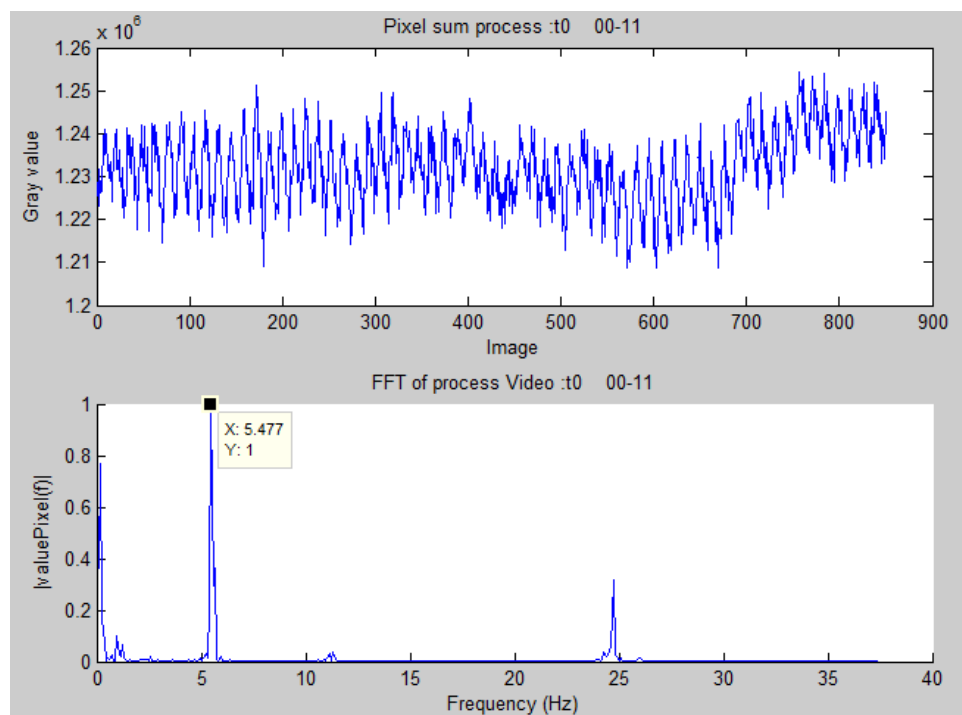


FIGURE B.3 – Résultat vidéo : Loading\_t0

**Vidéo : D1\_M2**

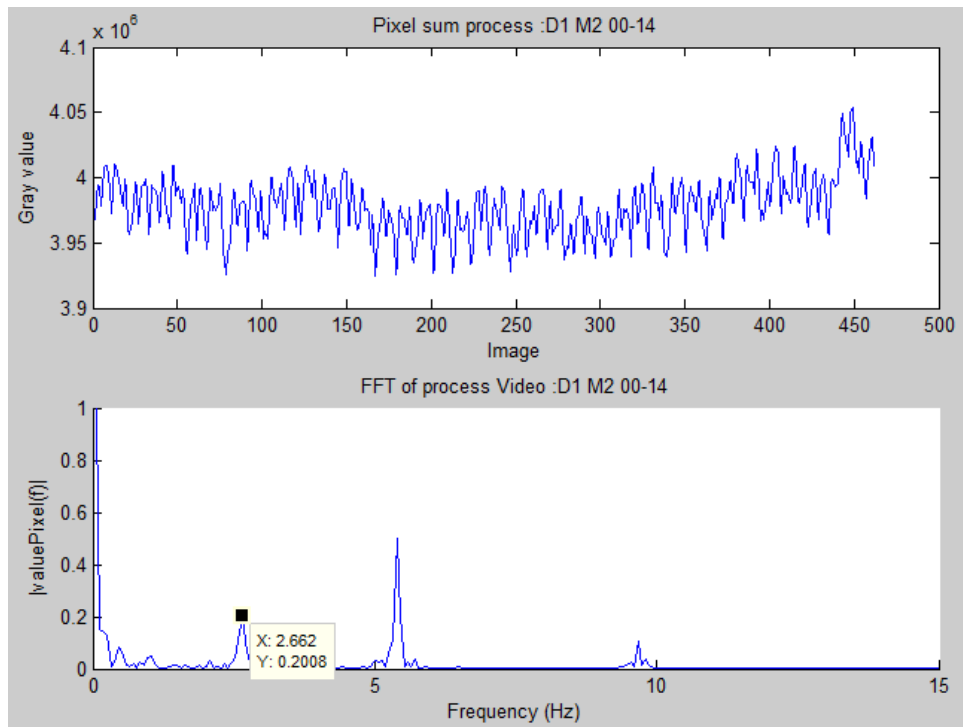


FIGURE B.4 – Résultat vidéo : D1\_M2

**Vidéo : B1\_M2**

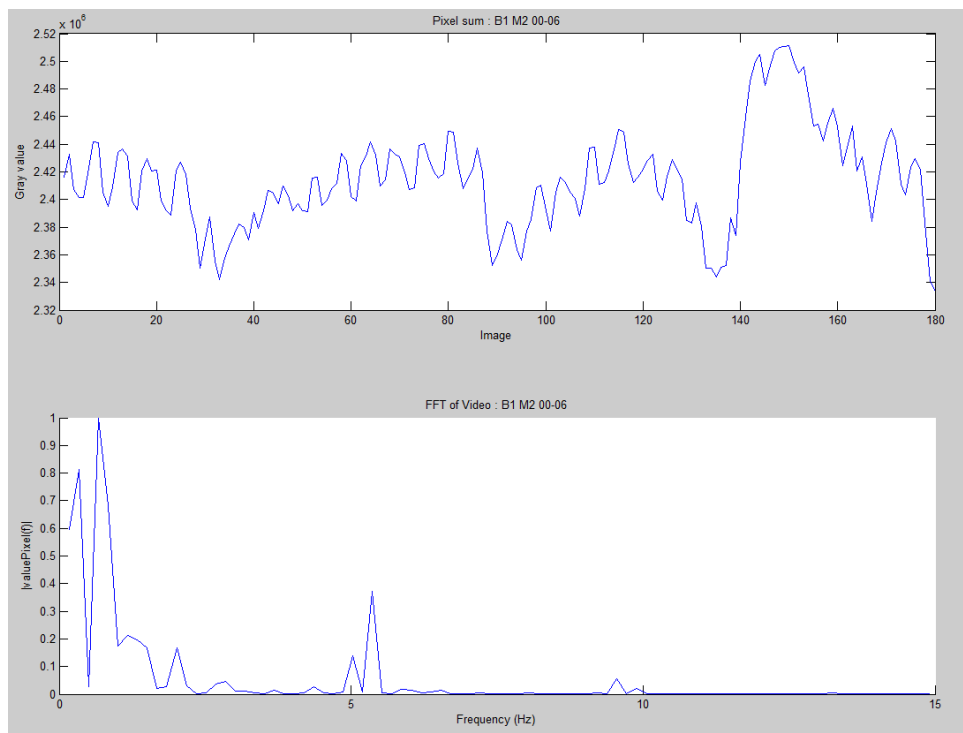


