

Rapport de stage : Acquisition et reconstruction  
de la carte de profondeur de la vidéo égocentrée  
pour l'asservissement d'une prothèse

DELMAS Rémi - Licence 3 MIAGE

August 21, 2015



8 Juin - 31 Juillet

# Contents

<b>1</b>	<b>Contexte de travail</b>	<b>6</b>
1.1	L'organisme d'accueil : le LaBRI . . . . .	6
1.2	L'équipe de travail : Image et Son . . . . .	7
1.3	Le projet : SuVIPP . . . . .	7
<b>2</b>	<b>Cahier des charges</b>	<b>8</b>
2.1	Positionnement du problème . . . . .	8
2.2	Planification du stage : les phases . . . . .	9
<b>3</b>	<b>Le contact avec les différentes équipes</b>	<b>9</b>
<b>4</b>	<b>Outils et méthodes utilisés</b>	<b>10</b>
4.1	Les caméras . . . . .	10
4.2	Le langage de programmation : C++11 . . . . .	11
4.2.1	Pourquoi le C++ ? . . . . .	11
4.2.2	Le choix de la norme C++11 . . . . .	11
4.3	La bibliothèque de manipulation d'image : OpenCV . . . . .	11
4.3.1	Présentation d'OpenCV . . . . .	11
4.3.2	Introduction a la notion de GPGPU . . . . .	12
4.3.3	OpenCV et CUDA . . . . .	12
4.4	Construction logitielle : CMake . . . . .	13
<b>5</b>	<b>Travail réalisé</b>	<b>13</b>
5.0.1	Pré-requis . . . . .	13
5.1	Présentation succincte de l'algorithme et des modules mis en place	14
5.2	Calibration des caméras . . . . .	15
5.2.1	Optique : Distorsion . . . . .	15
5.2.2	Théories concernant la correction des distorsions . . . . .	16
5.2.3	Correction des distorsions en pratique pour une caméra . . . . .	17
5.2.4	Le programme de correction . . . . .	18
5.3	La carte de disparité . . . . .	19
5.3.1	A quoi nous sert finalement la calibration ? La contrainte épipolaire . . . . .	19
5.3.2	L'algorithme de Block Matching . . . . .	20
5.3.3	L'algorithme Belief Propagation . . . . .	21
5.4	La temps réel et la carte de profondeur . . . . .	22
5.4.1	Mise en place du multithreading . . . . .	23
5.4.2	La carte de profondeur . . . . .	24
<b>6</b>	<b>Conclusion</b>	<b>24</b>
6.1	Bilan professionnel . . . . .	24
6.2	Bilan personnel . . . . .	24
<b>7</b>	<b>Annexes</b>	<b>27</b>

## List of Figures

1	LaBRI . . . . .	6
2	Robot Poppy . . . . .	7
3	Principe de fonctionnement . . . . .	8
4	Le système 3D GoPro et les lunettes Tobii . . . . .	9
5	Exemple de détection de visage avec OpenCV . . . . .	12
6	Illustration du wrapping de CUDA par OpenCV . . . . .	13
7	Organisation des modules . . . . .	14
8	Les lignes de ce plafond ne sont plus droites à cause du phénomène de distorsion radiale . . . . .	16
9	Capture de vidéo durant un processus de calibration avec la grille	17
10	Calibration du système stéréoscopique . . . . .	18
11	Géométrie épipolaire . . . . .	20
12	Carte de disparité, plus la couleur est sombre, plus l'objet est supposé loin. . . . .	21
13	Carte de disparité avec Belief Propagation . . . . .	22
14	Processus complémentaire à la carte de profondeur, la sélection de l'objet . . . . .	27
15	Positionnement des lunettes Tobii et des GoPro sur le patient . .	27

## Remerciements

Je tiens à remercier dans un premier temps **Jenny Benois-Pineau** qui, par la proposition de ce stage, m'a permis d'acquérir une bonne expérience du monde de la recherche et m'a beaucoup appris.

Je remercie également **Pierre-Marie Plans**, qui m'a encadré durant toute la durée de mon stage, Il a été présent tout au long de ce stage pour m'aider tout en m'offrant un chaleureux accueil.

Un grand merci également a **tous les stagiaires et doctorants** avec qui j'ai pu échanger au cours de ce stage. Merci pour votre bonne humeur et vos conseils.

Pour finir, je tiens a remercier **Julier Josseaume**, étudiant de MIAGE, pour le prêt de sa GoPro Black 3+, qui m'a été très utile dans les premières semaines de ce stage.

# Introduction

Les outils d'analyse et de reconnaissance de la vidéo égocentrée en provenance des dispositifs portés par des sujets ont beaucoup progressé ces dernières années. Ceci est dû à l'importance de l'exploration de l'humain dans un contexte médical, comportemental, d'assistance à l'handicap ou d'étude du geste sportif. Par ailleurs, les prothèses motorisées du membre supérieur se sont fortement développées.

Afin d'améliorer les mécanismes de contrôle de ces prothèses, l'utilisation de la vidéo égocentrée est une piste prometteuse. En effet, la vidéo enregistrée avec des lunettes de type Google Glass, et pouvant inclure un dispositif de suivi du regard, permet d'identifier la scène observée par le porteur de la prothèse ainsi que les mouvements de ses membres ou de sa prothèse. Ainsi la coordination des mouvements des porteurs de prothèses dans des tâches élémentaires telles que la saisie et la manipulation des objets peut être améliorée, grâce à l'intégration de cette information.

Dans le cadre de ce stage, j'ai été intégré à l'équipe Image et Son du LaBRI, qui a pour ambition, en s'appuyant sur les compétences des équipes<sup>1</sup> de l'INRIA et de l'INCIA, de développer des outils d'analyse des actions des sujets, tels la saisie des objets. Cette recherche, pionnière sur le plan national et naissante à l'international, permettra de mieux comprendre les mécanismes perception-action et d'utiliser cette information pour l'asservissement des prothèses. Le projet s'appuie également sur la plate-forme robotique Poppy, développée par l'équipe Flowers de l'Inria et utilisée en collaboration avec l'équipe INCIA.

Dans ce rapport, je décrirais donc les diverses équipes de chercheurs et ingénieurs prenant part au projet, en détaillant le rôle de l'équipe du LaBRI, ainsi que le mien. Nous verrons ensuite les outils et solutions que j'ai mis en place, en argumentant les choix d'implémentations importants.

---

<sup>1</sup>Les parties prenantes du projet seront décrites dans le premier chapitre de ce rapport.