FIGURE B.5 – Résultat vidéo : B1\_M2

**Vidéo : Loading\_t10**

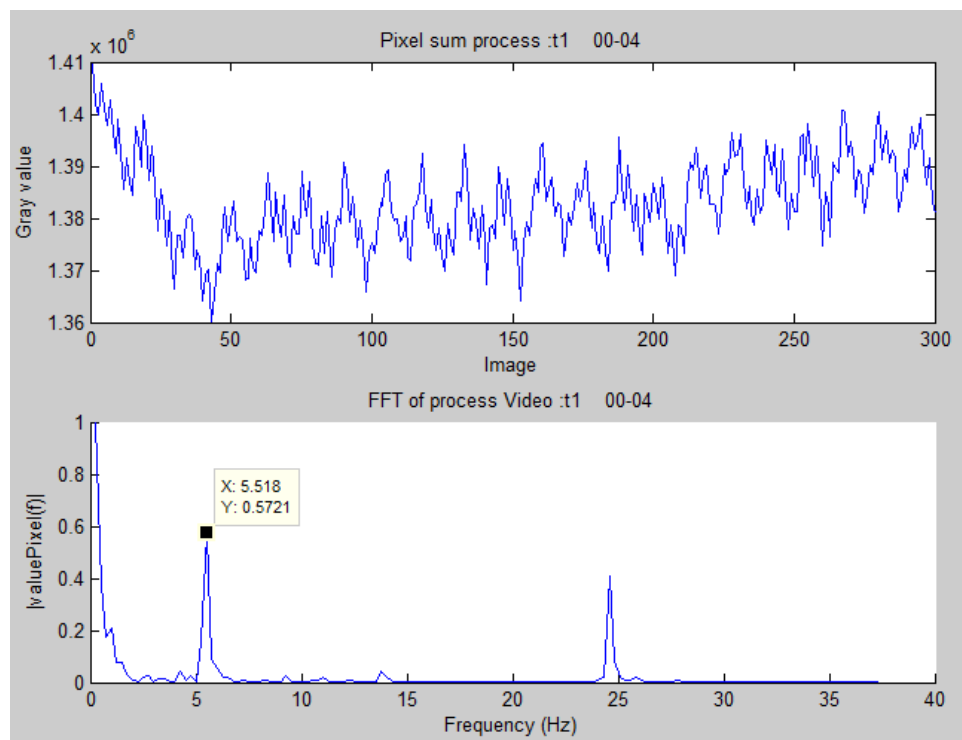


FIGURE B.6 – Résultat vidéo : Loading\_t10

## B.3 RÉSULTATS : ALGORITHME AVEC FILTRE

Vidéo : B2\_M2(Ephippia)