# 1 Contexte de travail

## 1.1 L'organisme d'accueil : le LaBRI

Le Laboratoire Bordelais de Recherche en Informatique (LaBRI) est une unité mixte de recherche du CNRS rattaché à l'Université de Bordeaux et à Bordeaux INP. En mars 2015, il réunit près de 320 personnes, dont 113 enseignants chercheurs (Université de Bordeaux, Bordeaux INP), 37 chercheurs (CNRS, Inria), 22 personnels administratifs et techniques (Université de Bordeaux, Bordeaux INP, CNRS, Inria) et plus de 140 doctorants, post-doctorants et ingénieurs. Le soutien du Conseil Régional d'Aquitaine a été une des briques essentielles du développement du LaBRI, via le financement de l'extension du bâtiment (superficie totale de 5000 m<sup>2</sup>) et de ces équipements, ainsi que l'attribution de bourses de thèses et post-doctorant. Les chercheurs et enseignants-chercheurs du LaBRI participent à la formation dans différents établissements de plus de 1300 étudiants inscrits dans les spécialités informatiques. Les activités des membres du laboratoire concernent principalement des thèmes suivants :

- Combinatoire et Algorithmique
- **Image et Son**
- Méthodes Formelles
- Modèles et Algorithmes pour la Bioinformatique et la Visualisation d'informations
- Programmation, Réseaux et Systèmes
- Supports et Algorithmes pour les Applications Numériques Hautes Performances

Le LaBRI collabore de façon active sur les plans internationaux, européens et français, avec de nombreux laboratoires et entreprises, dont l'INRIA avec qui plus de 10 projets communs ont vu le jour, dont le projet SuVIPP.<sup>2</sup>



Figure 1: LaBRI

---

<sup>2</sup>Détails du projet à la fin du chapitre

## 1.2 L'équipe de travail : Image et Son

L'équipe Image et Son du LaBRI est la plus volumineuse du laboratoire et est elle même décomposable en plusieurs groupes de recherche :

- **Le groupe Analyse et Indexation Vidéo** (Estimation du mouvement, d'indexation et reconnaissance des scènes spatio-temporelles)
- Le groupe Document (Analyse d'images de documents)
- Le groupe MorphoBoid (Classification automatique d'images)
- Le groupe OS3D (Outils pour la Segmentation 3D)
- Le groupe PICTURA (Traitement de l'image et applications à l'imagerie médicale)

Mon stage s'est déroulé au sein du groupe d'Analyse et Indexation Vidéo, dans laquelle je travaillais en autonomie, dirigé par Jenny Benois-Pineau, l'auteur et co-auteur de plus de 120 articles dans des revues internationales et actes de congrès (également professeur à l'Université de Bordeaux). Forte de l'expérience acquise avec le projet Dem@care (reconnaissance de l'activité pour la détection de la démence), ayant abouti à la publication de plusieurs papiers de recherche sur l'analyse et le traitement de la vidéo égocentrée, l'équipe participe maintenant au projet SuVIPP<sup>3</sup>. J'ai vraiment senti l'intérêt de l'équipe sur ce sujet de recherche, chacun proposant des méthodes et astuces pour obtenir le meilleur des résultats, selon ses capacités et spécialités.

## 1.3 Le projet : SuVIPP

Le projet exploratoire SuVIPP a pour finalité l'asservissement d'une prothèse de bras, en collaboration avec l'Inria et l'INCIA, la partie robotique du projet étant gérée par les autres équipes (déplacement en 3 dimensions, saisies, rotation, équilibre, ...) via le robot Poppy (Test - Inria).

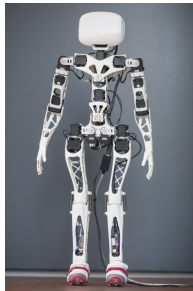


Figure 2: Robot Poppy

---

<sup>3</sup>Suppléance vidéo pour la sélection et l'atteinte d'objets avec une prothèse du bras

Les fonds de financements proviennent de L'INDEX Bordeaux<sup>4</sup> et du CNRS, à hauteur de 25000€. La demande de financement ayant été une des 18 retenues sur les 58 postulants.

## 2 Cahier des charges

### 2.1 Positionnement du problème

Afin que le patient puisse saisir un objet via la prothèse, l'objectif est ici de fournir des informations sur les objets d'intérêts (via traitement des images des lunettes Tobii, un eye-tracker)<sup>5</sup> et le calcul de la distance entre la caméra et l'objet (via un système stéréoscopique de deux GoPro Black 3+) le tout en temps réel. C'est un défi technique colossale. Ma tâche dans ce processus fût de mettre en place le système permettant le calcul de la carte de profondeur (en rouge sur le schéma ci-dessous), de la proposition d'une méthode de traitement et de la l'architecture logitielle à l'implémentation.

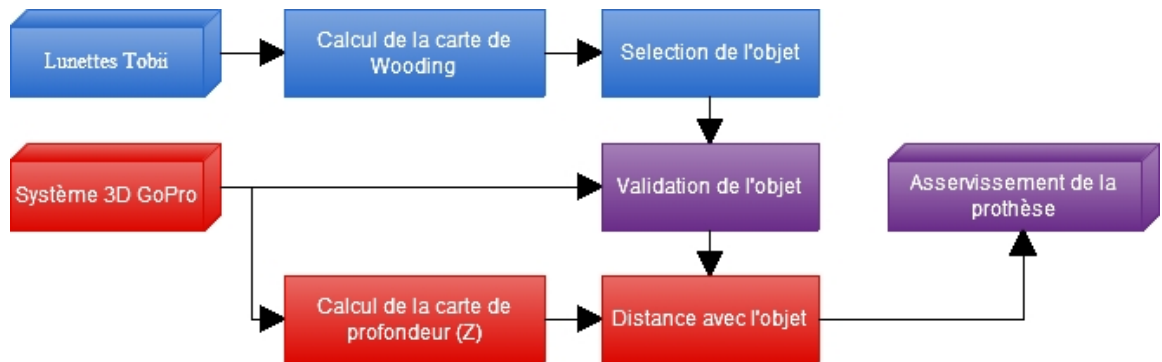


Figure 3: Principe de fonctionnement

La solution a fournir devait se presenter sous la forme d'un executable destiné à une plateforme Linux, permettant d'obtenir la carte de profondeur (où chaque pixel représente la distance Z avec l'objet) l'une scène a partir dy système de GoPro à un instant donné. Dans le futur, l'application est destinée à devenir une bibliothèque pour permettre la fusion des différents blocs des travaux réalisés.

<sup>4</sup>Initiative d'excellence de l'Université de Bordeaux

<sup>5</sup>Images d'illustration disponible en annexe





Figure 4: Le système 3D GoPro et les lunettes Tobii

En réalité peu de contraintes me furent imposées durant le déroulement de mon stage, et cette liberté dans mes choix d'implémentation est vraiment intéressante pour un étudiant, car on comprend ainsi bien tous les choix à effectuer lors du développement d'un logiciel, et la portée de tous ses choix, bon ou mauvais. C'est un défi imposant mais passionnant.

## 2.2 Planification du stage : les phases

Contrairement à la plupart des stagiaires qui partent d'un système existant et apporte des améliorations/corrections sur un ensemble complexe, la conception en amont de la phase de développement impose une méthodologie différente. Mon exercice c'est articulé autour de différentes phases, parfois rétroactive :

1. Recherche sur le thème de la vision stéréoscopique et de la reconstruction 3D
2. Synthétisation des informations récolté et proposition d'une méthode cohérente avec la problématique du temps réel
3. Auto-formation sur les méthodes associé au temps-réel (multithreading)
4. Elaboration de l'architecture logitielle
5. Préparation du système (Installation des bibliothèques, drivers, ...)
6. Implémentation, Tests, Debug
7. Si soucis, retour au point 4

## 3 Le contact avec les différentes équipes

Ce projet étant pluridisciplinaire, un contact permanent entre les parties prenantes est nécessaires afin de bien orienter nos travaux. J'ai ainsi pût assister à plusieurs

réunions avec les équipes de l'INCIA dans leurs locaux, afin de joindre nos réflexions et recueillir leurs impressions et avis après la présentation de nos travaux. Les commentaires furent toujours constructifs lors de ces réunions.

J'ai également eu des échanges plus réguliers avec Joel Ortiz, stagiaire en robotique à l'INRIA et travaillant sur le robot Poppy. Responsable de l'implémentation du mouvement du bras mécanique et des actions de ce dernier, mon rôle était de lui fournir le logiciel permettant d'obtenir les cartes de profondeur pour ces travaux. Nous avons donc convenu des normes à adopter et du format de sortie.

En plus des réunions avec les autres équipes, j'ai réalisé une présentation sur la reconstruction 3D au LaBRI pour le Dr. Mark Latash (Penn State University, USA) dans le cadre de la présentation des activités de l'équipe d'Analyse et Indexation Vidéo.

## 4 Outils et méthodes utilisés

L'intégralité de mon stage c'est déroulé sur un ordinateur muni d'Ubuntu 14.04<sup>6</sup>.

### 4.1 Les caméras

Le choix d'un bon système de caméra est un élément essentiel pour la réussite ou non du projet, la précision des mesures étant en effet directement impacté par la qualité de l'image et la fréquence des captures. Le choix d'un système c'est donc arrêté sur la combinaison de deux GoPro Hero 3+ Black, la qualité de l'image étant vraiment bonne, la résolution et le framerate étant modifiable (selon le niveau de performance atteignable par le programme, on peut donc modifier les paramètres en amont). En plus des caméras, l'achat d'un kit 3D GoPro fut effectué, nous permettant une synchronisation immédiate des GoPro tout en permettant une fixation et un confort d'utilisation accru.

Ce système n'est en revanche arrivé que vers la milieu de mon stage, les tests furent dans un premier temps effectués avec un système composée d'une GoPro Hero 3 White et d'une GoPro Hero 3+ Black. Ces deux systèmes ayant des focales différentes, et n'étant pas synchronisable par défaut on ralentit la production logicielle en début de stage, sans être trop handicapant (synchronisation des vidéos via un logiciel de montage).

Mon ordinateur ne possédant pas de carte WI-FI, la récupération en direct du flux du système ne fut pas possible. C'est un point plutôt dérangerant pour certaines étapes du développement, l'enregistrement de vidéo pour les tests des différents modules étant parfois longs et devant être répétés plusieurs fois par jours.

---

<sup>6</sup>Hormis durant la première semaine de stage, où je travaillais sous Ubuntu 12.04

## 4.2 Le langage de programmation : C++11

Le langage C++ est une «amélioration» du langage C (le langage C a été mis au point par M.Ritchie et B.W.Kernighan au début des années 70). Bjarne Stroustrup, un ingénieur considéré comme l’inventeur du C++, a en effet décidé d’ajouter au langage C les propriétés de l’approche orientée objet. C++ est l’un des langages de programmation les plus populaires actuellement, étant largement répandu dans quasiment tous les secteurs de l’informatique.

### 4.2.1 Pourquoi le C++ ?

Le C++ permet, comme son illustre ancêtre le C, une gestion fine de la mémoire et de bonnes performances temporelles pour un programme ainsi d’une bonne portabilité. Ce langage est une référence dans le domaine des applications temps réel et particulièrement dans le monde de l’image<sup>7</sup>. La principale bibliothèque utilisée dans le cadre de mon stage, OpenCV est écrite en C++.

### 4.2.2 Le choix de la norme C++11

Si le choix du langage ne relève pas de moi, bien que je l’approuve totalement, j’ai en revanche fait le choix d’utiliser la norme C++ 11. Il s’agit d’une mise à jour de la norme C++03 qui introduit plusieurs nouveautés au langage initial, tout en étant totalement compatible avec la norme précédente, comme les fonctions lambda et une syntaxe plus intuitive. C++11 introduit de nouvelles fonctionnalités à la bibliothèque standard du C++, notamment une classe pour la gestion des threads.

## 4.3 La bibliothèque de manipulation d’image : OpenCV

### 4.3.1 Présentation d’OpenCV

OpenCV<sup>8</sup> est une bibliothèque graphique libre, initialement développée par Intel, spécialisée dans le traitement d’images en temps réel. OpenCV est aujourd’hui développée, maintenue, documentée et utilisée par une communauté de plus de 40 000 membres actifs. C’est la bibliothèque de référence pour la vision par ordinateur, aussi bien dans le monde de la recherche que celui de l’industrie avec plus de 2000 algorithmes implémentés.

---

<sup>7</sup>C++ est énormément utilisé au LaBRI, ainsi que Python pour le prototypage notamment.

<sup>8</sup>Open Computer Vision



Figure 5: Exemple de détection de visage avec OpenCV

C'est sans doute l'outil le plus important que j'ai utilisé durant mon stage, une grande partie des traitements nécessaires étant implémentés par OpenCV, avec d'excellent temps d'exécutions.

#### 4.3.2 Introduction a la notion de GPGPU

Le GPGPU<sup>9</sup> désigne un calcul générique sur une carte graphique afin d'effectuer des calculs massivement parallèle (Un GPU possédant des centaines de coeurs). Le traitement d'image est un domaine qui se prête très bien à ce genre d'optimisation matériel, un traitement pouvant parfois être appliqué sur plusieurs pixels en simultané. Les gains sont souvent conséquents si le cas d'application s'y prête bien, par exemple, la société BNP Paribas a remplacé 500 CPU par 2 GPU pour des applications liées à la finance et à obtenu les résultats suivants :

- Réduction par 190 de l'ensemble de la consommation électrique donc diminution de l'impact environnemental.
- Temps de réponses divisés par 15.
- Réductions des coûts.

#### 4.3.3 OpenCV et CUDA

OpenCV nous fourni un module GPU, implémentant des algorithmes en CUDA, l'architecture de traitement parallèle développé par NVIDIA. Ce module nous permet un gain important dans le temps d'implémentation, l'appel étant quasiment transparent depuis OpenCV, tout en offrant un énorme gain de performance (7 fois plus rapide dans notre cas !).