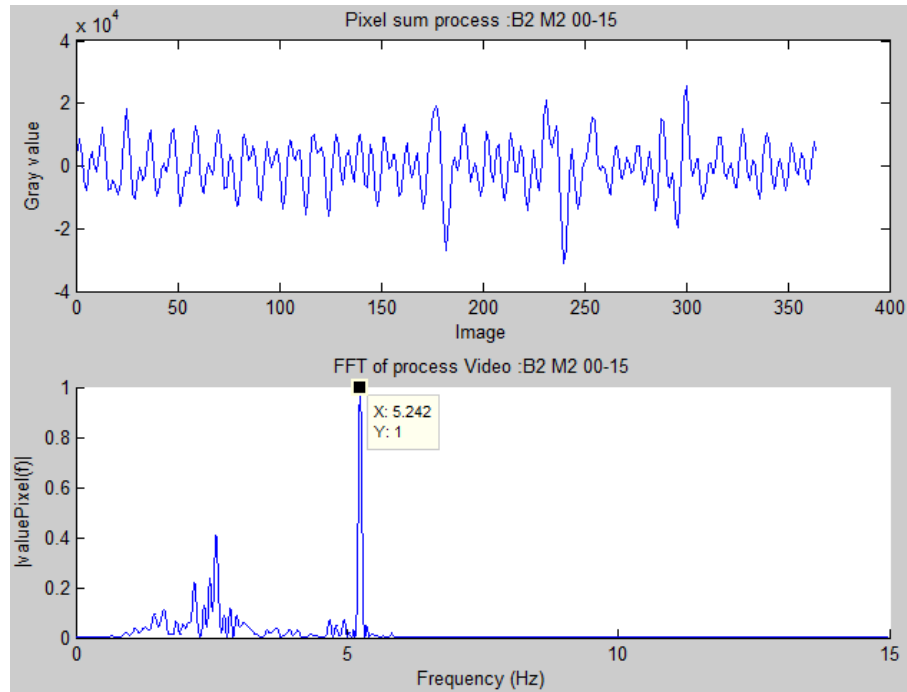


FIGURE B.7 – Résultat vidéo : B2\_M2(Ephippia)

Vidéo : Loading\_t0

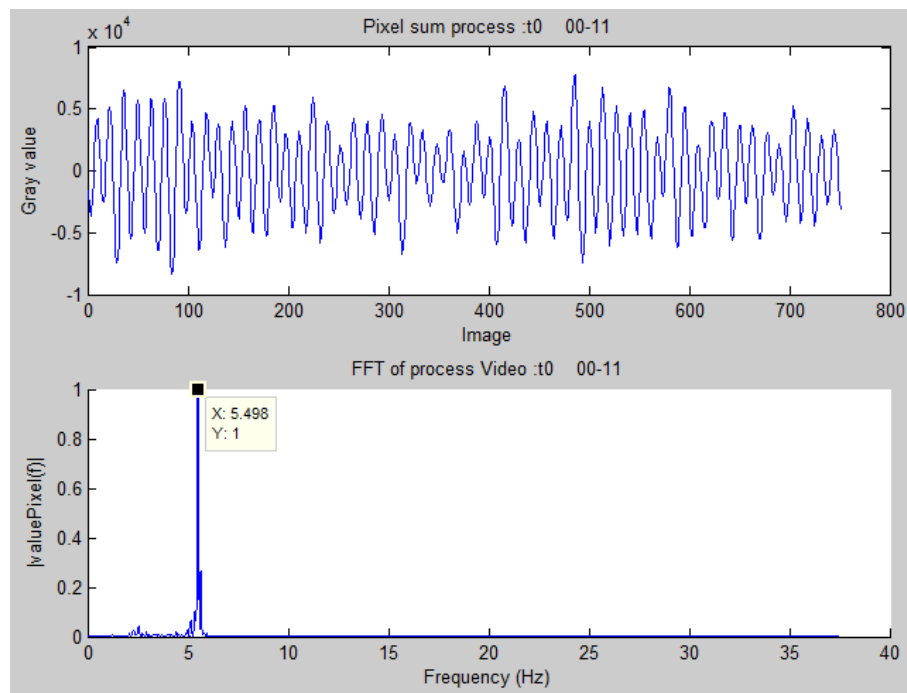


FIGURE B.8 – Résultat vidéo : Loading\_t0

## Vidéo : D1\_M2

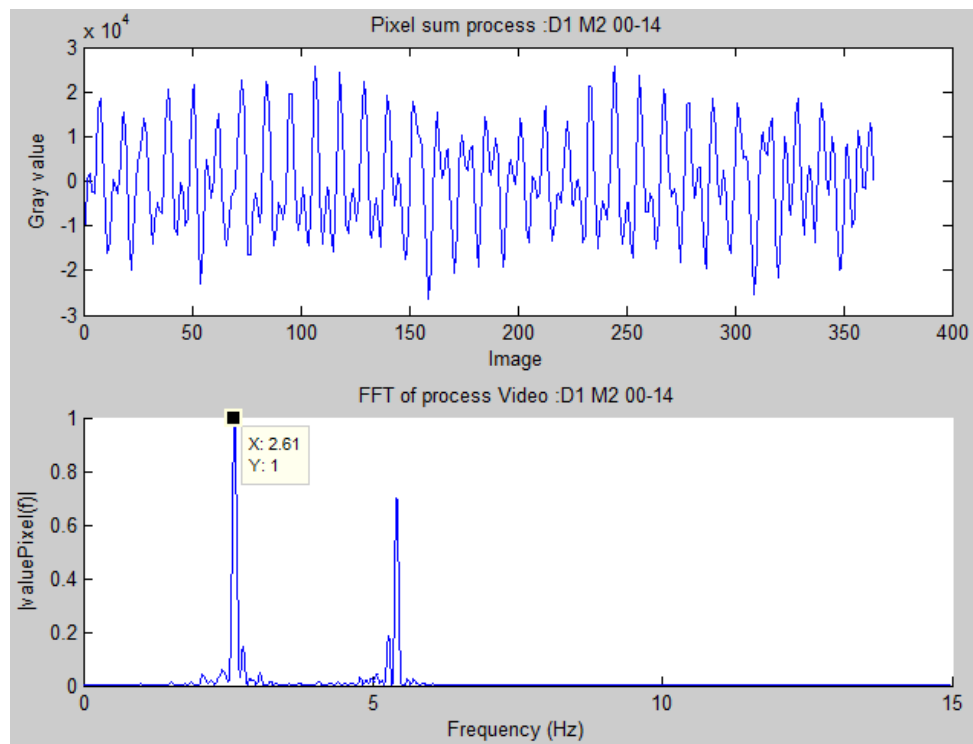


FIGURE B.9 – Résultat vidéo : D1\_M2

## Vidéo : B1\_M2

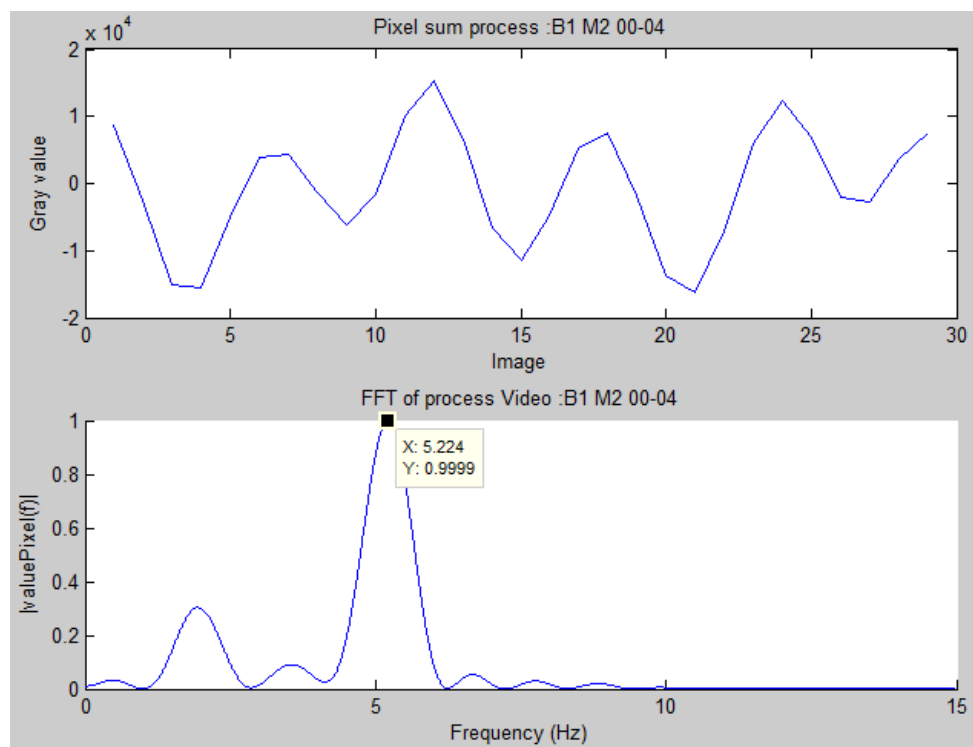


FIGURE B.10 – Résultat vidéo : B1\_M2