---

<sup>9</sup>general-purpose processing on graphics processing units

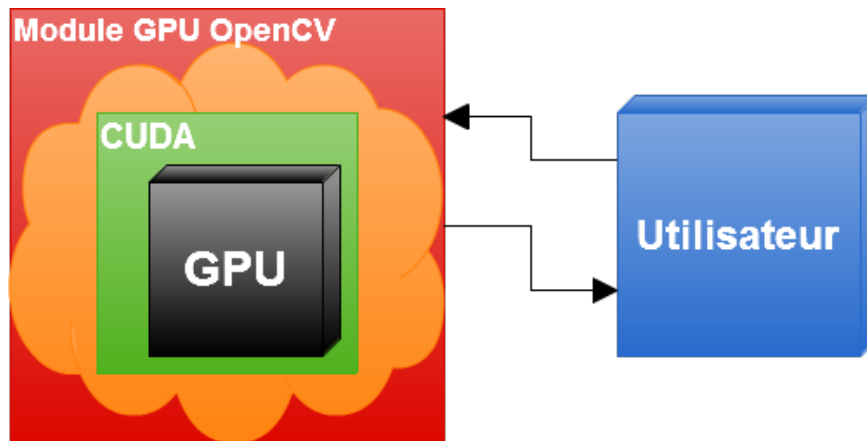


Figure 6: Illustration du wrapping de CUDA par OpenCV

#### 4.4 Construction logitielle : CMake

J'ai fait le choix d'utiliser CMake pour la construction logitielle, un moteur de production controlé par des fichiers de configuration, dont le but est la génération de fichiers de construction (Makefile pour un systeme Unix). Cet outil permet ainsi une compilation rapide et intuitive pour l'utilisateur, en plus de proposer des mécanismes de tests automatiques si nécessaire.

### 5 Travail réalisé

Dans un premier temps, nous introduirons les différentes étapes de l'algorithme, en détaillant progressivement ces dernieres sans pour autant être trop technique, les algorithmes composant ces différentes étapes étant généralement très complet.

#### 5.0.1 Pré-requis

Avant de rentrer dans le vif du sujet du stage, certains programmes/paramétrages sont nécessaire sur le système de l'utilisateur à savoir :

- Drivers NVIDIA à jour ( $\geq 210.56$ ) avec NVIDIA CUDA correctement installé ( $\geq 6$ ) (GPU)
- OpenCV 3.0 & FFmpeg
- GCC à jour ( $\geq 4.7$ , C++11)
- Paths vers les bibliothèques OpenCV et CUDA corrects

## 5.1 Présentation succincte de l'algorithme et des modules mis en place

Afin d'obtenir une carte de profondeur on procède comme suit :

1. Calibration du système stéréoscopique, on obtient alors un système sans distorsion, parfait pour l'application de la géométrie épipolaire<sup>10</sup> et des algorithmes de matching
2. Génération de la carte de disparité en temps réel, via des optimisations multithread et l'utilisation du GPU.
3. Génération de la carte de profondeur, grace à l'étape de calibration.

L'organisation des modules mis en place est assez simple, le projet n'étant pour l'instant pas très volumineux.

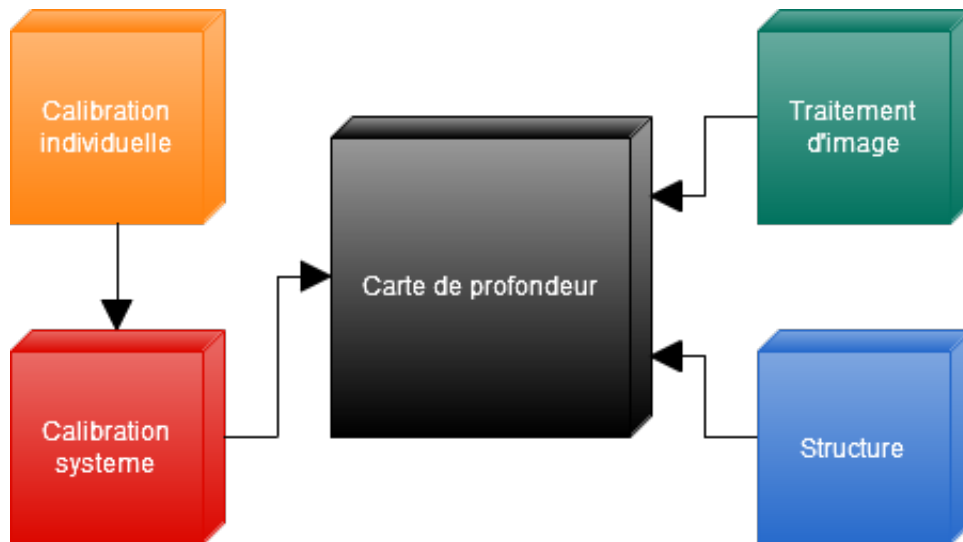


Figure 7: Organisation des modules

- Calibration individuelle : Un module plutôt autonome donc le but est de fournir le fichier de calibration d'une caméra au module de calibration du système.
- Calibration systeme : Le module le plus important implémentant Stereo-Calibration, la classe stockant les données résultat de la calibration, nécessaires notamment pour la correction des distorsions.
- Traitement d'image : Regroupe les divers traitements d'images, notamment la fonction générant la carte de profondeur.

---

<sup>10</sup>Pas de panique, les explications de toutes ces notions arrivent plus loin !

- Structure : N'implémente pour l'instant qu'une seule structure, la file thread-safe.
- Carte de profondeur : le "main" du projet, mettant en place les threads et mettant en place les étapes 2 et 3 de l'algorithme.

## 5.2 Calibration des caméras

Il s'agit d'une étape essentielle de l'algorithme, sur laquelle des dizaines d'heures furent nécessaires. En plus de corriger la distorsion, son rôle va être de fournir la matrice fondamentale du système optique et la matrice de reprojection, des données importantes, notamment pour la reprojection des points afin de déduire la profondeur d'un pixel dans le monde réel. Cette calibration n'est à effectuer d'une seule fois pour un système donné.

### 5.2.1 Optique : Distorsion

La distorsion radiale est une alteration d'image que l'on rencontre sur des dispositifs d'acquisition d'images tel qu'un appareil photo ou une camera. Cette distorsion a tendance à arrondir les bords de l'image et est particulièrement visible sur les objectifs à courte focale. Notre système à base de caméra GoPro est particulièrement susceptible à ce phénomène (la focale d'une de nos caméra est de 21mm, soit une très courte focale), or nous sommes à la recherche d'un système optique parfait pour le bon fonctionnement des prochaines étapes de l'algorithme<sup>11</sup>.