## Vidéo : Loading\_t10

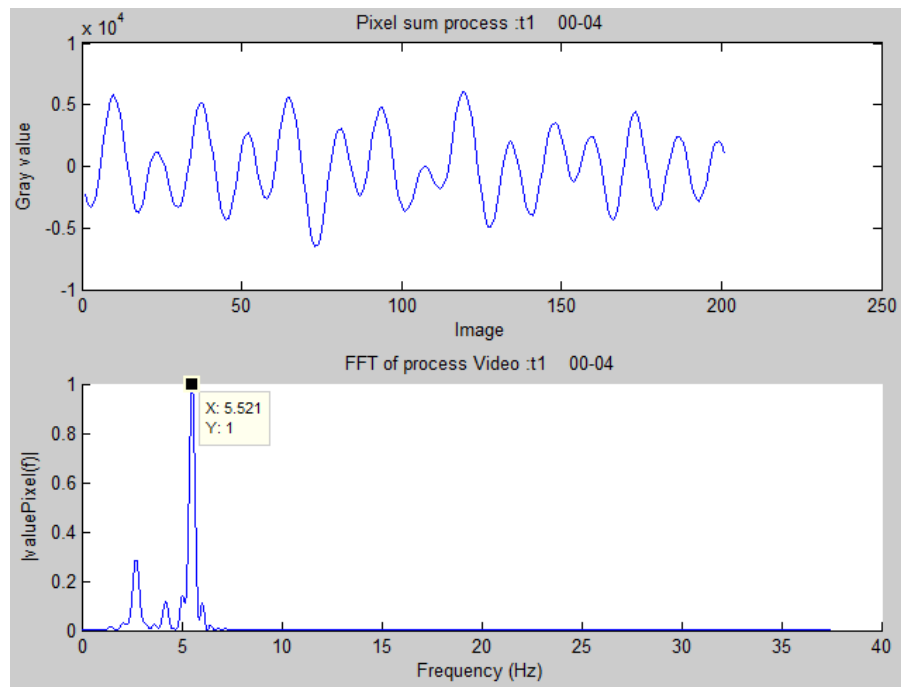


FIGURE B.11 – Résultat vidéo : Loading\_t10

## B.4 RÉSULTATS : ALGORITHME AVEC FENÊTRE GLISSANTE

### Vidéo : B2\_M2(Ephippia)

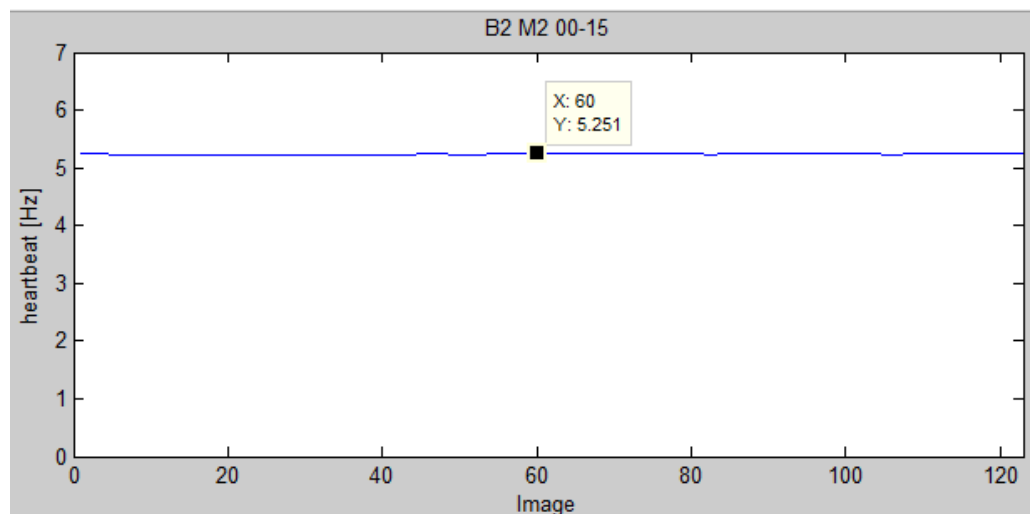


FIGURE B.12 – Résultat de la sliding window de la vidéo :B2\_M2(Ephippia) avec taille de la fenêtre = 170 img

**Vidéo : Loading\_t0**

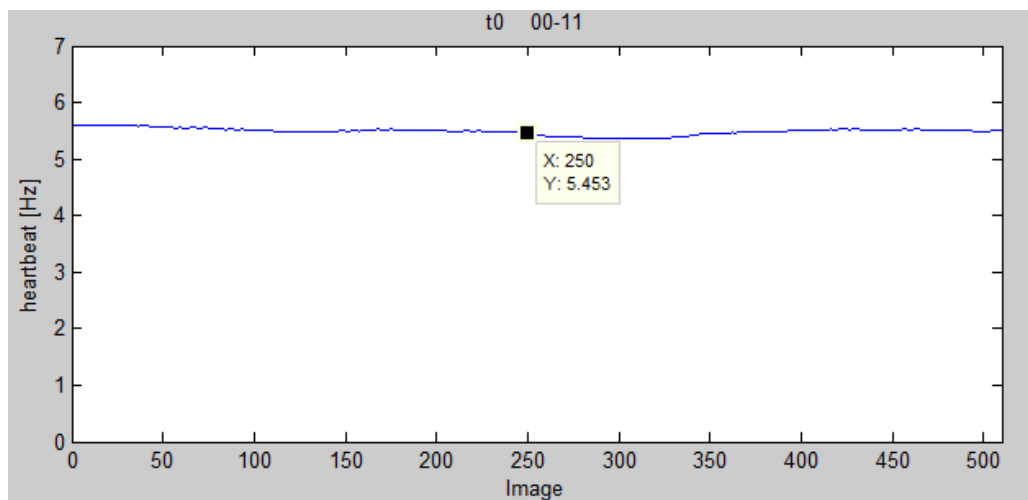


FIGURE B.13 – Résultat de la sliding window de la vidéo :Loading\_t02 avec taille de la fenêtre = 170 img

**Vidéo : D1\_M2**

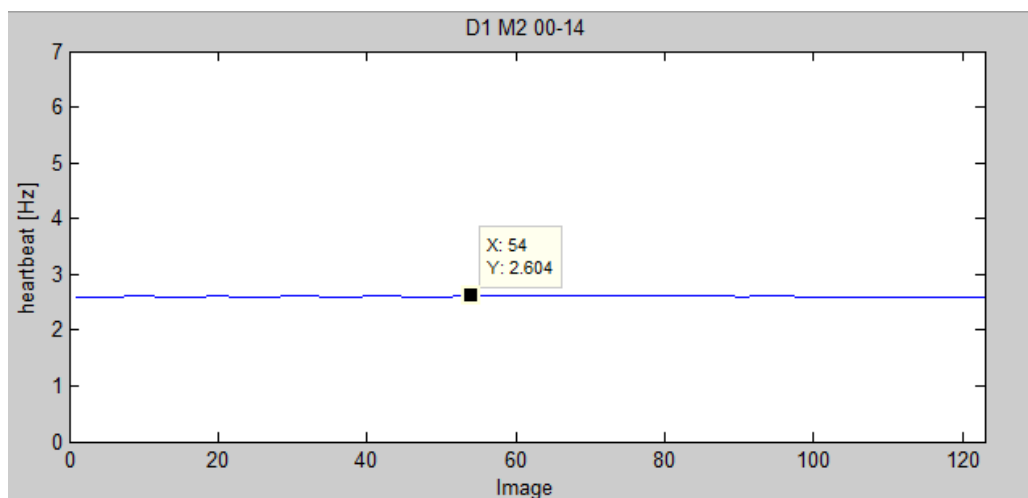


FIGURE B.14 – Résultat de la sliding window de la vidéo : D1\_M2 avec taille de la fenêtre = 170 img

## Annexe C

# Annexe : Comptage des battements cardiaque

### C.1 BATTEMENTS CARDIAQUES DE LA DAPHNIE ÉCHANTILLON 1, 2 ET 3

Echantillon 1 : "test1-0hz220V"

F. Sampl [img/s]	First pict.	Last pict.	heatbeat	# of pictures	length [s]	heatBeat [Hz]
20	3731	4101	90	371	18.55	4.85
20	4102	4347	64	246	12.30	5.20
20	4348	4545	60	198	9.90	6.06
20	4546	4730	50	185	9.25	5.41
20	3731	4730	264	1000	50.00	5.28

Echantillon 2 : "test2-0hz220vdiphnoir"

F. Sampl [img/s]	First pict.	Last pict.	heatbeat	# of pictures	length [s]	heatBeat [Hz]
20	8890	9129	60	240	12.00	5.00
20	9130	9387	70	258	12.90	5.43
20	9388	9685	80	298	14.90	5.37
20	9686	9889	56	204	10.20	5.49
20	8890	9889	266	1000	50.00	5.32

Echantillon 3 : "test1-2hz220v"

F. Sampl [img/s]	First pict.	Last pict.	heatbeat	# of pictures	length [s]	heatBeat [Hz]
20	2559	2821	70	263	13.15	5.32
20	2822	3051	62	230	11.50	5.39
20	3052	3457	104	406	20.30	5.12
20	3458	3558	26	101	5.05	5.15
20	2559	3558	262	1000	50.00	5.24

FIGURE C.1 – Résultat de la sliding window de la vidéo :Loading\_t02 avec taille de la fenêtre = 170 img