---

<sup>11</sup>Voir le prochain sous-chapitre sur la carte de disparité

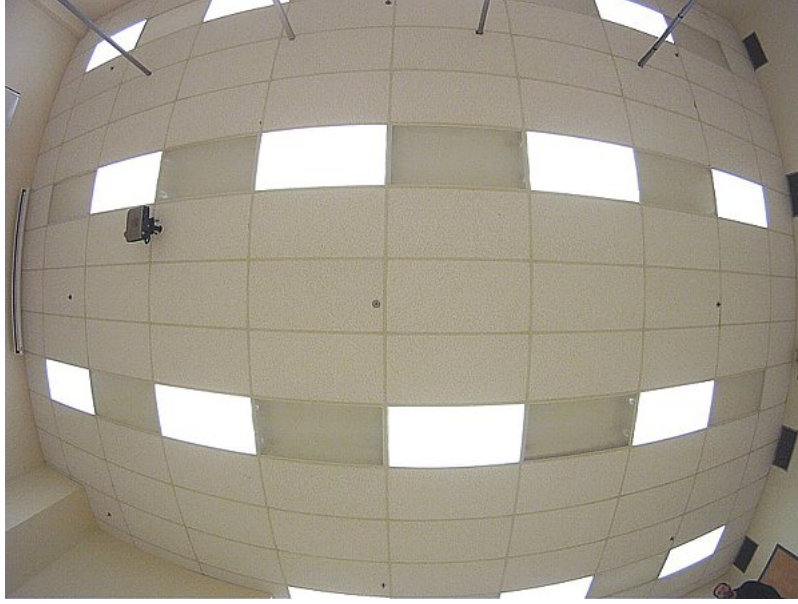


Figure 8: Les lignes de ce plafond ne sont plus droites à cause du phénomène de distorsion radiale

### 5.2.2 Théories concernant la correction des distorsions

Ces distorsion sont de deux type, les distorsions radiales et tangentielle (sorte de rotation autour du centre de l'image) et peuvent être rectifier par les formules suivantes :

$$\begin{aligned}
 x_{\text{corrigé-radiale}} &= x(1 + k_1r^2 + k_2r^4 + k_3r^6) \\
 y_{\text{corrigé-radiale}} &= y(1 + k_1r^2 + k_2r^4 + k_3r^6) \\
 x_{\text{corrigé-tangentielle}} &= x + [2p_1xy + p_2(r^2 + 2x^2)] \\
 y_{\text{corrigé-tangentielle}} &= y + [p_1(r^2 + 2y^2) + 2p_2xy]
 \end{aligned}$$

Un des buts de la calibrations est donc d'obtenir ces cinq paramètres de distorsion, qui dans OpenCV sont présentés comme une matrice a une ligne et cinq colonnes :

$$Distortion_{coefficients} = (k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3)$$

Ensuite, pour la conversion d'unité, on utilise la formule suivante :

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$



Avec  $f_x$  et  $f_y$  les longueurs focales du système et  $c_x$  et  $c_y$  les centres optiques exprimés en pixels, la matrice composée de ces paramètres se nommant la matrice de caméra. Le but de la calibration est de déduire ces deux matrices.

### 5.2.3 Correction des distorsions en pratique pour une caméra

Pour faire cette correction, le but est de prendre des captures d'une grille de calibration devant la caméra dans différentes positions et distances. Il nous "suffira" ensuite de faire appel à OpenCV pour rechercher le pattern de la grille dans l'image.

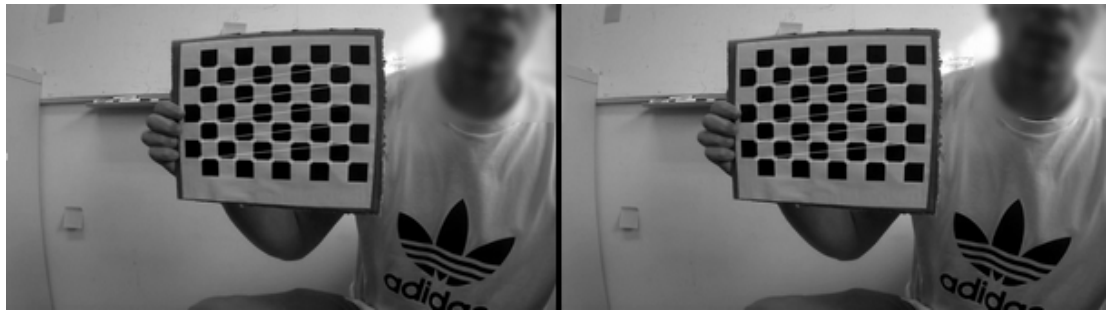


Figure 9: Capture de vidéo durant un processus de calibration avec la grille

Chaque nouveau pattern est une nouvelle équation, et pour résoudre l'équation il nous faut un minimum de capture (un dizaine) afin de former un bon système d'équation. Des captures similaires (mêmes positions/distances) entraîneront un système mal posé et donc un échec de la calibration. Après avoir mis en mémoire suffisamment de pattern, il faut ensuite faire appel à la méthode de calibration en elle-même qui se charge de la résolution du système :

```
double rms = calibrateCamera(objectPoints,
                             imagePoints, imageSize,
                             cameraMatrix, distortionCoeffs
                             ,...);
```

Sans partir dans des détails trop techniques (notamment sur `objectPoints` et `imagePoints` désignant les sauvegardes et reprojections des patterns), nous obtenons ainsi la matrice de caméra et la matrice de distorsion.

Cette étape de la calibration n'est à réaliser qu'une fois, et on peut mesurer la précision de la calibration via la valeur RMS<sup>12</sup> (Ma meilleure calibration a un RMS de 0.3 ce qui est acceptable, une bonne valeur étant inférieure à 1).

Après avoir sérialisés les instances de calibrations obtenues pour les deux caméras (au format YAML<sup>13</sup>), on réalise ensuite la calibration du système

<sup>12</sup>Erreur de reprojection

<sup>13</sup>YAML Ain't Markup Language, semblable à du XML

complet en utilisant les instances sérialisés (afin d'obtenir la matrice 4x4 de disparité-à-profondeur qui nous servira à obtenir la carte de profondeur) .

Le schéma suivant résume bien les explications précédentes :

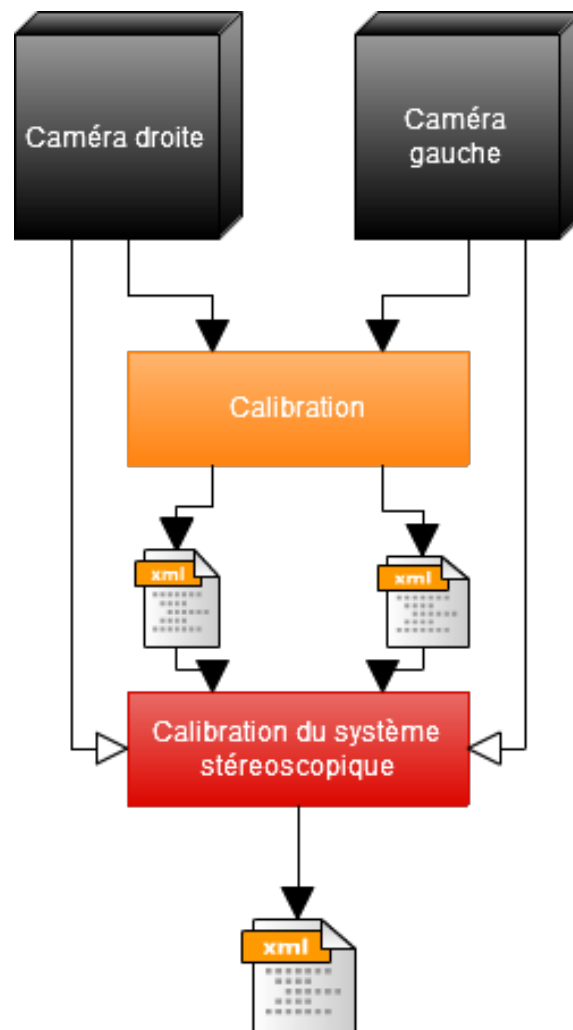


Figure 10: Calibration du système stéréoscopique

#### 5.2.4 Le programme de correction

Cette phase se compose donc de deux calibrations, et donc de deux exécution, l'un pour la calibration d'une caméra, l'autre pour celle du système complet.