## Annexe D

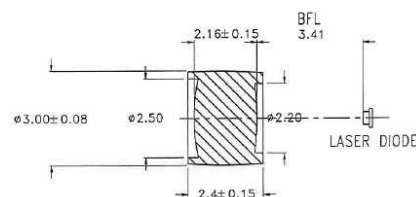
# Annexe : Divers

### D.1 SPÉCIFICATION LENTILLE

#### **LC3** ACRYLIC LASERDIODE COLLIMATOR

##### 3 Φ Collimating Lens for Laser Diode

1. Injection Molded Plastic (PMMA)
2. Aspheric Surfaces Design
3. Design Wavelength 650nm
4. F-Number 1.80
5. BFL 3.41mm
6. Clear Aperture 2.5mm
7. Lens Diagram



#### **ROITHNER LASERTECHNIK**

A-1040 WIEN, FLEISCHMANNGASSE 9  
 TEL: +43 -1- 586 52 43 FAX: +43 -1- 586 41 43  
 e-mail: [office@roithner-laser.com](mailto:office@roithner-laser.com) <http://www.roithner-laser.com>

## D.2 DOCUMENTATION FFT CMSIS

tes: Real FFT Functions

file:///home/sdi/Bureau/doxygene/html/group\_\_r\_f...

Real FFT Functions

Functions

void `arm_rfft_f32` (const arm\_rfft\_instance\_f32 \*S, float32\_t \*pSrc, float32\_t \*pDst)

Processing function for the floating-point RFFT/RIFFT. [More...](#)

arm\_status `arm_rfft_init_f32` (arm\_rfft\_instance\_f32 \*S, arm\_cfft\_radix4\_instance\_f32 \*S\_CFFT, uint32\_t fftLenReal, uint32\_t ifftFlagR, uint32\_t bitReverseFlag)

Initialization function for the floating-point RFFT/RIFFT. [More...](#)

arm\_status `arm_rfft_init_q15` (arm\_rfft\_instance\_q15 \*S, arm\_cfft\_radix4\_instance\_q15 \*S\_CFFT, uint32\_t fftLenReal, uint32\_t ifftFlagR, uint32\_t bitReverseFlag)

Initialization function for the Q15 RFFT/RIFFT. [More...](#)

arm\_status `arm_rfft_init_q31` (arm\_rfft\_instance\_q31 \*S, arm\_cfft\_radix4\_instance\_q31 \*S\_CFFT, uint32\_t fftLenReal, uint32\_t ifftFlagR, uint32\_t bitReverseFlag)

Initialization function for the Q31 RFFT/RIFFT. [More...](#)

void `arm_rfft_q15` (const arm\_rfft\_instance\_q15 \*S, q15\_t \*pSrc, q15\_t \*pDst)

Processing function for the Q15 RFFT/RIFFT. [More...](#)

void `arm_rfft_q31` (const arm\_rfft\_instance\_q31 \*S, q31\_t \*pSrc, q31\_t \*pDst)

Processing function for the Q31 RFFT/RIFFT. [More...](#)

Detailed Description

Complex FFT/IFFT typically assumes complex input and output. However many applications use real valued data in time domain. Real FFT/IFFT efficiently process real valued sequences with the advantage of requirement of low memory and with less complexity.

This set of functions implements Real Fast Fourier Transforms(RFFT) and Real Inverse Fast Fourier Transform(RIFFT) for Q15, Q31, and floating-point data types.

Algorithm:

Real Fast Fourier Transform:

Real FFT of N-point is calculated using CFFT of N/2-point and Split RFFT process as shown below figure.



Real Fast Fourier Transform

The RFFT functions operate on blocks of input and output data and each call to the function processes fftLenR samples through the transform. pSrc points to input array containing fftLenR values. pDst points to output array containing 2\*fftLenR values.

Input for real FFT is in the order of

```
{real[0], real[1], real[2], real[3], ...}
```

Output for real FFT is complex and are in the order of

```
{real[0], imag[0], real[1], imag[1], ...}
```

Real Inverse Fast Fourier Transform:

Real IFFT of N-point is calculated using Split RIFFT process and CFFT of N/2-point as shown below figure.



Real Inverse Fast Fourier Transform

The RIFFT functions operate on blocks of input and output data and each call to the function processes 2\*fftLenR samples through the transform. pSrc points to input array containing 2\*fftLenR values. pDst points to output array containing fftLenR values.

Input for real IFFT is complex and are in the order of

```
{real[0], imag[0], real[1], imag[1], ...}
```

Output for real IFFT is real and in the order of

```
{real[0], real[1], real[2], real[3], ...}
```

Lengths supported by the transform:

Real FFT/IFFT supports the lengths [128, 512, 2048], as it internally uses CFFT/CIFFT.

Instance Structure

A separate instance structure must be defined for each Instance but the twiddle factors can be reused. There are separate instance structure declarations for each of the 3 supported data types.

Initialization Functions

There is also an associated initialization function for each data type. The initialization function performs the following operations:

- Sets the values of the internal structure fields.
- Initializes twiddle factor tables.
- Initializes CFFT data structure fields.