La fonction *usage* illustre bien son fonctionnement et est extrait des sources du programme de calibration du système.

```
void usage()
{
    cout << "\nUsage : ./Calibration -w <Calibration_board_weight> -h <Calibration_board_height>\n"
    << "\t-sz <Square_size> -vids <Left_video> <Right_video>\n"
    << "\t [Option] -single <Single_camera_calibration> HIGHLY RECOMMENDED\n"
    << "\t [Option] -singleDiff <Left_camera_calibration> <Right_camera_calibration>\n"
    << "\t [Option] -zt <CALIB_ZERO_TANGENT_DIST> -p <CALIB_FIX_PRINCIPAL_POINT>\n"
    << "\t [Option] -u <Video_test_file_left> <Video_test_file_right>\n"
    << "\t [Output] stereo.yml\n"
    << " \n";
}
```

La calibration peut donc s'effectuer avec ou sans fichier de calibration des caméras, avec le même fichier pour les deux caméras ou deux fichiers différents. Il est cependant recommandé d'utiliser l'étape intermédiaire de la calibration indépendante de caméra sous peine de moins bons résultats. On utilise deux vidéos synchronisées et on cherche le pattern dans les deux images, en cas de recherche positive dans les deux cas, on affiche l'image à l'utilisateur et on attend la pression d'une touche (après vérification qu'il ne s'agit pas d'un faux positif en cas d'image flouée par le mouvement par exemple) pour enregistrer le pattern ou non.

La calibration effectuée, on peut maintenant calculer la carte de disparité.

### 5.3 La carte de disparité

La carte de disparité obtenue à partir d'un système stéréoscopique désigne une image qui contient l'information sur la disparité (le décalage) sur chaque pixel des deux images, en bref, on va évaluer la distance entre le point  $x$  d'une des caméras et son équivalent  $x'$  sur l'autre caméra. Pour ce faire on utilise un système de deux caméras dans le but de retrouver l'élément perdu dans l'usage d'un appareil photographique ou d'une caméra : la profondeur.

#### 5.3.1 A quoi nous sert finalement la calibration ? La contrainte épipolaire

En présence d'un système calibré, nous sommes dans la situation suivante :

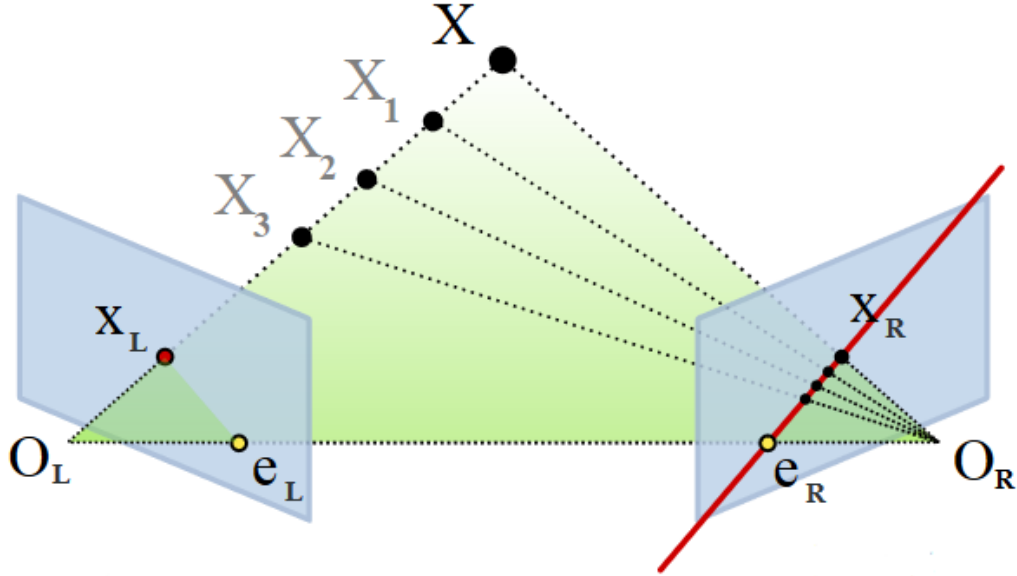


Figure 11: Géométrie épipolaire

Le correspondant d'un point d'une image sur l'autre, quelle que soit sa distance avec le centre optique, sera situé sur la droite en rouge sur le schéma. En effet si le point  $X$  se projette sur l'image de gauche en  $X_L$ , le point objet correspondant ne peut être que sur la droite  $XX_L$ . En projetant l'ensemble des points de cette droite sur le plan image de droite, on obtiens la ligne epipolaire ou le point correspondant peut possiblement être.

En cas de distorsion sur les images, la ligne n'est plus droite. La contrainte épipolaire, c'est à dire que le correspondant se trouve sur la droite mise en évidence précédemment, n'est alors pas respectée et l'on ne peut limiter la recherche du correspondant sur la ligne épipolaire et le coup de l'algorithme de recherche explose. L'étape de calibration est donc indispensable pour le calcul de la carte de disparité. OpenCV implémente deux algorithme optimisé GPU permettant ce calcul, à savoir l'algorithme de Block Matching et celui de Belief Propagation.

### 5.3.2 L'algorithme de Block Matching

La bibliotheque OpenCV implémente une version sur GPU d'un algorithme basé sur l'algorithme de SAD<sup>14</sup>, un algorithme très connu et notamment utilisé dans la compression d'image et permet la mesure de la similarité entre des blocs d'images. On va alors comparer un block d'une des deux images (de taille variable, par exemple de  $15 \times 15$  pixel) avec les positions vraisemblables du block

<sup>14</sup>Sum of Absolute Differences

sur la seconde image. On effectue a chaque itération la somme de la différence des valeurs des pixels associés. Par exemple pour un block de  $3 \times 3$  si les valeurs des pixels sont entre 0 et 2:

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 0 & 1 \\ 2 & 1 & 2 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 2 & 1 \\ 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} |1-1| & |0-2| & |0-1| \\ |2-2| & |0-2| & |1-1| \\ |2-0| & |1-0| & |2-0| \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 2 & 1 \\ 0 & 2 & 0 \\ 2 & 1 & 2 \end{bmatrix} \rightarrow 10$$

Le block avec le quel le résultat est le plus faible est alors sélectionné comme correspondant. L'avantage de cet algorithme est principalement son coup, relativement rapide en comparaison à Belief propagation, les résultats sont en revanche moins précis.



Figure 12: Carte de disparité, plus la couleur est sombre, plus l'objet est supposé loin.

Les résultats sont plutôt satisfaisant, hormis pour la zone totalement noire, résultant de l'échec de l'algorithme, les résultats de blocks étant surrement trop similaires sur cette zone monochrome.

### 5.3.3 L'algorithme Belief Propagation

On ne détaillerai pas ici le fonctionnement de cet algorithme très complexe, néanmoins une petite présentation est nécessaire. L'algorithme du Belief Propagation est un algorithme dont le fonctionnement est fondé sur l'utilisation de probabilités. Peu rapide et très gourmand en mémoire, cet algorithme est cependant réputé optimal au sens du maximum de vraisemblance, c'est-à-dire que les valeurs de decision sont les meilleures que l'on puisse trouver.

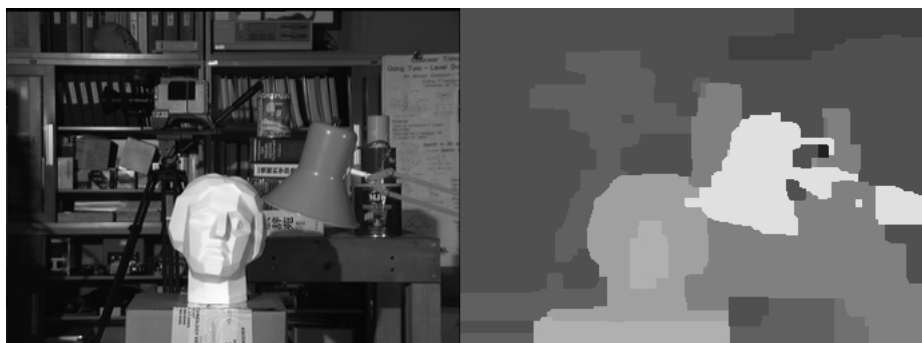


Figure 13: Carte de disparité avec Belief Propagation

L'implémentation de cet algorithme dans OpenCV 3.0 n'a pas donné les résultats espérés. L'image précédente proviens de l'utilisation de ce même algorithme implémenté par Felzenszwalb et dont le code est disponible librement. Par manque de temps, je n'ai pas pu fournir un wrap de ce code pour OpenCV, bien son utilisation sans optimisation GPU en temps réel est peu vraisemblable. Les résultats sont néanmoins très précis.

#### 5.4 La temps réel et la carte de profondeur

Le traitement d'image est une opération lourde dans le monde de l'informatique, et la combinaison entre opération lourdes et temps réel impose souvent l'adaptation du code au matériel. Ce stage m'a donc permis d'acquérir les notions fondamentales de la programmation parallèle, un sujet qui m'a passionné.

#### 5.4.1 Mise en place du multithreading

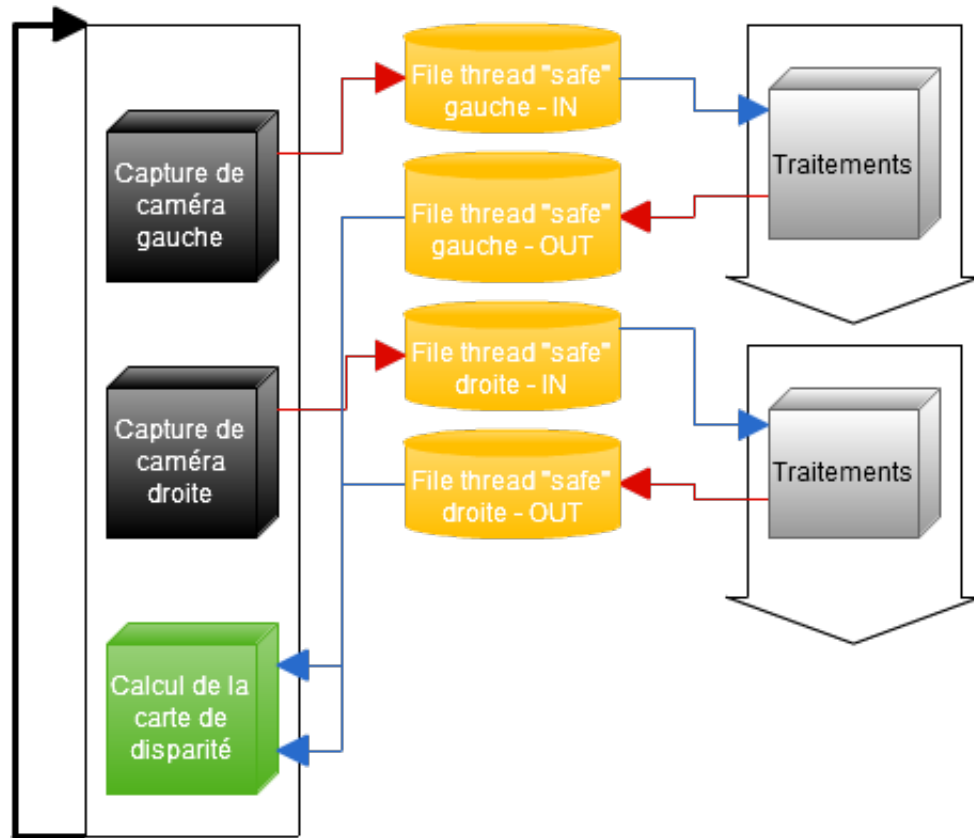


Table 1: La solution multithread mise en place

Le schéma ci-dessus présente la solution implémentée, discutons maintenant ce choix. Après capture du flux sur les deux vidéos, on enfile ces deux images (copie de surface) dans la file IN correspondante, puis des threads lancés préalablement, appliqueront les traitements nécessaires (filtres, correction des distorsions,...) et enfileront ensuite les images corrigées dans la file OUT correspondante, et les deux piles de sorties seront défilées lors de l'appel à lors du calcul de la carte de disparité.

J'ai utilisé la bibliothèque *thread* de la norme C++11 pour le contrôle des objets du même nom, la syntaxe étant relativement intuitive et puissante dans cette bibliothèque. Une classe File "safe" fût également nécessaire pour ne pas avoir d'accès concurrents, et joue également le rôle de verrou, par exemple, si la file IN gauche est vide et qu'un thread tente de défiler cette dernière, il

sera mis en attente le temps qu'une nouvelle valeur soit introduite<sup>15</sup>. Cette solution présente également un avantage en cas d'évolution nécessitant l'ajout de traitement sur les images initiales lues, l'ajout d'opérations étant par ce procédé moindres.

Nous obtenons une moyenne de 25 images par seconde pour des images en 720p avec un masque de 15x15 pour l'algorithme de Block Matching (avec affichage), ce qui est satisfaisant.