Use of the initialization function is optional. However, if the initialization function is used, then the instance structure cannot be placed into a const data section. To place an instance structure into a const data section, the instance structure must be manually initialized. Manually initialize the instance structure as follows:

1 sur 4

01. 07. 14 10:48

HES-SO Valais / Jérémie Mayor - v1.0  
10 juillet 2014

98 / 100

tes: Real FFT Functions

file:///home/sdi/Bureau/doxygene/html/group\_\_r\_f...

```

arm_status arm_rfft_init_q15 ( arm_rfft_instance_q15 * S,
                             arm_cfft_radix4_instance_q15 * S_CFFT,
                             uint32_t fftLenReal,
                             uint32_t ifftFlagR,
                             uint32_t bitReverseFlag
                             )

```

Initialization function for the Q15 RFFT/RIFFT.

**Parameters**

[in, out] <b>*S</b>	points to an instance of the Q15 RFFT/RIFFT structure.
[in] <b>*S_CFFT</b>	points to an instance of the Q15 CFFT/CIFFT structure.
[in] <b>fftLenReal</b>	length of the FFT.
[in] <b>ifftFlagR</b>	flag that selects forward (ifftFlagR=0) or inverse (ifftFlagR=1) transform.
[in] <b>bitReverseFlag</b>	flag that enables (bitReverseFlag=1) or disables (bitReverseFlag=0) bit reversal of output.

**Returns**

The function returns ARM\_MATH\_SUCCESS if initialization is successful or ARM\_MATH\_ARGUMENT\_ERROR if fftLenReal is not a supported value.

**Description:**

The parameter fftLenReal Specifies length of RFFT/RIFFT Process. Supported FFT Lengths are 128, 512, 2048.

The parameter ifftFlagR controls whether a forward or inverse transform is computed. Set(=1) ifftFlagR to calculate RIFFT, otherwise RFFT is calculated.

The parameter bitReverseFlag controls whether output is in normal order or bit reversed order. Set(=1) bitReverseFlag for output to be in normal order otherwise output is in bit reversed order.

This function also initializes Twiddle factor table.

```

arm_status arm_rfft_init_q31 ( arm_rfft_instance_q31 * S,
                             arm_cfft_radix4_instance_q31 * S_CFFT,
                             uint32_t fftLenReal,
                             uint32_t ifftFlagR,
                             uint32_t bitReverseFlag
                             )

```

Initialization function for the Q31 RFFT/RIFFT.

**Parameters**

[in, out] <b>*S</b>	points to an instance of the Q31 RFFT/RIFFT structure.
[in, out] <b>*S_CFFT</b>	points to an instance of the Q31 CFFT/CIFFT structure.
[in] <b>fftLenReal</b>	length of the FFT.
[in] <b>ifftFlagR</b>	flag that selects forward (ifftFlagR=0) or inverse (ifftFlagR=1) transform.
[in] <b>bitReverseFlag</b>	flag that enables (bitReverseFlag=1) or disables (bitReverseFlag=0) bit reversal of output.

**Returns**

The function returns ARM\_MATH\_SUCCESS if initialization is successful or ARM\_MATH\_ARGUMENT\_ERROR if fftLenReal is not a supported value.

**Description:**

The parameter fftLenReal Specifies length of RFFT/RIFFT Process. Supported FFT Lengths are 128, 512, 2048.

The parameter ifftFlagR controls whether a forward or inverse transform is computed. Set(=1) ifftFlagR to calculate RIFFT, otherwise RFFT is calculated.

The parameter bitReverseFlag controls whether output is in normal order or bit reversed order. Set(=1) bitReverseFlag for output to be in normal order otherwise output is in bit reversed order.

This function also initializes Twiddle factor table.

3 sur 4

01. 07. 14 10:49

## D.3 PLANNING

Planning		Semaine 1 12.05.14	Semaine 2 19.05.14	Semaine 3 26.05.14	Semaine 4 02.06.14	Semaine 5 09.06.14	Semaine 6 16.06.14	Semaine 7 23.06.14	Semaine 8 30.06.14	Semaine 9 07.07.14
Installation environnement développement		x	x							
Compréhension et analyse des programmes existants		x	x							
Modif. du programme Qt (sélection zone)		x	x							
Modif. Prog uC, découpe zone + envoi USB			x							
Test rapidité (fréquence max de prise d'images)										
Prise d'images avec daphnie (sauv. SD + algo sur Matlab)										
Implémentation des algorithmes en C										
Test de l'algorithme C image des vidéos fournis										
Test de l'algorithme C image filmé depuis sys. Emb.										
Adaptation de l'algorithme C pour le microcontrôleur										
Test de l'algorithme du système embarqué										
Rapport										
Algo. Matlab sur image filmé depuis sys. Emb.										
Problème printf										
Problème librairie USB										
Problème fréquence de prise d'images insuffisante										
Test du fonctionnement de la librairie FFT de ARM										
Amélioration de l'algorithme Matlab										

 prévu  
x réalisé