#### 5.4.2 La carte de profondeur

Le passage de la carte de disparité à la carte de profondeur se fait par une méthode d'OpenCV utilisant la matrice de reprojection. Les premiers résultats sont encourageants mais pour l'instant insuffisants (Plusieurs centimètres d'écart entre la distance réelle et la distance calculée). La sortie se fait dans un fichier texte contenant une matrice correspondant à la résolution des images d'entrée conformément à ce qui fût convenu avec Joel Ortiz.

## 6 Conclusion

### 6.1 Bilan professionnel

L'objectif fixé au début du stage, qui était donc de fournir une carte de profondeur à la fin de mon stage est rempli. Cependant le résultat est je pense encore améliorable, autant sur le plan de la qualité des résultats que sur la vitesse d'exécution. Cependant pour un temps de développement aussi court, le résultat me paraît acceptable, les modules de calibrations et la structure du logiciel étant en place. Je ne suis pas en revanche sûr que dans mes résultats soient exploitable dans l'immédiat. Le projet SuVIPP est très jeune mais me semble sur une bonne dynamique d'évolution, bien qu'une grosse partie travail reste à faire, notamment dans le domaine de la vision par ordinateur que j'ai découvert durant ce stage.

### 6.2 Bilan personnel

Je n'avais que peu d'expérience dans le domaine du traitement de l'image avant ce stage et j'ai énormément appris. Ce stage m'a permis de me faire une première idée du monde de la recherche qui m'intéresse depuis toujours, bien que j'ai conscience que mes études en MIAE m'orientent plus vers le monde de l'entreprise. Bien qu'ayant déjà une bonne expérience du C++ grâce à plusieurs ouvrages et projets, personnel et universitaire, j'ai également acquis de nouvelles compétences. De plus, la tâche très spécifique qui m'a été confiée m'a permis d'acquérir des compétences beaucoup plus poussées dans ce domaine. J'ai bien sûr été confrontée aux difficultés du travail en autonomie mais j'ai aussi pu apprécier la liberté dans la prise de décisions.

---

<sup>15</sup>Le code source de mon template est disponible en annexe



## Glossaire

- Vidéo égocentrée : Vidéo essayant de reproduire les caractéristiques de la vision humaine, comme par exemple les vidéos fournies par les Googles Glass.
- focale : La focale d'un objectif, c'est la distance en millimètre qui existe entre la surface sensible (film ou capteur numérique) et le centre optique de l'objectif, lorsque la mise au point a été faite sur un sujet éloigné.
- C++ : Language de programmation orienté objet.
- GPGPU : General-purpose Processing on Graphics Processing Units, Désigne l'utilisation d'un processeur graphique pour des traitements détournés.
- Thread : Processus léger, Un processus peut contenir un ou plusieurs thread (applications dites multi-threadées) s'exécutant en quasi-simultanéité ou simultanément sur les processeurs multi-coeurs. A la différence des processus, les threads partagent le même espace mémoire protégé, les mêmes ressources et le même espace mémoire.
- CUDA : CUDA (Compute Unified Device Architecture) est la technologie de GPGPU de NVIDIA.
- Carte de disparité : Désigne une image qui contient l'information sur la disparité (le décalage) sur chaque pixel des deux images.
- File : Structure de données basée sur le principe du Premier entré, premier sorti, en anglais FIFO.

## References

- [1] Documentation d'OpenCV
- [2] Multithreading en C++ 11
- [3] La géométrie épipolaire
- [4] Article sur le GPGPU
- [5] Felzenszwalb, P. and Huttenlocher, D. (2004). Efficient belief propagation for early vision. In Proc. CVPR , volume 1, pages 261–268.
- [6] Learning OpenCV Computer Vision in C++ with the OpenCV Library By Adrian Kaehler, Gary Bradski.
- [7] Le langage C++,3e édition, Jesse Liberty, Siddhartha Rao, Bradley Jones.

## 7 Annexes

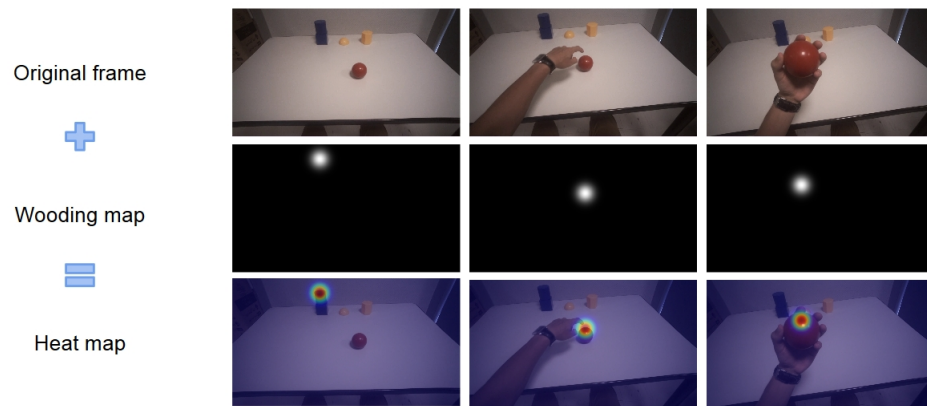


Figure 14: Processus complémentaire à la carte de profondeur, la sélection de l'objet



Figure 15: Positionnement des lunettes Tobii et des GoPro sur le patient

Code source de la file "Safe"

```

#ifndef SAFE_QUEUE
#define SAFE_QUEUE
#include <queue>
#include <mutex>
#include <condition_variable>
using namespace std;
/* C++11 thread safe queue template*/
template <class T> class SafeQueue
{
public:
    // 0 : unlimited
    SafeQueue(unsigned int max=0): m_queue(),
                                   m_mutex(),
                                   m_cv(),
                                   m_max(max)
    {}
    bool enqueue(const T &t);
    T dequeue();
    bool empty()
    {
        lock_guard<mutex> lock(m_mutex);
        return m_queue.empty();
    }
    int size()
    {
        lock_guard<mutex> lock(m_mutex);
        return m_queue.size();
    }

private:
    queue<T> m_queue;
    mutex m_mutex;
    condition_variable m_cv;
    unsigned int m_max;
};

template <class T>
bool SafeQueue<T>::enqueue(const T &t)
{
    /* Lock thread*/
    lock_guard<mutex> lock(m_mutex);
    if (m_max != 0 && m_queue.size() >= m_max)
        /*queue is full*/
        return false;
    m_queue.emplace(t);
    /* ok - another thread can dequeue*/

```

```

        //cout << m_queue.size() << "/" << m_max << endl;
        m_cv.notify_one();
        return true;
    }

template <class T>
T SafeQueue<T>::dequeue()
{
    /* Lock thread */
    unique_lock<mutex> lock(m_mutex);
    /* Our condition to dequeue -> m_queue not empty*/
    m_cv.wait(lock, [this]() { return !m_queue.empty(); });
    /* Get next elem*/
    T val = m_queue.front();
    m_queue.pop();
    return val;
}

#endif

```